21MCA24C1: ADVANCE SOFTWARE ENGINEERING

UNIT-II

Agile Software Development: Basics and Fundamentals of Agile Process Methods, Values of Agile, Principles of Agile, stakeholders, Challenges.

Agile and Scrum Principles: Agile Manifesto, Twelve Practices of Extreme Programming (XP), Scrum Practices, Applying Scrum. Need of scrum, working of scrum, advanced Scrum
Applications, Scrum and the Organization, scrum values.

Agile Requirements: User Stories, Backlog Management. **Agile Architecture:** Feature Driven Development. **Agile Risk Management:** Risk and Quality Assurance, Agile Tools.

Agile Testing: Agile Testing Techniques, Test-Driven Development, User Acceptance Test

MODULE 1:

Agile Software Development:

Agile software development is an iterative approach to software development that emphasizes collaboration, flexibility, and customer satisfaction. It is a set of values and principles outlined in the Agile Manifesto that prioritize individuals and interactions, working software, customer collaboration, and responding to change over following rigid processes and plans.

Agile software development is based on the concept of sprints or iterations, which involve breaking down a project into smaller, more manageable pieces of work. This allows teams to quickly deliver working software and gather feedback from customers, which can then be used to inform future development efforts. Agile also emphasizes the importance of continuous improvement, with teams regularly reflecting on their processes and making adjustments as necessary.

There are several different methodologies that fall under the umbrella of Agile software development, including Scrum, Kanban, Extreme Programming (XP), and Lean Software Development. Each methodology has its own unique set of practices and principles, but they all share a common focus on collaboration, flexibility, and responsiveness to change.

Basics and Fundamentals of Agile Process Methods,

Agile is a project management approach developed as a more flexible and efficient way to get products to market. The word ‘agile’ refers to the ability to move quickly and easily. Therefore, an Agile approach enables project teams to adapt faster and easier compared to other project methodologies.

Many of today's projects have more unknowns than a traditional project management methodology can adequately handle. This uncertainty makes it challenging to document requirements and adapt to changes successfully.

This guide will explain the Agile methodology, which projects will benefit from an Agile approach, and how to implement one effectively.

Agile methodology is an approach to project management that uses four values and 12 principles to organize projects.

The four values of the Agile Manifesto are:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The Agile method works in ongoing sprints of project planning and execution, enabling you to continuously adapt and mature your plan, scope, and design throughout the project.

Agile projects require an iterative approach, which supports incremental, frequent, and consistent delivery of workable products to your customer or client. This innovative approach ensures your project team can consistently deliver concrete products without being delayed by changes and evolving requirements.

Agile has a high level of customer involvement and includes frequent reviews of progress with both the project team and the customer.

You can run an Agile project using several different frameworks. Some of the more popular ones include:

- Scrum
- Kanban
- Extreme Programming
- DSDM

This video on the Agile methodology further explains the approach, breaking down everything from the core principles to the Agile frameworks.

Agile methodologies typically involve the use of sprints or iterations, which are short periods of time during which the team focuses on delivering a specific set of features or functionality. Sprints typically last between one and four weeks, and involve a cycle of planning, development, testing, and review.

Agile methodologies also emphasize the importance of regular communication and collaboration between team members and with customers. This often involves daily stand-up meetings, where team members provide updates on their progress and identify any obstacles they are facing.

Values of Agile,

The Agile Manifesto outlines four core values that underpin Agile methodologies:

- Individuals and interactions over processes and tools: Agile methodologies prioritize the people involved in the project and the interactions between them over strict adherence to processes and tools.

- Working software over comprehensive documentation: Agile methodologies prioritize delivering working software that meets the needs of customers over comprehensive documentation.

- Customer collaboration over contract negotiation: Agile methodologies prioritize working closely with customers to understand their needs and collaborate with them throughout the development process, rather than relying on strict contracts.

- Responding to change over following a plan: Agile methodologies prioritize the ability to respond to changing requirements and circumstances over sticking rigidly to a pre-defined plan.

These values reflect an emphasis on collaboration, flexibility, and responsiveness to change. They prioritize the delivery of value to customers over strict adherence to plans and processes, and recognize the importance of individuals and their interactions in the success of a project.

In addition to these core values, Agile methodologies also emphasize the importance of continuous improvement, with teams regularly reflecting on their processes and making adjustments based on customer feedback and changing circumstances. This focus on continuous improvement helps Agile teams to deliver high-quality products and services that meet the evolving needs of their customers.

Principles of Agile,

The Manifesto for Agile Software Development outlines 12 Agile principles that all projects should follow. These are:

1. **Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.**The first principle of Agile methodology states that customers should receive project deliverables or iterations across regular intervals throughout the project's life cycle, rather than just one product delivery at the end.

2. **Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage**. The Manifesto's authors found that, with traditional project management, it was difficult to accommodate last-minute change requests. This principle ensures that Agile projects can adapt to any changes, no matter how late in the game, with minimal delay.

3. **Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for shorter timescales.**Agile projects plan for frequent, short project timelines that allow for a fast turnaround of workable products. Teams will often break Agile projects into one to four week-long sprints or project intervals, each one ending in the delivery of a product.

4. **Business people and developers must work together daily throughout the project.** This Agile principle states that regular communication with all stakeholders is critical to the project's success. Commonly, this involves a short daily meeting with both the project team and any other key stakeholders.
5. **Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.**A central concept of the Agile project management methodology is that the right people need to be placed in the right positions and given the autonomy required to do their jobs well. It's essential to design a project team based on capabilities rather than job positions or titles. The project manager's focus should be on motivating the project team and supporting them, rather than micromanaging them.

6. **The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.** The Agile Manifesto emphasizes the importance of co-locating teams and stakeholders whenever possible, as face-to-face communication is more effective than email or phone. If your team cannot be co-located, video conferencing is an option that can still capture the value of non-verbal cues.

7. **Working software is the primary measure of progress.**The Agile methodology aims to provide complete, working deliverables. This goal should always take priority over any additional requirements, such as project documentation. Other metrics, such as hours spent or time elapsed, are not considered as important as delivering working products.

8. **Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.** According to this principle, Agile projects should have a consistent pace for each iterative cycle or sprint within the project. This breakdown should eliminate the need for overtime or crashing schedules while promoting frequent output of workable products. It should also create a repeatable cycle that the team can continuously follow for as long as necessary.

9. **Continuous attention to technical excellence and good design enhances agility.**An Agile project's primary focus should be on improving the end product and achieving advancements consistently over time. Each iteration should always improve on the previous one, and the team should always be looking to innovate.

10. **Simplicity – the art of maximizing the amount of work not done – is essential**.An Agile project aims to get just enough done to complete the project and meet the requested specifications. Any additional documentation, steps, processes, or work that does not add value to the customer or enhance the project outputs should be avoided or eliminated.

11. **The best architectures, requirements, and designs emerge from self-organizing teams.**Agile is based on the belief that you need motivated, autonomous, and skilled teams to deliver the best results and products. Teams should be empowered to organize and structure themselves as required. They should have the freedom to collaborate and innovate as they see fit, without being hampered by too much oversight.

12. **The team discusses how to become more effective at regular intervals, then tunes and adjusts its behavior accordingly.** A successful, self-motivated team requires a strong focus on advancing their skills and processes to grow and improve. The team should have regular reviews on their performance and outcomes, including discussions on improving as they move forward.

Stakeholders

Stakeholders are individuals or groups who have an interest or stake in the success of a project or organization. In the context of Agile software development, stakeholders can include a wide range of individuals and groups, including:

1) Customers: The individuals or organizations who will use or benefit from the software being developed.

2) Product Owners: The individuals responsible for defining and prioritizing the features and functionality of the software being developed.

3) Development team members: The individuals responsible for building and testing the software.

4) Project managers: The individuals responsible for coordinating the development team and ensuring that the project is completed on time and within budget.

5) Business analysts: The individuals responsible for gathering and analyzing requirements and ensuring that the software meets the needs of stakeholders.

6) Quality assurance team members: The individuals responsible for testing and ensuring the quality of the software being developed.

7) Stakeholder representatives: Individuals or groups who represent the interests of specific stakeholders, such as regulatory bodies or user groups.

Effective stakeholder management is critical to the success of an Agile project. By engaging with stakeholders early and often, Agile teams can ensure that they are building software that meets the needs of all stakeholders and delivers value quickly and efficiently. Regular communication, collaboration, and feedback are key to effective stakeholder management in Agile software development.

Challenges

The continuous nature of agile development processes raises a few serious testing challenges:

1. Changing Requirements

It can sometimes happen that management changes requirements or drops stories during a sprint, even though this is not encouraged in an agile/Scrum framework. This means that work already half-done needs to be discarded or modified, which changes the scope of testing unexpectedly.

How to master:

Testers should be able to react and modify their processes according to changing conditions, because in agile projects, change is common. When requirements change, testers should share as much information as they can about what tests they have conducted, and which areas of the application have not been tested yet. This can help the team understand how to make the required changes to the sprint without hurting the quality of the release.

2. Not Enough Information

Product owners, who are responsible for developing user stories, may have an idea about a new feature but may not be aware of the specifics. This means they can't write a good set of acceptance criteria. If there is missing information about requirements, testers can't build comprehensive test cases.

How to master:

Testers do not need in-depth requirements to begin testing, they can begin by coming up with high level scenarios that test the idea of the story, confirming them with the product owner. Testing can be done without the complete details about a feature. You can create high level test scenarios, even when particulars change.

3. Continuous Testing

Testing is not restricted to one part of the development process, rather it's an ongoing activity that starts before the development phase. This creates a major challenge because testers are expected to start building tests for features before coding has even started, or while coding is taking place.

How to master:

To make life easier for testers, user stories in the backlog should be expanded during sprint planning. Testers, developers and product owners should jointly define the details of each story and then write effective acceptance criteria.

The team should ensure that each story has sufficient acceptance criteria and that the context of the story is universally understood before work on development begins. This makes it possible to start creating tests early on, which can be implemented when the code for the feature is complete.

4. Technical Skills
Testers who work in an agile environment need to be technically savvy, helping developers with API testing, integration testing, and scripting UI automation checks with Selenium or similar frameworks. Testers with an exploratory or manual testing background entering the world of agile will encounter a steep learning curve.

How to master:

Testers can and should learn programming or scripting languages such as Javascript and Ruby. Testers who are familiar with programming but lack practical experience can ask for help from developers. Testers can also learn automated testing tools like Selenium tool and JMeter.

For specialized testing areas, such as performance, security, or compliance testing, teams should have dedicated testers with the relevant professional background, or leverage consultants with deep experience in these areas.

5. Frequent Regression Cycles
Developers frequently and continuously add features to the product. This can cause regressions in previous features. Testers use regression tests to identify this problem and overcome it, but manual regression testing is impractical in a fast-paced agile environment.

Another challenge is that modern web applications behave differently when viewed on different devices or browsers. This creates a complex matrix of compatibility testing scenarios, which need to be tested to ensure that the application functions correctly for all users.

How to master:

Agile testers rely on automation. They use unit testing to ensure recent changes have not broken the code, and tools like Selenium and JMeter to verify there is no regression in basic functionality. Testers can use Docker or Selenium Grid to manage and run their automated test, in parallel on various browsers and machines.

6. Lack of Communication
If communication between developers, testers and the product owners is lacking, agile testing will simply not work.

How to master:

Direct communication within the team should be strongly encouraged. Developers, testers and product owners should talk face-to-face on a regular basis to ensure everyone is on the same page. Scrum ceremonies such as stand-up meetings, sprint planning, and retrospectives are instrumental in creating a common understanding of the sprint scope and goals.

7. No Quality Measurement
Agile teams today have no single measurement of quality, which they can use to optimize and plan testing efforts.

There are numerous metrics like unit test code coverage, lines of code (LOC) and code complexity, but none of them provides a clear picture as to "where we stand" with quality at each point in a sprint or release – which areas of the product are working well, which are less stable and at higher risk of quality issues.

Most agile teams are flying blind, responding to production failures or bugs, but unable to proactively focus on product areas which have the biggest quality issues.

MODULE 2:

Agile and Scrum Principles:

Agile is an iterative and collaborative approach to software development that emphasizes flexibility, adaptability, and customer satisfaction. Scrum is a framework that follows agile principles and is used for managing and completing complex projects.

Some key principles of Agile include:

- Individuals and interactions over processes and tools: Agile prioritizes people and communication over rigid procedures and technical tools.
- Working software over comprehensive documentation: Agile values delivering functioning software over exhaustive documentation.
- Customer collaboration over contract negotiation: Agile emphasizes working closely with customers to understand their needs and expectations, rather than relying on a fixed contract.
- Responding to change over following a plan: Agile prioritizes being able to adapt to changing circumstances and requirements over sticking to a rigid plan.

Some key principles of Scrum include:

- Empirical process control: Scrum is based on transparency, inspection, and adaptation. Teams use feedback loops to continually improve their work processes.

- Self-organizing teams: Scrum teams are self-organizing, cross-functional, and accountable for their work.

- Iterative development: Scrum divides work into sprints, which are short, time-boxed periods during which the team completes a set of tasks.

- Prioritized product backlog: Scrum teams maintain a prioritized list of work items, known as the product backlog, which helps them focus on delivering the most valuable features first.

Agile Manifesto,

The Agile Manifesto is a set of guiding values and principles for Agile software development that emphasizes flexibility, collaboration, and continuous improvement. The manifesto was created in 2001 by a group of software development experts who were frustrated with the traditional, rigid approach to software development. The four values of the Agile Manifesto are:

- Individuals and interactions over processes and tools: Agile emphasizes the importance of people and communication in software development, rather than relying solely on processes and tools.

- Working software over comprehensive documentation: Agile values delivering working software that meets the customer's needs over exhaustive documentation.

- Customer collaboration over contract negotiation: Agile emphasizes the importance of working closely with the customer to understand their needs and expectations, rather than relying on a fixed contract.

- Responding to change over following a plan: Agile values the ability to adapt to changing circumstances and requirements, rather than sticking to a rigid plan.

In addition to these four values, the Agile Manifesto also includes 12 principles that further define and elaborate on the values. These principles include things like welcoming changes in requirements, delivering working software frequently, and promoting sustainable development practices. The Agile Manifesto has been widely adopted in software development and has influenced many other industries and fields.

Twelve Practices of Extreme Programming (XP)

Extreme Programming (XP) is an Agile software development methodology that emphasizes collaboration, feedback, and continuous improvement. XP consists of a set of 12 practices, which are:

1. Planning Game: Collaborative planning between customers and developers to prioritize features and set goals for each iteration.

2. Small Releases: Frequent delivery of working software in small, manageable increments.

3. Metaphor: A shared understanding of the system and its behavior through the use of an agreed-upon metaphor.

4. Simple Design: Keeping the design of the system as simple as possible to reduce complexity and allow for flexibility.

5. Testing: Automated testing at all levels (unit, integration, and acceptance) to ensure that the software is working correctly.

6. Refactoring: Continuous improvement of the code through refactoring, which involves restructuring the code without changing its behavior.

7. Pair Programming: Two programmers work together on a single task, with one typing and the other reviewing and offering feedback.

8. Collective Ownership: The entire team is responsible for the quality of the code, and anyone can modify or improve any part of it.

9. Continuous Integration: Frequent integration of code changes into a shared repository, with automated builds and tests to detect problems early.

10. 40-Hour Week: Keeping the workweek to a reasonable limit to prevent burnout and ensure sustainable development.

11. On-Site Customer: A customer representative is available to the team on a regular basis to provide feedback and answer questions.

12. Coding Standards: Standardizing the coding practices and conventions used by the team to ensure consistency and maintainability.

These practices are designed to promote collaboration, communication, and quality, and they help teams to deliver high-quality software that meets the customer's needs in a timely and efficient manner.

Scrum Practices

Scrum is an Agile framework for software development that emphasizes collaboration, flexibility, and continuous improvement. Scrum consists of a set of practices that help teams to organize their work and manage complex projects effectively. Some of the key practices of Scrum include:

1) Sprint: Scrum projects are divided into fixed-length iterations called Sprints, usually 1-4 weeks long.

2) Product Backlog: A prioritized list of features and requirements that need to be developed to achieve the project's goals.

3) Sprint Planning: At the beginning of each Sprint, the team meets to plan the work for the upcoming Sprint.

4) Daily Scrum: A brief daily meeting, usually 15 minutes, where team members discuss progress and any issues or obstacles they're facing.

5) Sprint Review: A meeting at the end of each Sprint where the team presents the completed work to stakeholders and receives feedback.

6) Sprint Retrospective: A meeting after each Sprint to reflect on the team's performance and identify areas for improvement.

7) Scrum Master: A facilitator who ensures that the team is following the Scrum framework and helps to resolve any issues or obstacles.

8) Product Owner: The person responsible for defining and prioritizing the requirements and features of the product being developed.

9) Self-organizing Teams: The team is responsible for planning and executing the work, and for making decisions about how to do it.

10) Burn-down Chart: A visual representation of the team's progress, showing how much work is remaining and how much time is left in the Sprint.

These practices help teams to work collaboratively, stay focused on their goals, and continuously improve their work processes. By using Scrum, teams can more effectively manage complex projects and deliver high-quality software that meets the needs of the customer.

Applying Scrum

To apply Scrum to a software development project, a team would typically follow these steps:

- Identify the Product Owner: The Product Owner is the person who is responsible for defining and prioritizing the requirements and features of the product. The Product Owner works closely with the team to ensure that the product is being developed to meet the customer's needs.

- Form the Scrum Team: The Scrum Team is a cross-functional team of developers, testers, and other stakeholders who are responsible for delivering the product.

- Create the Product Backlog: The Product Backlog is a prioritized list of features and requirements that need to be developed to achieve the project's goals. The Product Owner is responsible for maintaining the Product Backlog and ensuring that it is up-to-date.

- Plan the Sprint: At the beginning of each Sprint, the team meets to plan the work for the upcoming Sprint. The team decides how much work they can commit to completing during the Sprint and creates a Sprint Backlog, which is a list of the specific tasks that need to be completed to deliver the Sprint Goal.

- Daily Scrum: During the Sprint, the team meets daily for a brief Daily Scrum meeting, where each team member reports on their progress and any issues or obstacles they're facing.

- Sprint Review: At the end of each Sprint, the team presents the completed work to stakeholders and receives feedback. The team may demonstrate the working software and review the Product Backlog to determine what items should be prioritized for the next Sprint.

- Sprint Retrospective: After the Sprint Review, the team meets for a Sprint Retrospective meeting to reflect on the team's performance and identify areas for improvement. The team discusses what went well, what didn't go well, and what changes they can make to improve their performance in the next Sprint.

- Repeat: The team then repeats the process by selecting new items from the Product Backlog for the next Sprint and starting the planning process again.

By following these steps, the team can work collaboratively, stay focused on their goals, and continuously improve their work processes. This can lead to the delivery of high-quality software that meets the needs of the customer in a timely and efficient manner.

Need of scrum

Scrum is a popular Agile methodology used in software development projects. It is designed to help teams work more effectively and efficiently, with a focus on delivering high-quality software that meets the needs of the customer. Here are some reasons why Scrum is important:

- Increased collaboration: Scrum emphasizes collaboration among team members, with a focus on communication and transparency. By working together closely, team members can share knowledge and ideas more effectively, which can lead to better solutions and higher-quality work.

- Flexibility: Scrum is designed to be flexible, with a focus on adapting to changing requirements and priorities. This can help teams to respond quickly to changing business needs and customer requirements.

- Continuous improvement: Scrum emphasizes continuous improvement, with a focus on reflecting on the team's performance and identifying areas for improvement. By continuously improving their processes, teams can become more efficient and effective over time.

- Customer focus: Scrum is designed to be customer-focused, with a focus on delivering software that meets the needs of the customer. By involving the customer in the development process, teams can ensure that the software they deliver meets the customer's needs and requirements.

- Predictability: Scrum provides a predictable framework for software development, with fixed-length iterations (Sprints) and regular planning, review, and

retrospective meetings. This can help teams to better plan and manage their work, and to deliver software on time and within budget.

Overall, Scrum provides a structured and effective approach to software development that can help teams to work more collaboratively, efficiently, and effectively, resulting in higher-quality software that meets the needs of the customer.

Working of scrum

Scrum is an Agile methodology for software development that is based on a framework of iterative and incremental development. Scrum involves a set of practices, roles, and events that work together to support the development of high-quality software in a collaborative and transparent manner. Here is an overview of how Scrum works:

- Product Owner: The Product Owner is responsible for defining the product vision and prioritizing the product backlog. They work closely with the development team to ensure that the product is being developed to meet the customer's needs.

- Development Team: The Development Team is responsible for delivering the product increment at the end of each Sprint. The team is cross-functional, meaning that it includes developers, testers, and other stakeholders who work together to deliver the product.

- Scrum Master: The Scrum Master is responsible for ensuring that the Scrum framework is being followed and that the team is working effectively. They help to remove any obstacles or impediments that may be hindering the team's progress.

- Product Backlog: The Product Backlog is a prioritized list of features and requirements that need to be developed to achieve the project's goals. The Product Owner is responsible for maintaining the Product Backlog and ensuring that it is up-to-date.

- Sprint: A Sprint is a fixed-length iteration of 1-4 weeks, during which the team works to deliver a product increment. The team selects items from the Product Backlog and commits to delivering them by the end of the Sprint.

- Sprint Planning: At the beginning of each Sprint, the team meets to plan the work for the upcoming Sprint. The team decides how much work they can commit to completing during the Sprint and creates a Sprint Backlog, which is a list of the specific tasks that need to be completed to deliver the Sprint Goal.

- Daily Scrum: During the Sprint, the team meets daily for a brief Daily Scrum meeting, where each team member reports on their progress and any issues or obstacles they're facing.

- Sprint Review: At the end of each Sprint, the team presents the completed work to stakeholders and receives feedback. The team may demonstrate the working software and review the Product Backlog to determine what items should be prioritized for the next Sprint.

- Sprint Retrospective: After the Sprint Review, the team meets for a Sprint Retrospective meeting to reflect on the team's performance and identify areas for improvement. The team discusses what went well, what didn't go well, and what changes they can make to improve their performance in the next Sprint.

- By following these practices, roles, and events, Scrum provides a structured and effective approach to software development that helps teams to work more collaboratively, efficiently, and effectively, resulting in higher-quality software that meets the needs of the customer.

Advanced Scrum Applications

Advanced Scrum Applications are variations or adaptations of the Scrum framework that are tailored to specific needs or situations. These applications build on the core Scrum framework and introduce additional practices or techniques to address specific challenges or requirements. Here are some examples of advanced Scrum applications:

1. Large-Scale Scrum (LeSS): LeSS is a framework for applying Scrum to large-scale projects with multiple teams. It introduces additional practices, such as Area Product Owners, Overall Retrospectives, and Integration Sprints, to help teams collaborate and coordinate more effectively.

2. Scrum of Scrums (SoS): SoS is a technique for coordinating multiple Scrum teams working on the same project. It involves regular meetings between the Scrum Masters of each team to share information, identify dependencies, and coordinate their work.

3. Distributed Scrum: Distributed Scrum is an adaptation of Scrum for teams that are geographically dispersed. It involves the use of technology, such as video conferencing and collaboration tools, to facilitate communication and collaboration between team members.

4. Kanban-Scrumban: Kanban-Scrumban is a hybrid approach that combines elements of Scrum and Kanban. It introduces Kanban practices, such as visualizing work and limiting work in progress, to help teams manage their work more effectively.

5. Agile Release Trains (ARTs): ARTs are a technique for coordinating multiple Scrum teams working on a large-scale project. It involves the use of a shared backlog, regular synchronization meetings, and a common release schedule to ensure that all teams are working towards the same goals.

These advanced Scrum applications are designed to help teams address specific challenges or requirements that may not be fully addressed by the core Scrum framework. By adapting Scrum to their specific needs, teams can achieve better results and deliver higher-quality software more effectively.

Scrum and the Organization

Scrum is not just a development methodology but also a framework for organizing and managing work. As such, it has a significant impact on the organization as a whole. Here are some ways in which Scrum affects the organization:

1) Increased Transparency: Scrum promotes transparency by making all aspects of the work visible to the team and stakeholders. This increased transparency helps to build trust and improves communication, which can lead to better decision making.

2) Agile Mindset: Scrum promotes an Agile mindset within the organization, which is focused on continuous improvement, collaboration, and delivering value to the customer. This mindset can help to drive innovation and enable the organization to respond more quickly to changing market conditions.

3) Empowered Teams: Scrum emphasizes the importance of self-organizing and cross-functional teams. By empowering teams to make decisions and take ownership of their work, Scrum can help to create a more engaged and motivated workforce.

4) Business Agility: Scrum can help organizations to become more agile by enabling them to respond more quickly to changing customer needs and market conditions. This can give organizations a competitive advantage and help them to stay ahead of the curve.

5) Continuous Improvement: Scrum promotes a culture of continuous improvement by emphasizing regular reflection and adaptation. This can help organizations to identify areas for improvement and make changes to improve their processes and outcomes.

6) Stakeholder Engagement: Scrum involves regular engagement with stakeholders, including customers, users, and sponsors. This engagement helps to ensure that the product is meeting the needs of the stakeholders and can lead to increased customer satisfaction and loyalty.

Overall, Scrum has a significant impact on the organization, promoting transparency, agility, and collaboration, and enabling teams to deliver higher-quality products more effectively. By embracing Scrum and its principles, organizations can create a culture of continuous improvement and innovation that can help them to stay ahead in an increasingly competitive market.

Scrum values

Scrum is based on a set of values that guide the behavior and interactions of the Scrum team. These values are essential for creating a culture of collaboration, openness, and continuous improvement. The five Scrum values are:

- Focus: The Scrum team focuses on delivering value by prioritizing the work that is most important to the customer. The team stays focused on the goal and works together to achieve it.

- Courage: The Scrum team has the courage to take on difficult challenges and to do what is right, even if it is not easy. Team members are encouraged to speak up and share their ideas and concerns.

- Openness: The Scrum team is open and transparent in their communication and work. They share their progress and obstacles with each other and with stakeholders, and they are willing to listen to feedback and adapt.

- Commitment: The Scrum team is committed to delivering high-quality products and to continuously improving their processes. They take ownership of their work and hold themselves accountable for their results.

- Respect: The Scrum team respects each other, their roles, and their differences. They work collaboratively and support each other to achieve their goals.

These values are not just words on a page, but are actively practiced by the Scrum team in their daily work. By embracing these values, the Scrum team can create a culture of trust, respect, and continuous improvement, which can lead to better outcomes and higher quality products.

MODULE 3:

Agile Requirements:

User Stories

User Stories are a technique used in Agile software development to capture and express requirements from the user or customer perspective. They are short, simple, and easy-to-understand statements that describe a particular feature or functionality that the user wants in the product.

The typical format for a user story is:

"As a [user persona], I want [feature or functionality], so that [benefit or value]."

For example:

"As a customer, I want to be able to easily search for products on the website, so that I can quickly find what I'm looking for."

User stories are often written on index cards or sticky notes and organized on a board, such as a Kanban board or a Scrum board, to track progress and prioritize work. They are designed to be small, independent, and deliverable within a single iteration or sprint.

The purpose of user stories is to focus on the user's needs and the value that the feature or functionality will provide to them, rather than getting bogged down in technical details or implementation specifics. By focusing on the user and their needs, user stories help to ensure that the development team is delivering features that are relevant and valuable to the user.

User stories are often complemented by acceptance criteria, which are a set of conditions that must be met for the user story to be considered complete. These acceptance criteria help to define the boundaries and expectations of the user story and ensure that it meets the user's needs.

Overall, user stories are a powerful tool for Agile requirements gathering and help to ensure that the development team is delivering value to the user in a timely and efficient manner.

Backlog Management.

Backlog management is a critical aspect of Agile development and is essential for ensuring that the team is working on the most important items at any given time. The product backlog is a prioritized list of user stories, features, and enhancements that the team plans to work on in the future. The backlog is owned by the Product Owner, who is responsible for maintaining it and ensuring that it reflects the priorities of the customer or stakeholders.

Here are some key aspects of backlog management:

1. Prioritization: The Product Owner is responsible for prioritizing the backlog items based on the needs of the customer or stakeholders. The most important items are placed at the top of the list, and the least important items are placed at the bottom.

2. Grooming: The Product Owner regularly reviews and refines the backlog to ensure that it is up to date and reflects the current priorities of the customer or stakeholders. Backlog grooming involves adding new items, removing obsolete items, and clarifying existing items.

3. Estimation: The team estimates the effort required to complete each backlog item using techniques such as planning poker or relative sizing. This estimation helps to ensure that the team can realistically plan and commit to the work for each sprint.

4. Sprint Planning: Before the start of each sprint, the team reviews the backlog items and selects the ones that they will work on during the sprint. The team also defines the sprint goal and creates a plan for how they will deliver the selected items.

5. Refinement: Backlog refinement is an ongoing process that involves the entire team. The team works with the Product Owner to clarify and refine the backlog items, ensuring that they are well understood and can be completed within a single sprint.

Effective backlog management is critical for ensuring that the team is working on the most important items and that they can deliver value to the customer or stakeholders in a timely manner. By regularly reviewing, refining, and prioritizing the backlog, the team can ensure that they are focused on delivering the highest value items first.

Agile Architecture:

Agile architecture is an approach to software architecture that emphasizes the importance of flexibility, adaptability, and collaboration in software design. It is a way of designing software that is aligned with Agile principles and values, such as responding to change, delivering working software frequently, and working closely with customers and stakeholders.

Here are some key aspects of Agile architecture:

- Emergent design: Agile architecture embraces the idea that design should emerge over time rather than being fully specified upfront. The team should be able to respond to changing requirements and customer feedback by adapting the architecture as needed.

- Collaborative design: Agile architecture is a collaborative effort that involves the entire team, including developers, architects, testers, and product owners. Everyone works together to create a design that meets the needs of the customer and can be delivered in an Agile way.

- Continuous delivery: Agile architecture is focused on delivering working software frequently and continuously. This requires a design that is flexible and adaptable, and can be delivered in small, incremental releases.

- Minimal viable architecture: Agile architecture emphasizes the importance of delivering a minimal viable architecture (MVA) that is just sufficient to meet the needs of the customer. The MVA is designed to be flexible and adaptable, allowing the team to respond to changing requirements and feedback.

- Technical excellence: Agile architecture requires a focus on technical excellence, including practices such as continuous integration, automated testing, and refactoring. The team must be able to deliver high-quality software that is reliable, maintainable, and scalable.

Overall, Agile architecture is a way of designing software that is aligned with the principles and values of Agile development. By embracing flexibility, collaboration, and continuous delivery, Agile architecture allows the team to deliver working software that meets the needs of the customer in a timely and efficient manner.

Feature Driven Development

Feature Driven Development (FDD) is an Agile software development methodology that focuses on delivering small, working features in a timely and efficient manner. It was developed by Jeff De Luca in the mid-1990s and is a lightweight, iterative approach to software development.

Here are some key aspects of FDD:

- Domain-driven design: FDD emphasizes the importance of understanding the domain of the software being developed. The team works closely with the domain

experts to develop a shared understanding of the domain and to identify the key features that need to be developed.

- Feature list: The team maintains a prioritized feature list, which is a comprehensive list of all the features that need to be developed. Each feature is broken down into smaller, more manageable tasks.

- Short iterations: FDD emphasizes the importance of short iterations, typically two weeks in length. Each iteration focuses on developing a small set of related features.

- Process milestones: FDD includes five process milestones, which are used to manage the development process. These milestones include developing an overall model, developing a feature list, planning by feature, designing by feature, and building by feature.

- Code ownership: FDD emphasizes the importance of code ownership, with each developer being responsible for a specific set of features. This allows the team to work in parallel, with each developer focusing on their own area of responsibility.

Overall, FDD is a lightweight, Agile methodology that focuses on delivering small, working features in a timely and efficient manner. By emphasizing domain-driven design, short iterations, and code ownership, FDD allows the team to develop software that meets the needs of the customer while minimizing risk and maximizing productivity.

Agile Risk Management:

Agile risk management is the process of identifying, assessing, and responding to risks in an Agile software development project. It is an iterative process that is integrated into the overall Agile development process, and it emphasizes the importance of continuous risk assessment and response.

Here are some key aspects of Agile risk management:

1. Risk identification: The first step in Agile risk management is to identify potential risks. This is typically done in collaboration with the team and stakeholders, and it involves looking at the project from multiple perspectives to identify any areas of potential risk.

2. Risk assessment: Once potential risks have been identified, they need to be assessed in terms of their likelihood and potential impact. This helps the team to prioritize risks and focus on those that are most likely to have a significant impact on the project.

3. Risk response: Based on the assessment of risks, the team can develop a risk response plan that outlines how each risk will be addressed. This may involve taking proactive measures to mitigate or avoid the risk, or it may involve developing a contingency plan to respond if the risk occurs.

4. Continuous risk management: Agile risk management is an iterative process that is integrated into the overall Agile development process. As the project progresses, the team continues to identify, assess, and respond to risks, making adjustments as needed to ensure that the project stays on track.

5. Collaborative approach: Agile risk management emphasizes collaboration between the team and stakeholders, as well as between team members. Everyone is encouraged to share their perspectives and insights to help identify potential risks and develop effective risk response plans.

Overall, Agile risk management is a collaborative, iterative process that is integrated into the overall Agile development process. By identifying, assessing, and responding to risks in a timely and effective manner, the team can minimize the impact of risks and ensure that the project stays on track to meet its goals.

Risk and Quality Assurance,

Risk and Quality Assurance are two important concepts in software development that work together to ensure that software products are developed to meet the needs of the customer and are of high quality.

Risk management focuses on identifying and mitigating risks throughout the software development lifecycle. Risks can include anything that may impact the success of the project, such as technical issues, resource constraints, or changes in requirements. Risk management involves identifying potential risks, assessing their likelihood and potential impact, and developing strategies to mitigate or avoid those risks.

Quality assurance, on the other hand, focuses on ensuring that the software being developed meets the needs of the customer and is of high quality. Quality assurance involves defining standards and processes for software development, testing the software to ensure it meets those standards, and making sure that defects are identified and corrected in a timely manner.

Both risk management and quality assurance are important for ensuring the success of a software development project. Effective risk management can help to mitigate potential issues that could impact the quality of the software being developed, while quality assurance can help to identify and correct defects before they become major issues.

Ultimately, both risk management and quality assurance are focused on delivering software products that meet the needs of the customer and are of high quality. By working together, these two concepts can help to ensure that software development projects are completed successfully, on time, and within budget.

Agile Tools.

In agile development, leading as project management is not the easiest job. Jumping between your daily scrums to your next sprint, it causes hard to focus on the work. The agile development tools fulfill your needs, and does it for you.

There are several agile tools available in the market. Some of them are listed below:

JIRA Agile

Jira is a tool developed by Australian Company **Atlassian**. It is used for **issue tracking, bug tracking, and project management**. The bugs and issues are related to your software and Mobile apps. The Jira dashboard consists of many useful functions and features. This function and features make secure handling of issues.

Agile Software Features:

- Issue tracking
- Bug tracking
- Boards
- Epics
- Custom fields

ClickUp

ClickUp is one of the ultimate agile management tools. It is used for anyone who uses agile methodology. It is the only project management tool whose goal is "to move quickly and easily". **CuickUp** is in the hand of some of the most famous agile team, including **Google** and **Apple!** It is a free forever plan, so, the team can get their hand of ClickUp.

Agile Software Features:

- Create epics
- Use story points
- Analyze sprint performance
- Time estimates
- Start and due dates
- Time tracking

Github

Github is one of the largest hosted Git serve where the developers can store all of their codes for a vast number of projects there. The Github provides such a facility of record edits across an entire team in **real time**. Github is also integrated with many other tools so, many people such as developer and product owner can work on the same code at the same time.

The project manager can make the Github work for their team. It includes lots of project management tools which help him to inspect what the development team is working on.

Agile Software Features:

- Issue tracking
- Mentions
- Labels
- Link issues and pull requests

LeanKit

LeanKit is the ultimate management tool for a **Kanban** board on the agile progress for your sprints. It uses cards to represent the work items and live statuses of team member. It work perfect for the remote employees to ensure everyone can see the Kanban board in real time. It prevents the same task to complete twice and make sure the whole team remains on the same page.

LeanKit work well for cross-functional team which is benefit for Scrum or Kanban boards.

Agile Software Features:

- Board view templates
- Track issues and bugs
- Manage project portfolios
- Lean metrics and reporting

Planbox

Planbox is a tool that tracks the process of burndown charts. Using this everyone knows how far you are from the Sprint?s completion/goal. Burndown charts are most important part of the agile cycle. Planbox also integrates the customer bug reports, and fixes, making it useful for a wide range of users.

It has an advance reporting features which make it easy to review the status and areas where improvement is needed at Daily Scrum.

MODULE 4:

Agile Testing: Agile Testing Techniques

Agile software development emphasizes on the importance of testing throughout the entire software development lifecycle. Here are some common Agile testing techniques:

- Test-Driven Development (TDD): TDD is a technique that involves writing tests before writing the code. The idea is to create small tests that focus on specific functionality, and then write the code to pass those tests.
- Behavior-Driven Development (BDD): BDD is a technique that involves defining user requirements in the form of scenarios or user stories, and then writing tests based on those scenarios. BDD tests are typically written in a natural language format that is easy for non-technical stakeholders to understand.
- Acceptance Test-Driven Development (ATDD): ATDD is a technique that involves defining acceptance criteria for user stories, and then writing tests to validate that the software meets those criteria. ATDD tests are typically written collaboratively by developers, testers, and business stakeholders.
- Exploratory Testing: Exploratory testing is a technique that involves exploring the software to find defects and unexpected behavior. Unlike other testing techniques, exploratory testing is not scripted and is often performed informally by experienced testers.
- Continuous Integration (CI) Testing: CI testing is a technique that involves automatically building and testing the software every time code is committed to the repository. This ensures that defects are identified early in the development process and can be fixed before they become larger issues.

Agile testing techniques are designed to ensure that the software being developed meets the needs of the customer and is of high quality. By incorporating testing throughout the entire development process, Agile teams can identify and fix defects quickly, leading to faster delivery of high-quality software products.

Test-Driven Development

Test-Driven Development (TDD) is a software development technique that involves writing tests before writing the actual code. The idea is to create small tests that focus on specific functionality, and then write the code to pass those tests. The process is iterative, with the developer writing a test, running it to ensure that it fails, writing the code to make it pass, and then refactoring the code to ensure that it is of high quality.

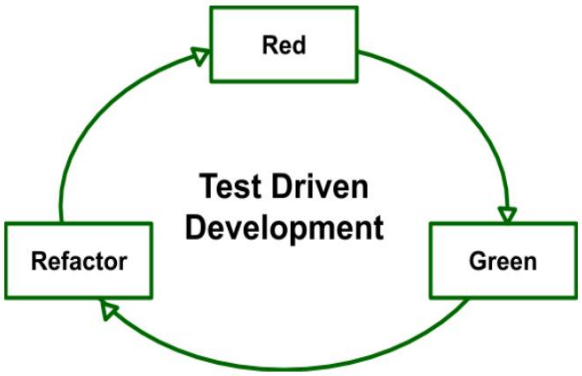
Test Driven Development is the process in which test cases are written before the code that validates those cases. It depends on repetition of a very short development cycle. Test driven Development is a technique in which automated Unit test are used to drive the design and free decoupling of dependencies.

The following sequence of steps is generally followed:

1. Add a test – Write a test case that describe the function completely. In order to make the test cases the developer must understand the features and requirements using user stories and use cases.
2. Run all the test cases and make sure that the new test case fails.
3. Write the code that passes the test case
4. Run the test cases

5. Refactor code – This is done to remove duplication of code.
6. Repeat the above mentioned steps again and again

Motto of TDD:



1. **Red –** Create a test case and make it fail
2. **Green –** Make the test case pass by any means.
3. **Refactor –** Change the code to remove duplicate/redundancy.

Benefits:

- Unit test provides constant feedback about the functions.
- Quality of design increases which further helps in proper maintenance.
- Test driven development act as a safety net against the bugs.
- TDD ensures that your application actually meets requirements defined for it.
- TDD have very short development lifecycle.

User Acceptance Test

User Acceptance Testing (UAT) is a type of testing performed on software to ensure that it meets the requirements of the end-users. UAT is typically performed by a group of end-users who test the software in a real-world environment to ensure that it meets their needs and expectations.

The goal of UAT is to ensure that the software meets the business requirements and user expectations before it is released to the production environment. UAT is usually the final step in the testing process and is performed after the software has gone through various testing stages, such as functional testing, integration testing, and system testing.

During UAT, end-users test the software in a real-world environment to ensure that it meets their needs and expectations. The testers perform tasks that represent real-life scenarios to ensure that the software is working as intended. Any issues or defects that are identified during UAT are reported back to the development team for correction.

UAT is an essential part of the software development process because it ensures that the software meets the needs of the end-users. By incorporating UAT into the development process, software development teams can ensure that they are delivering high-quality software that meets the business requirements and user expectations.