MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Autonomous Institution – UGC, Govt. of India

Department of COMPUTATIONAL INTELLIGENCE (AIDS)

B. TECH (R-20 Regulation)

(IV YEAR – I SEM)

2024-25

AGILE METHODOLOGIES

(R20A6607)

LECTURE NOTES

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12(B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE-Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad–500100, Telangana State, India

Department of COMPUTATIONAL INTELLIGENCE

(AIDS)

AGILE METHODOLOGIES

(R20A6607)

LECTURE NOTES

Department of Computational Intelligence

CSE (Artificial Intelligence and Machine Learning)

Vision

To be a premier centre for academic excellence and research through innovative interdisciplinary collaborations and making significant contributions to the community, organizations, and society as a whole.

Mission

- ❖ To impart cutting-edge Artificial Intelligence technology in accordance with industry norms.
- ❖ To instill in students a desire to conduct research in order to tackle challenging technical problems for industry.
- ❖ To develop effective graduates who are responsible for their professional growth, leadership qualities and are committed to lifelong learning.

QUALITY POLICY

- ❖ To provide sophisticated technical infrastructure and to inspire students to reach their full potential.
- ❖ To provide students with a solid academic and research environment for a comprehensive learning experience.
- ❖ To provide research development, consulting, testing, and customized training to satisfy specific industrial demands, thereby encouraging self-employment and entrepreneurship among students.

For more information: www.mrcet.ac.in

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

IV Year B.Tech. AI&DS- I Sem L/T/P/C 3/-/-/-3

(R20A6607) AGILE METHODOLOGIES
PROFESSIONAL ELECTIVE - VI

Course Objectives:

i. To introduce characteristics of an agile development process.
ii. To understand agile software development process models and plan driven process models.
iii. To understand software project characteristics that would be suitable for an agile process.
iv. To impart and Identify software project characteristics that would not be suitable for an agile process.
v. To implement a small scale software project using the Scrum process methodology

Unit 1:

History of Agile Methodologies, Agile and Lean Software Development: Basics and Fundamentals, Extreme Programming, Scrum, Agile and Scrum Principles Agile Manifesto, Twelve Practices of XP

Unit 2:

Agile Estimation &amp; Planning ,Agile Requirements,User Stories, Backlog Management, Agile Architecture

Unit 3:

Tracking Agile Projects, Lean Software Development, Agile Risk management,

Unit 4:

Agile Project Tools, Continuous Integration (CI)

Unit 5:

Agile Testing, Scaling Agile for Large Projects

Text Books:

1) Agile Development with Scrum, Ken Schwaber & Mike Beedle, Prentice Hall, 2001
2) Integrating Agile Development in the Real World, Peter Schuh, Charles River Media, 2005 (on Books 24x7)

References:

1) Agile Software Development – The Cooperative Game (2nd Edition), Alistair Cockburn, 2007
2) Succeeding With Agile, Software Development Using Scrum, Mike Cohn, Addison

Wesley, 2010 SDLC 3.0 Beyond a Tacit Understanding of Agile, Mark Kennaley, Fourth Medium Press, 2010 Course Outcomes:

Upon completion of the course, the student will be able to:

i. Define the common characteristics of an agile development process.

ii. List and contrast state of the practice agile methodologies.
iii. Contrast agile software development process models and plan driven process models.
iv. Identify software project characteristics that would be suitable for an agile process.
v. Identify software project characteristics that would not be suitable for an agile process.
vi. Plan and implement a small scale software project using the Scrum process methodology.

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

CSE (Artificial Intelligence and Machine Learning)

INDEX

History of Agile

- In 1957, people started figuring out new ways to build computer programs. They wanted to make the process better over time, so they came up with iterative and incremental methods.
- In the 1970s, people started using adaptive software development and evolutionary project management. This means they were adjusting and evolving how they built software.
- In the 1990s, there was a big change. Some people didn't like the strict and super-planned ways of doing things in software development. They called these old ways "waterfall." So, in response, lighter and more flexible methods showed up.

These included:

1. Rapid Application Development (RAD) in 1991.
2. Unified Process (UP), Dynamic Systems Development Method (DSDM) in 1994.
3. Scrum in 1995.
4. Crystal Clear and Extreme Programming (XP) in 1996.
5. Feature-Driven Development (FDD) in 1997.

Even though these came before the official "Agile Manifesto", we now call them agile software development methods.

- In 2005, Alistair Cockburn and Jim Highsmith added more ideas about managing projects, creating the PM Declaration of Interdependence.
- Then, in 2009, a group, including Robert C. Martin, added principles about software development. They called it the Software Craftsmanship Manifesto, focusing on being professional and skilled.
- In 2011, the Agile Alliance, a group of agile enthusiasts, made the Guide to Agile Practices (later called Agile Glossary). This was like a shared document where agile people from around the world put down their ideas, terms, and guidelines. It's a bit like a dictionary for how to do agile things.

What is LSD?

Lean Software Development (LSD) is an approach derived from lean manufacturing principles aimed at optimizing efficiency and minimizing waste in the software development process.

1. **Prevent Defects:** It integrates quality assurance throughout the development process to prevent defects.
2. **Eliminate Waste:** It focuses on activities that add value to the customer and eliminates those activities that do not add value.
3. **Fast Delivery:** Reduces cycle time to deliver software quickly and respond to feedback and changing requirements rapidly.
4. **Delay Decisions:** Delay decisions until they can be made based on facts.

History of LSD

Here is a brief timeline highlighting the key milestones in the history of LSD:

Timeline	Milestone	Description
1980s	Lean Manufacturing	Toyota Production Systems (TPS) developed by Taiichi Ohno and

Timeline	Milestone	Description
	Principles	Shigeo Shingo emphasizes on eliminating waste, improving quality, and continuous improvement. These were the foundation principles for lean thinking.
1990s	Emergence of Lean Thinking	The book “The Machine That Changed The World” by James P. Womack, Daniel T. Jones, and Daniel Roos describes the principles of lean manufacturing.
Early 2000s	Lean Principles in Software Development	The book “Lean Software Development: An Agile Toolkit” by Mary Poppendieck and Tom Poppendieck outlines the key principles of LSD.
2000s	Adoption and Spread	Lean principles started becoming popular in the software industry.
2010s	Integration with Agile	LSD becomes more integrated with Agile methodologies.
2020s	Continuous Evolution	Lean principles continue to evolve and influence modern software development practices.

Seven Principles of LSD

There are 7 established lean principles that come with a set of tactics, practices, and processes that build more efficient software products:

1. Eliminating the Waste

To identify and eliminate wastes e.g. unnecessary code, delay in processes, inefficient communication, issues with quality, data duplication, more tasks in the log than completed, etc. regular meetings are held by Project Managers. This allows team members to point out faults and suggest changes in the next turn.

2. Fast Delivery

Previously long-time planning used to be the key to success in business, but with time, it has been found that engineers spend too much time on building complex systems with unwanted features. So they came up with an MVP strategy which resulted in building products quickly that included a little functionality and launching the product to market and seeing the reaction. Such an approach allows them to enhance the product based on customer feedback.

3. Amplify Learning

Learning is improved through ample code reviewing and meetings that are cross-team applicable. It is also ensured that particular knowledge isn’t accumulated by one engineer who’s writing a particular piece of code so paired programming is used.

4. Builds Quality

LSD is all about preventing waste and keeping an eye on not sacrificing quality. Developers often apply test-driven programming to examine the code before it is written. Quality can also be gained by getting constant feedback from team members and project managers.

5. Respect Teamwork

LSD focuses on empowering team members, rather than controlling them. Setting up a collaborative atmosphere, keeping perfect balance when there are short deadlines and immense workload. This method becomes very important when new members join a well-established team.

6. Delay the Commitment

In traditional project management, it often happens when you make your application and it turns out to be completely unfit for the market. LSD method recognizes this threat and makes room for improvement by postponing irreversible decisions until all experiment is done. This methodology always constructs software as flexible, so new knowledge is available and engineers can make improvements.

7. Optimizing the Whole System

Lean's principle allows managers to break an issue into small constituent parts to optimize the team's workflow, create unity among members, and inspire a sense of shared responsibility which results in enhancing the team's performance.

LSD Process

Here is the overview of the lean software development process:

1. **Identify Value:** Understand the customer values and focus on delivering features that meet these needs.
2. **Map the Value Stream:** This involves mapping out the entire software development process to identify and eliminate wasteful activities that do not add value.
3. **Create Flow:** Ensure a smooth and continuous flow of work by minimizing delays and interruptions.
4. **Establish Pull:** Develop features based on customer demand rather than pushing features through the process.
5. **Seek Perfection:** Regularly review and refine the development process. Always encourage the team members to identify the areas of improvement and implement changes iteratively.
6. **Build Quality In:** Use practices such as test-driven development (TDD) and continuous integration to integrate quality assurance throughout the development process.
7. **Empower Teams:** Empower development teams by providing them with the necessary tools, resources, and autonomy to make decisions.

LSD vs Agile

Aspect	Lean Software Development (LSD)	Agile
Origin	It originated from lean manufacturing, especially the Toyota Production System.	It originated from the Agile Manifesto in 2001.
Focus	The focus is on waste elimination and value optimization.	The focus is on customer collaboration and iterative delivery.
Process and Practices	Kanban, Value Stream Mapping, Continuous Improvement (Kaizen).	Scrum, Extreme Programming (XP), Iterative Development.
Decision Making	Delays decisions until necessary and are based on facts.	Flexible and adaptive to changes as they arise.
Iteration and Feedback	It involves continuous improvement through regular feedback.	It involves frequent reassessment and adaptation in short cycles.
Customer Involvement	It involves understanding and delivering customer value continuously.	It involves continuous collaboration and feedback from customers.

Benefits of LSD

Here are some key benefits of LSD that help organizations to improve their software development processes and outcomes:

1. **Increased Efficiency:** LSD reduces delays and inefficiencies by identifying and eliminating non-value-adding activities.
2. **Higher Quality:** It integrates quality assurance throughout the development process, thus preventing defects and ensuring quality products.
3. **Faster Delivery:** Shorter development cycles allow for quicker release of features and updates, thus meeting customer demands more rapidly.
4. **Adaptability:** Delaying decisions until they are necessary and are based on facts, allowing teams to adapt to changes and new information.
5. **Enhanced Collaboration:** Engages customers throughout the development process, ensuring that their needs and feedback are continuously addressed.

Limitations of LSD

Here are some key limitations of LSD:

1. **Cultural Resistance:** Implementing LSD requires a significant cultural shift and if there is resistance to change from team members and management then it can hinder its adoption and effectiveness.
2. **Learning Curve:** There is a steep learning curve associated with understanding and applying lean principles and practices effectively.
3. **Requires Strong Leadership:** Successful implementation of LSD requires strong and committed leadership to guide the transition.
4. **Difficulty in Measuring Waste:** In LSD, determining waste is subjective and challenging. It requires a deep understanding of processes and value streams.
5. **Resource Intensive:** Implementing LSD requires an initial investment in training, tools, and process redesign. This can be significant.

The Agile Manifesto consists of four core values and twelve principles that provide a framework for teams to deliver high-quality software in a more adaptive and responsive manner

Four Core Values

Individuals and interactions over processes and tools

Agile emphasizes the importance of people and their interactions as the primary drivers of project success. Effective communication, collaboration, and teamwork are vital in Agile environments, fostering a sense of ownership and responsibility among team members.

Working software over comprehensive documentation

While documentation remains essential, Agile prioritizes the delivery of working software that meets customer needs. Frequent and incremental releases allow stakeholders to see tangible progress and provide valuable feedback throughout the development process.

Customer collaboration over contract negotiation

Agile encourages close collaboration with customers and end-users. This customer-centric approach ensures that the software being developed aligns with their evolving needs, increasing the likelihood of delivering a product that satisfies their requirements.

Responding to change over following a plan

Agile acknowledges that change is inevitable in software development. Rather than rigidly adhering to a fixed plan, Agile teams embrace change and view it as an opportunity for improvement.

Frequent iterations enable teams to adapt to new information and feedback, fostering a more responsive development process. Agile software development thrives on change and adaptability, making flexibility the heartbeat of its success.

Twelve Agile Development Principles:

the 12 principles from the Agile Manifesto :

1. Prioritize satisfying the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, with a preference for shorter timescales.
4. Collaborate closely between business people and developers throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. Use face-to-face communication whenever possible for effective information sharing.
7. Measure progress primarily through working software.
8. Maintain a sustainable pace of work for the development team. Continuous work is sustainable work.
9. Focus on technical excellence and good design to enhance agility.
10. Keep things simple and maximize the amount of work not done (avoid unnecessary tasks).
11. Allow self-organizing teams to make decisions on how to accomplish their work.
12. Reflect at regular intervals on team effectiveness and adjust behavior accordingly.

Iterative and Incremental Development

At the heart of Agile lies the concept of iterative and incremental development. Unlike traditional "waterfall" methods, where all development occurs sequentially in one large phase, Agile divides the software development process into smaller iterations or time-boxed cycles.

Each iteration results in a potentially shippable increment of the software, with new features or improvements being added in every cycle.

This iterative approach offers several advantages:

- **Early Value Delivery:** Customers can start using and benefiting from the software early in the development process, gaining tangible value with each iteration.
- **Continuous Feedback:** Frequent releases allow stakeholders to provide feedback, guiding the development direction and ensuring that the final product aligns with their expectations.
- **Risk Mitigation:** By breaking the project into smaller chunks, Agile reduces the risk associated with large-scale development, making it easier to adjust and adapt to changes.
- **Increased Transparency:** Teams and stakeholders have a clear view of progress, making it easier to identify and address potential issues or delays.

Cross-functional Teams and Collaborative Environment

Agile emphasizes the importance of cross-functional teams, where members possess diverse skills and expertise needed to deliver a complete product.

These teams work together in a collaborative environment, promoting shared responsibility, knowledge exchange, and collective ownership of project outcomes.

The cross-functional nature of Agile teams fosters a sense of empowerment. Team members are encouraged to make decisions collectively, take ownership of tasks, and resolve challenges collaboratively. This dynamism enables faster problem-solving, improved creativity, and a more resilient team structure.

Moreover, Agile methodologies often embrace practices like pair programming, where two developers work together on the same piece of code. This leads to higher code quality, knowledge transfer, and reduced knowledge silos within the team.

The fundamentals of Agile Software Development revolve around valuing people and interactions, focusing on delivering working software, collaborating closely with customers, and responding to change in a flexible manner.

By employing iterative and incremental development and fostering a collaborative team environment, Agile lays the groundwork for efficient and customer-focused software development processes.

In the next sections, we will delve into specific Agile methodologies like Scrum and Kanban to see how these principles are put into action.

Scrum: A Comprehensive Approach

Scrum is one of the most widely adopted Agile frameworks in the software development industry. It provides a structured and comprehensive approach to managing projects, enabling teams to deliver high-quality software efficiently.

In this section, we will explore Scrum in detail, understanding its framework, key roles, artifacts, ceremonies, and the benefits it brings to software development teams.

Overview of Scrum Framework

The Scrum framework is built on the foundation of Agile principles and is designed to maximize productivity, foster collaboration, and deliver value to customers.

It consists of three essential elements:

Scrum Roles

Product Owner: The Product Owner is the voice of the customer and stakeholders. They are responsible for defining and prioritizing the product backlog, ensuring that the development team is working on the most valuable features.
The Product Owner collaborates with stakeholders to gather requirements and provide feedback on delivered increments.

Scrum Master: The Scrum Master acts as a facilitator and servant-leader for the development team. Their primary role is to ensure that the Scrum framework is understood and followed correctly.
They remove any impediments that hinder the team's progress, promote a collaborative team environment, and facilitate the various Scrum ceremonies.

Development Team: The Development Team consists of professionals who do the actual work of delivering a potentially shippable product increment in each sprint. They are self-organizing, cross-functional, and collaborate closely to complete the tasks from the sprint backlog.

Scrum Artifacts

Product Backlog: The Product Backlog is a prioritized list of all the work items required to complete the project. These items can include features, enhancements, bug fixes, and technical tasks.
The Product Owner continuously refines and updates the backlog based on feedback and changing requirements.

Sprint Backlog: Before each sprint, the Development Team pulls a set of work items from the Product Backlog and creates the Sprint Backlog.
The Sprint Backlog contains the tasks the team commits to completing during the sprint. It provides transparency and a clear plan for the upcoming iteration.

Increment: The Increment represents the sum of all completed Product Backlog items at the end of each sprint. It is a potentially shippable piece of software that should be in a usable state and adhere to the team's definition of "*done*."

Scrum Ceremonies

Sprint Planning: At the beginning of each sprint, the Product Owner and Development Team collaborate in the Sprint Planning meeting. They discuss and agree on the sprint goal, select the top items from the Product Backlog, and create the Sprint Backlog with associated tasks.
Daily Standup (Daily Scrum): The Daily Standup is a brief daily meeting where the Development Team synchronizes their work. Each team member shares what they worked on the previous day, what they plan to work on that day, and any impediments they are facing.
Sprint Review: At the end of each sprint, the team holds a Sprint Review meeting to demonstrate the completed Increment to stakeholders. Feedback is gathered, and the Product Backlog is updated based on the stakeholders' input.
Sprint Retrospective: Following the Sprint Review, the team conducts the Sprint Retrospective to reflect on the previous sprint. They identify what went well, what could be improved, and define actionable items to enhance their processes in the upcoming sprints.

Benefits and Advantages of Scrum

Scrum offers a number of benefits that contribute to its popularity and success in Agile software development:

1. **Transparency:** The use of visible backlogs, frequent progress updates, and regular meetings ensures transparency among team members and stakeholders. This fosters a shared understanding of the project's status.
2. **Adaptability:** Scrum's iterative nature allows teams to adapt to changing requirements and priorities. This ensures that the delivered product remains aligned with the customer's needs.
3. **Continuous Improvement:** The Sprint Retrospective encourages continuous improvement by providing a platform for the team to reflect on their practices and identify opportunities for enhancement.
4. **Early Value Delivery:** The focus on delivering potentially shippable increments at the end of each sprint allows customers to see tangible progress early in the development process.
5. **Customer Collaboration:** The involvement of the Product Owner and regular Sprint Reviews promote active collaboration with customers, resulting in a product that better meets their expectations.

Scrum Challenges and How to Overcome Them

While Scrum is highly effective, it is not without its challenges. Some common hurdles that teams may encounter include:

1. **Overcommitment:** Teams might take on too much work in a sprint, leading to incomplete tasks and a compromised Increment. Regularly evaluating capacity and being realistic about commitments can help avoid this pitfall.
2. **Lack of Empowerment:** If team members are not empowered to make decisions and are overly dependent on the Scrum Master, the efficiency and effectiveness of the team may suffer. Encouraging self-organization and trust within the team can mitigate this challenge.
3. **Incomplete Definition of "Done":** Ambiguity about what constitutes a "done" user story can lead to misunderstandings and incomplete work. Clearly defining and agreeing upon the team's "definition of done" is crucial for consistent delivery.

4. **Product Owner Availability:** Insufficient availability of the Product Owner can slow down decision-making and result in unclear requirements. Maintaining constant communication and involvement with the team can help alleviate this issue.

Scrum provides a structured and comprehensive approach to Agile Software Development, offering a well-defined set of roles, artifacts, and ceremonies that facilitate collaboration, transparency, and continuous improvement.

By embracing Scrum's core principles and addressing its challenges proactively, development teams can harness the full potential of this framework to deliver successful software projects.

Extreme Programming (XP): A Development Approach

Extreme Programming (XP) is an Agile software development approach that embraces a set of best practices and values to deliver high-quality software efficiently.

Created by Kent Beck in the late 1990s, XP challenges traditional development practices by promoting a customer-centric and team-oriented philosophy.

In this section, we will explore the key principles and core practices of Extreme Programming and understand its impact on software development teams.

Overview of Extreme Programming

Extreme Programming is based on a set of values that drive the development process. These values include communication, simplicity, feedback, courage, and respect.

XP encourages open and frequent communication between team members and stakeholders. This simplifies processes and solutions and encourages team members to seek and act upon feedback regularly. Team members are also encouraged to have the courage to make necessary changes and respect the expertise and contributions of all team members.

Core Practices of XP

Test-Driven Development (TDD): Test-Driven Development is a fundamental practice in XP where developers write tests before writing the code.
The process involves creating a test that initially fails, then writing the code to pass the test. TDD ensures that the code is thoroughly tested. This makes it easier to identify and fix issues early in the development process.

Pair Programming: In Pair Programming, two developers work collaboratively at the same workstation. One programmer writes the code while the other reviews it in real-time. This promotes continuous feedback, knowledge sharing, and improved code quality.
This practice enhances team communication and leads to the development of more robust solutions.

Continuous Integration: Continuous Integration involves frequently integrating code changes into a shared repository. This ensures that the software is continuously built and tested as new code is added, reducing integration issues and enabling faster feedback on potential defects.
Collective Code Ownership: XP encourages the concept of collective code ownership, where all team members take responsibility for the entire codebase.
This fosters a sense of ownership and accountability within the team, leading to a collaborative and supportive work environment.

On-Site Customer: In XP, having an on-site customer or a dedicated customer representative is vital for effective communication and quick decision-making.
The on-site customer provides real-time feedback and clarifications, ensuring that the team builds the right features and meets customer expectations.

Pros and Cons of Extreme Programming

Pros:

- **High-Quality Code:** TDD and Pair Programming lead to better-tested and more maintainable code.
- **Fast Feedback:** Continuous Integration and frequent releases provide rapid feedback on code changes.
- **Customer Collaboration:** Involving an on-site customer ensures better alignment with customer needs.
- **Adaptability:** XP's practices allow teams to adapt to changing requirements and priorities effectively.

Cons:

- **Learning Curve:** Adopting XP may require a cultural shift and training for team members unfamiliar with its practices.
- **Resource Intensive:** Pair Programming and on-site customer involvement may require additional resources.
- **Initial Overhead:** Writing tests before code and maintaining continuous integration can add initial overhead.

Extreme Programming (XP) is a development approach grounded in a customer-focused philosophy and driven by a set of core practices.

By emphasizing test-driven development, pair programming, continuous integration, and collective code ownership, XP aims to deliver high-quality software while promoting effective teamwork and continuous improvement.

Like any methodology, XP has its advantages and challenges, but when applied in the right context with committed team members, it can lead to substantial improvements in software development efficiency and customer satisfaction.

Unit2:

Agile project planning is an iterative approach to project management that focuses on delivering frequent and incremental value. It promotes cross-functional collaboration and encourages ongoing improvement based on stakeholder feedback.

Unlike traditional project planning methods (the Waterfall Method, for example) that emphasize having a strict plan and timeline, agile project planning prioritizes flexibility and adaptability.

While the former method is more suited for long-term, less complicated projects that demand a rigorous structure, agile planning works best for short-term and complex development projects that require frequent stakeholder feedback.

In this article, we'll cover the essentials of agile project planning to help you get the most out of it.

The Benefits of Agile Project Planning

With user needs shifting constantly and development projects getting more complex, switching to agile can make your dev cycles more efficient. Here are some of the benefits of agile project planning:

- **Improved usability:** Incorporating feedback from users and other stakeholders ensures the software meets their need
- **Incremental delivery:** Agile projects are broken down into smaller, manageable increments, prioritizing the delivery of the most valuable features early in the project lifecycle for faster feedback loops

- **Continuous improvement:** Agile teams regularly reflect on their processes and seek opportunities to improve efficiency, quality, and effectiveness over time
- **Adaptability:** Agile offers ample room for flexibility. It encourages adapting the project plan based on changing requirements, priorities, and market conditions

Agile Planning Methodologies
Let’s discuss the three most popular agile planning methodologies in project management.

Scrum
Scrum is an agile software development framework designed to deliver value iteratively and incrementally.

This subset of agile emphasizes adopting a flexible, holistic product development strategy where the dev team works as a unit to reach a common goal.

Key elements of Scrum project management include:

- **Sprints**: Short, time-boxed work cycles where the team focuses on completing a set of deliverables from the product backlog. These cycles typically last 1-4 weeks and keep the project focused and adaptable
- **Daily stand-up meetings**: Also known as daily scrums, these are brief meetings (usually 15-20 minutes) held each day during a sprint. The team uses this time to discuss progress, identify roadblocks, and ensure everyone is aligned.
- **Product backlog**: This is a prioritized list of features, requirements, and fixes for the entire project. It’s a living document that evolves throughout the project as new information emerges
- **Sprint backlog**: A subset of the product backlog, it includes the specific list of items the development team will work on during a particular sprint. This list is created during sprint planning and reflects what the team believes they can accomplish in that timeframe
- **Sprint review meetings**: Held at the end of each sprint, the review meeting is an opportunity for the team to showcase what they’ve completed and gather feedback from stakeholders
- **Sprint retrospectives**: Another meeting held at the conclusion of a sprint, the retrospective is a chance for the team to reflect on what went well, what didn’t, and how they can improve their process for the next sprint

Key Principles of Agile Planning
Here are the four main principles that determine the direction of projects in agile planning:

1. Iterative and incremental planning

In agile planning, a project is simply broken down into small, manageable iterations or increments.

Instead of planning the entire project upfront, teams focus on preparing for the next iteration based on feedback and insights gained from previous iterations.

2. Agile planning based on user stories

User stories are short, simple descriptions of a feature or functionality from an end-user’s perspective.

They read as follows:

As a [Who], I want to [What], So that [Why]

- **As a [Who]**: This identifies the user or persona who will benefit from the functionality
- **I want to [What]**: This describes the specific goal or action the user wants to accomplish
- **So that [Why]**: This explains the benefit or value the user will receive by achieving the goal

Here’s an example of a user story written in the typical Scrum format:

As a fitness instructor, I want to be able to create and manage workout routines for my clients online so that I can provide them with personalized exercise plans and easily track their progress.

Agile planning revolves around creating and prioritizing user stories based on their value to the customer. These user stories serve as building blocks for planning and executing work during iterations, ensuring the end product meets customer expectations and preferences.

3. Division of agile project plan into releases and sprints

Agile projects are generally organized into releases and sprints.

Releases represent larger milestones or deliverables that contain a collection of features or functionalities. On the other hand, Sprints are short, time-boxed iterations (usually one to four weeks) during which teams work on a subset of user stories or tasks.

This division allows teams to deliver value incrementally, each contributing to the overall project goals.

4. The role of agile in strategic management

What’s strategic management? It is the process of managing an organization’s resources to meet its goals and objectives.

Agile principles and practices allow businesses to respond quickly to market changes, innovate, incorporate customer feedback, reduce time to market (TTM), and improve project success rates, which lead to more effective strategic management.

What are Agile Requirements?

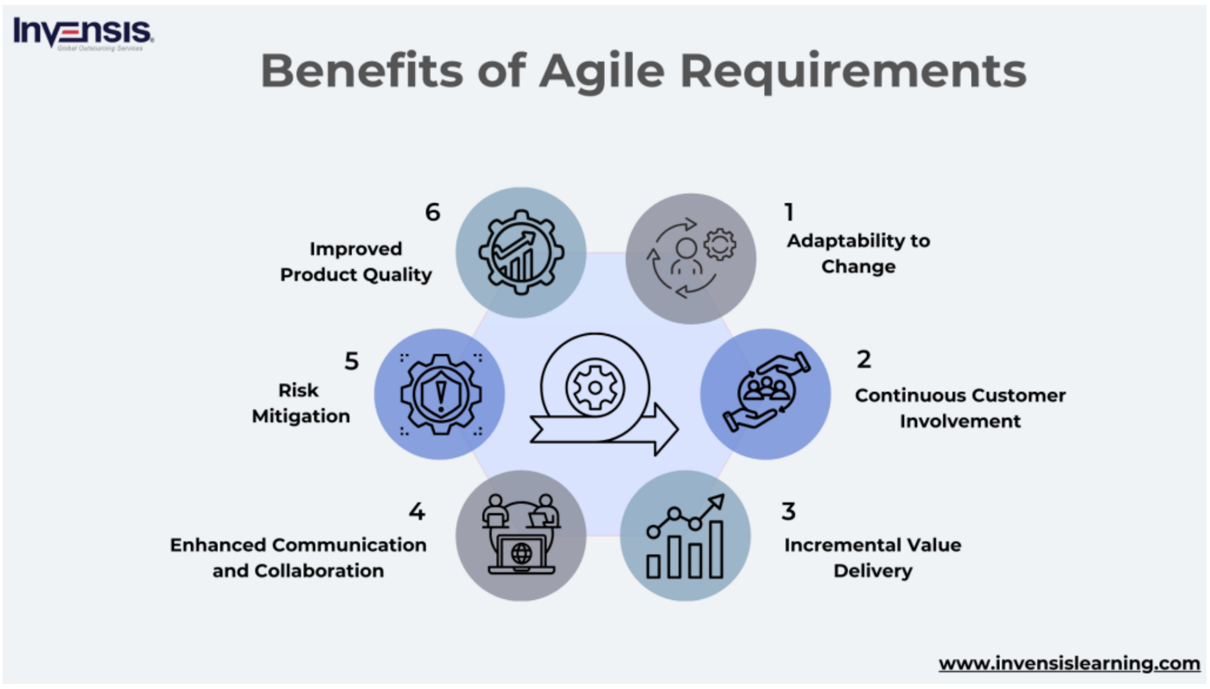
In Agile methodologies, requirements take on a dynamic and adaptive nature. Unlike traditional project management approaches, where requirements are fixed and detailed upfront, Agile requirements are fluid and responsive.

- **Collaborative Shaping:** Agile requirements are collaboratively shaped throughout the development process
- **Adaptability:** Teams can adjust to changing circumstances and priorities, a crucial aspect in Agile development
- **Emphasis on Flexibility:** Agile methodologies prioritize delivering incremental value and prompt responses to customer feedback
- **Expression through User Stories:** Agile requirements are often expressed through user stories, capturing end-users perspectives and needs
- **Prioritization by Business Value:** User stories are prioritized based on business value, ensuring a focus on the most crucial features
- **Iterative and Incremental Nature:** Agile requirements follow an iterative and incremental approach, allowing real-time adjustments.
- **Continuous Improvement Cycle:** The nature of Agile requirements fosters a continuous improvement cycle within the development process.

- **Effective Accommodation of Changes:** This approach accommodates changes effectively, enhancing overall responsiveness and project success in Agile methodologies.

Benefits of Agile Requirements

Agile requirements refer to managing and documenting project requirements within the Agile framework, a flexible and iterative methodology for software development and project management.



INVENSIS Global Outsourcing Services
Benefits of Agile Requirements
1 Adaptability to Change
2 Continuous Customer Involvement
3 Incremental Value Delivery
4 Enhanced Communication and Collaboration
5 Risk Mitigation
6 Improved Product Quality
www.invensislearning.com

Here are some benefits of Agile requirements:

- Adaptability to Change

Agile requirements prioritize flexibility, allowing teams to adjust project priorities and features as needed. This ensures the project can easily respond to changing market conditions, customer feedback, or business requirements without significant disruptions.

- Continuous Customer Involvement

Agile encourages ongoing collaboration with customers, involving them in the development process from start to finish. This iterative engagement ensures that the final product aligns closely with customer expectations, resulting in higher satisfaction and a more successful end product.

- Incremental Value Delivery

Agile requirements focus on delivering small, valuable increments of functionality in each iteration. This approach allows teams to release usable features sooner, providing stakeholders with tangible benefits early in the project and enabling quicker time-to-market.

- Enhanced Communication and Collaboration

Agile methodologies emphasize open communication and collaboration between the team and stakeholders. This ensures a shared understanding of project goals, reduces the risk of miscommunication, and fosters a more productive and cohesive working environment.

- Risk Mitigation

Agile’s iterative nature allows teams to identify and address potential risks early in development. By breaking down the project into smaller, manageable increments, teams can proactively manage challenges, reducing the overall risk and ensuring a smoother project flow.

- Improved Product Quality

Agile requirements promote continuous testing and feedback loops throughout development. This iterative testing helps identify and address issues promptly, contributing to a higher overall product quality. By addressing concerns early, teams can deliver a more reliable and robust end product.

How to Collect Agile Requirements?

Efficient strategies for gathering Agile requirements ensure alignment providing a streamlined and effective process.

Stakeholder Collaboration and Goal Determination

Engage in collaborative discussions with each project team member, including developers, customers, and stakeholders. Establish a shared goal by clarifying the ideal end state and responsibilities for everyone involved.

This ensures a cohesive understanding of the project's purpose and helps the team move forward with a unified vision. Work together for each sprint or feature to define clear objectives and expectations.

User Stories and Narrowing Requirements

Utilize user stories to capture specific functionalities from an end-user perspective. Break down these stories into clear, concise one- or two-page documents, emphasizing the most crucial information.

Many Agile tools and software systems provide a ticketing structure that facilitates the organization of requirements. Project managers can then rearrange and prioritize these requirements based on their importance and changes, providing a clear order for their release.

Iteration and Continuous Refinement

Engage in iterative cycles to continually refine and adjust requirements. Regularly review and reassess the gathered requirements to accommodate changing project needs or evolving priorities. This iterative approach allows for flexibility and adaptability, ensuring that the requirements remain aligned with the dynamic nature of Agile projects.

Finalization and Unique Identifiers

Once the team has narrowed down and established requirements, finalize them by assigning unique identifiers. These identifiers can coherently integrate requirements into schedules, project plans, or Agile boards. Consider creating a system where each requirement has a clear reference point for tracking progress and implementation throughout the workflow.

Review with Product Teams and Leadership

Before executing tasks based on the finalized requirements, conduct a thorough review with product teams and leadership. Ensure that everyone agrees on the requirements and aligns with overall project goals.

This step helps mitigate misunderstandings, validates the clarity of requirements, and secures the necessary buy-in from key stakeholders before proceeding with task execution.

Types of Agile Requirements for Project Management

Identifying Agile requirements in project management is crucial for shaping the end product and setting performance benchmarks. Here are some common types of Agile requirements integral to effective project management.

Functional Requirements (FRs)

Agile functional requirements pinpoint specific functions or features the final product must possess. Teams utilize these to outline steps for product development, establish goals, and set benchmarks to track progress.

Examples include features like a customer feedback form on a landing page or a searchable database for past invoices.

Nonfunctional Requirements (NFRs)

Nonfunctional requirements, or quality attributes, define the expected performance of a solution. Areas such as usability, security, reliability, and overall system performance fall under NFRs.

These requirements ensure the product behaves as intended, protecting against unauthorized access, ensuring reliability, and measuring user satisfaction.

User Stories

Agile teams express requirements from an end-user perspective through user stories. These stories help assess feature importance, break down complex functionalities, and guide product development to meet user needs.

These stories facilitate communication by articulating the end-user perspective, fostering a cohesive approach to feature prioritization and development alignment with user needs.

Acceptance Criteria

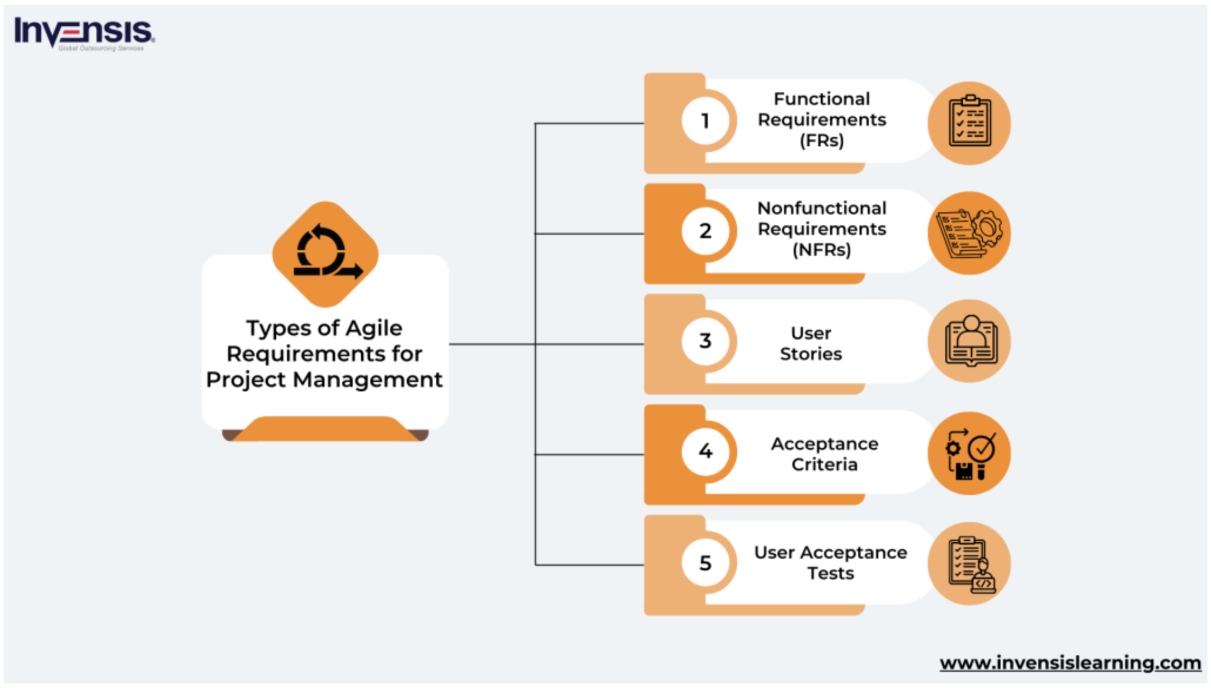
Acceptance criteria define how to test or measure the success of a user story. These criteria establish metrics to assess project success, such as improvements in customer retention or specific speed benchmarks for product dispatch.

Following the golden rule of goal setting, like SMART goal methodology, acceptance criteria ensure specific, measurable, achievable, relevant, and time-based requirements.

User Acceptance Tests

User acceptance tests provide scenarios for testing specific product or service features. These tests help teams and clients understand how features work and verify if solutions meet customer needs.

For instance, a user acceptance test for a confirmation email feature may include visiting the registration website, filling out a form, and confirming email receipt.


Invensis
Global Outsourcing Services
Types of Agile Requirements for Project Management
1 Functional Requirements (FRs)
2 Nonfunctional Requirements (NFRs)
3 User Stories
4 Acceptance Criteria
5 User Acceptance Tests
www.invensislearning.com

What are agile user stories?

A user story is the smallest unit of work in an agile framework. It's an end goal, not a feature, expressed from the software user's perspective.

A user story is an informal, general explanation of a software feature written from the perspective of the end user or customer.

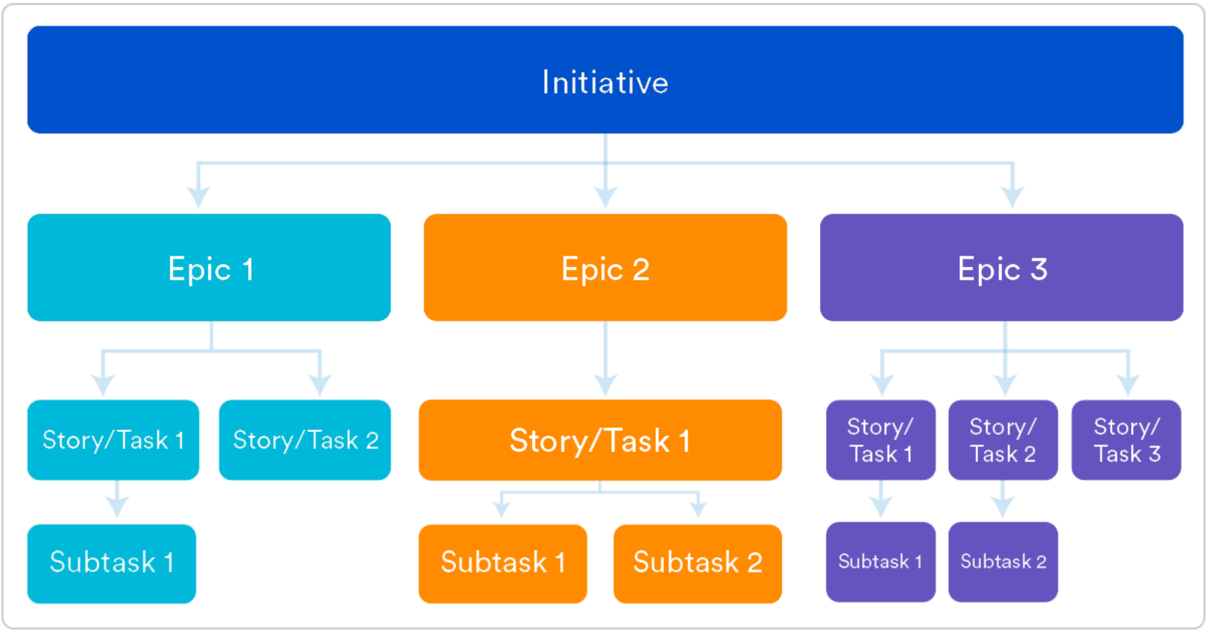
The purpose of a user story is to articulate how a piece of work will deliver a particular value back to the customer. Note that "customers" don't have to be external end users in the traditional sense, they can also be internal customers or colleagues within your organization who depend on your team.

User stories are a few sentences in simple language that outline the desired outcome. They don't go into detail. Requirements are added later, once agreed upon by the team.

Stories fit neatly into agile frameworks like scrum and kanban. In scrum, user stories are added to sprints and "burned down" over the duration of the sprint. Kanban teams pull user stories into their backlog and run them through their workflow. It's this work on user stories that help scrum teams get better at estimation and sprint planning, leading to more accurate forecasting and greater agility. Thanks to stories, kanban teams learn how to manage work-in-progress (WIP) and can further refine their workflows.

User stories are also the building blocks of larger agile frameworks like epics and initiatives. Epics are large work items broken down into a set of stories, and multiple epics comprise an initiative. These larger structures ensure that the day-to-day work of the development team (on stores) contributes to the organizational goals built into epics and initiatives.

Learn more about epics and initiatives



Why create user stories?

For development teams new to agile, user stories sometimes seem like an added step. Why not just break the big project (the epic) into a series of steps and get on with it? But stories give the team important context and associate tasks with the value those tasks bring.

User stories serve a number of key benefits:

- **Stories keep the focus on the user.** A to-do list keeps the team focused on tasks that need to be checked off, but a collection of stories keeps the team focused on solving problems for real users.
- **Stories enable collaboration.** With the end goal defined, the team can work together to decide how best to serve the user and meet that goal.
- **Stories drive creative solutions.** Stories encourage the team to think critically and creatively about how to best solve for an end goal.
- **Stories create momentum.** With each passing story, the development team enjoys a small challenge and a small win, driving momentum.

Working with user stories

Once a story has been written, it's time to integrate it into your workflow. Generally a story is written by the product owner, product manager, or program manager and submitted for review.

During a sprint or iteration planning meeting, the team decides what stories they'll tackle that sprint. Teams now discuss the requirements and functionality that each user story requires. This is an opportunity to get technical and creative in the team’s implementation of the story. Once agreed upon, these requirements are added to the story.

Another common step in this meeting is to score the stories based on their complexity or time to completion. Teams use t-shirt sizes, the Fibonacci sequence, or planning poker to make proper estimations. A story should be sized to complete in one sprint, so as the team specs each story, they make sure to break up stories that will go over that completion horizon.

How to write user stories

Consider the following when writing user stories:

- **Definition of “done”** — The story is generally “done” when the user can complete the outlined task, but make sure to define what that is.
- **Outline subtasks or tasks** — Decide which specific steps need to be completed and who is responsible for each of them.
- **User personas** — For whom? If there are multiple end users, consider making multiple stories.
- **Ordered Steps** — Write a story for each step in a larger process.
- **Listen to feedback** — Talk to your users and capture the problem or need in their words. No need to guess at stories when you can source them from your customers.
- **Time** — Time is a touchy subject. Many development teams avoid discussions of time altogether, relying instead on their estimation frameworks. Since stories should be completable in one sprint, stories that might take weeks or months to complete should be broken up into smaller stories or should be considered their own epic.

Once the user stories are clearly defined, make sure they are visible for the entire team.

User story template and examples

User stories are often expressed in a simple sentence, structured as follows:

“As a [persona], I [want to], [so that].”

Breaking this down:

- "As a [persona]": Who are we building this for? We’re not just after a job title, we’re after the persona of the person. Max. Our team should have a shared understanding of who Max is. We’ve hopefully interviewed plenty of Max’s. We understand how that person works, how they think and what they feel. We have empathy for Max.
- “Wants to”: Here we’re describing their intent — not the features they use. What is it they’re actually trying to achieve? This statement should be implementation free — if you’re describing any part of the UI and not what the user goal is you're missing the point.
- “So that”: how does their immediate desire to do something this fit into their bigger picture? What’s the overall benefit they’re trying to achieve? What is the big problem that needs solving?

For example, user stories might look like:

- As Max, I want to invite my friends, so we can enjoy this service together.
- As Sascha, I want to organize my work, so I can feel more in control.
- As a manager, I want to be able to understand my colleagues progress, so I can better report our sucess and failures.

This structure is not required, but it is helpful for defining done. When that persona can capture their desired value, then the story is complete. We encourage teams to define their own structure, and then to stick to it.

Getting started with agile user stories

User stories describe the why and the what behind the day-to-day work of development team members, often expressed as *persona + need + purpose*. Understanding their role as the source of truth for what your team is delivering, but also why, is key to a smooth process.

Start by evaluating the next, or most pressing, large project (e.g. an epic). Break it down into smaller user stories, and work with the development team for refinement. Once your stories are out in the wild where the whole team can see them, you're ready to get to work.

What is a product backlog?

A product backlog is a prioritized list of tasks — including new features to build, bugs to fix, improvements to implement, etc. — that is derived from the product requirements and roadmap. It is a scrum artifact that serves as a to-do list for agile development teams.

Let's define product backlog even more granularly. According to *the Scrum Guide*, a product backlog is "an emergent, ordered list of what is needed to improve the product. It is the single source of work undertaken by the Scrum Team."

Breaking down this definition piece by piece:

- **Emergent** — The product backlog is not a plan; it evolves as you learn. If you're not removing old items, you're either not learning or getting stuck with the past and blocking the future
- **Improve the product** — A backlog is neither a wishlist, nor a set of requirements, nor promises. It contains items that contribute to value for users and businesses
- **Single source of work** — You should have only a single product backlog. Having separate tech, discovery, and idea backlogs is a common mistake. Doing so creates silos and stymies collaboration

Here's the bottom line: the simpler your product backlog, the better.

What does a product backlog include?

When considering what to include in a product backlog, the key is to focus on collaboration instead of prescriptive items.

In theory, the scrum team can choose how to organize and format the product backlog. That may work if you have enough seniority among the team. If not, things will quickly derail.

The following items are commonly included in a product backlog:

- Epics
- User stories
- Bugs
- Tasks
- Spikes

Epics
An epic represents a big picture of something you want to achieve. It should name the desired result and why that matters.

An example of an epic might be implementing search recommendations with the expected outcome of increasing conversion rate.

User stories
A user story names the problems you want to solve and why they matter.

User stories should be enablers to reaching the goal of epics. But keep in mind that the focus is on collaboration and not contracts. Keep your stories as invitations for conversations and refine them with your teams.

Bugs
A bug is an unexpected behavior. When creating bugs, you should precisely name how to reproduce them and estimate the importance and percentage of impacted users.
Tasks
The team doesn't work only on features. They also have some jobs to keep the lights on.

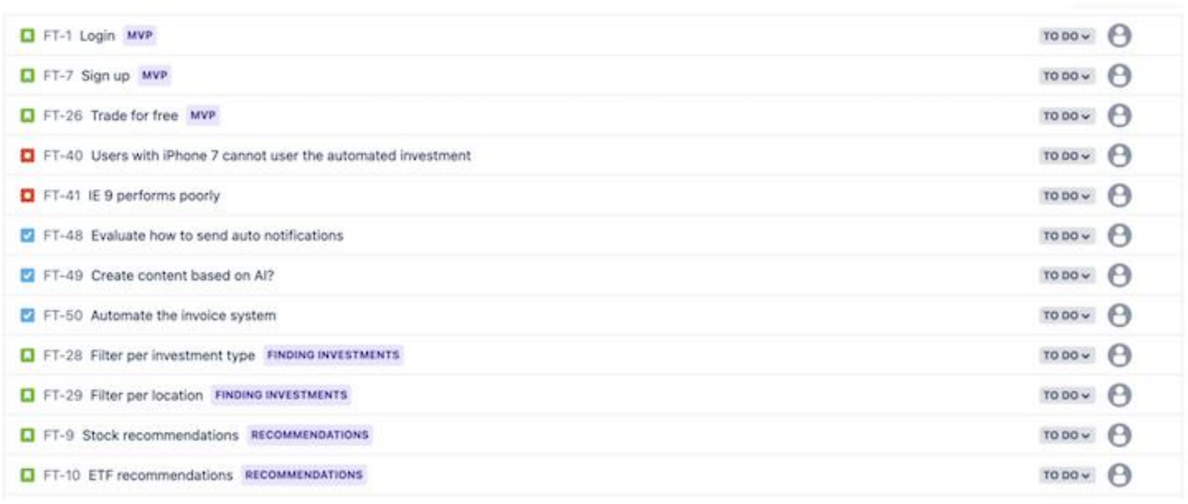
Generally, this is best represented by a task — for example, "set a backup routine," or "update the angular version."

Spikes
Sometimes, the team doesn't know where to start. They may have understood the problem but solving that requires some research.

A spike is good for those cases. You define a goal and timebox it.

Product backlog example
The product backlog item types described above intend to simplify how you work and set clear expectations for each type. I recommend limiting it to five types; don't go wild. Otherwise, confusion is the result.

The following image represents an example of a product backlog for an investment platform:



FT-1	Login	MVP	TO DO
FT-7	Sign up	MVP	TO DO
FT-26	Trade for free	MVP	TO DO
FT-40	Users with iPhone 7 cannot user the automated investment		TO DO
FT-41	IE 9 performs poorly		TO DO
FT-48	Evaluate how to send auto notifications		TO DO
FT-49	Create content based on AI?		TO DO
FT-50	Automate the invoice system		TO DO
FT-28	Filter per investment type	FINDING INVESTMENTS	TO DO
FT-29	Filter per location	FINDING INVESTMENTS	TO DO
FT-9	Stock recommendations	RECOMMENDATIONS	TO DO
FT-10	ETF recommendations	RECOMMENDATIONS	TO DO

Who is accountable for prioritizing the product backlog?

The content of a product backlog must be aligned with your product strategy, which hopefully aligns with your product vision.

The items inside of your product backlog enable your team to get a step closer to the goals you're pursuing. If not, they should be removed.

Collaboration is a core part of product backlog management. When you're aligned, managing the product backlog becomes a piece of cake.

Roles involved in product backlog management include the product manager, stakeholders, and agile teams. Let's take a closer look at the responsibilities of each:

Product manager
The product manager is ultimately responsible for the product backlog's content and prioritization. However, that doesn't mean that PMs must work alone. Great professionals set the right context and let the team evolve the product backlog content.
Stakeholders
Many teams encourage stakeholders to create tickets in the product backlog. That often gets in the way of collaboration because it becomes a service-provider relationship.

A better way is to focus on partnering with stakeholders and endeavoring to understand their needs and how they align with the strategy and vision.

Agile teams
Team members are encouraged to create product backlog items. For example, software engineers commonly create tech debt items and designers point out usability glitches.

That said, it's the product manager's responsibility to decide what makes it to the top of the list. The secret is to inform the why behind each item and what that enables.

Product backlog vs. sprint backlog

Scrum includes both a product backlog and a sprint backlog, which often leads to confusion. Let's look to *the Scrum Guide* to clarify the distinction:
"The Sprint Backlog is composed of the Sprint Goal (why), the set of Product Backlog items selected for the Sprint (what), as well as an actionable plan for delivering the Increment (how)."
In summary, the sprint backlog contains what the team will focus on during the sprint cycle. It's often a subset of the product backlog focused on reaching the sprint goal.

The following table highlights the differences between a product and sprint backlog:

	Product backlog	**Sprint backlog**
Focus	Enable the team to achieve the product goal	Enable the team to achieve the sprint goal
Responsibility	Product owner/managers	Scrum team
Content	Epics, user stories, bugs, spikes, and tasks	Subset of tasks that level up to the sprint goal
Crafting	The entire scrum team	Developers (software engineers, UX designers, QA, data analysts)
Evolving	Refinement sessions	Sprint planning

How does good product backlog management help your team stay agile?

The product backlog is a strong indicator of a team's agile maturity. When done right, it will help your team benefit from agility.

The more you focus on collaboration, the more agile you become. The more you focus on processes, the less agile you become.

Below are some best practices for effective backlog management:

- Be goal-oriented
- Keep it simple
- Don't keep dinosaurs
- Ditch definitions of ready

Be goal-oriented
All items inside the product backlog must be related to an ultimate goal. If backlog items don't level up to an established goal, you should remove them.
Keep it simple
Keep your backlog items focused on opportunities and problems. Don't write detailed requirements.

To put it simply, a developer should not be able to pick an item from the backlog and implement it without talking to you. Focus on talking to your team and building understanding together. Otherwise, your team will descend into a waterfall mindset.

Don't keep dinosaurs
The older a backlog item is, the less relevant it becomes. The quicker you remove outdated items, the faster you can focus on your learnings.
Ditch definitions of ready
Some teams like having standards for product backlog items. This can hurt your collaboration.

I would discourage you from using definitions of ready because the focus shifts to meeting standards instead of collaborating with your team.

As stated in the Agile Product Manifesto, "Keep the Product Backlog clean. Backlog items age like milk, not like wine."

Backlog management antipatterns to avoid

It's equally important to understand what to do and what not to do. We often focus on getting things done right but still fall into common pitfalls.

There are many common antipatterns associated with product backlog management. We've already touched on some of them, but here is a more complete list of traits your product backlog should NOT have:

- Wishlist
- Multiple product backlog
- Extensive requirements
- No space for learning
- Strict format

Wishlist
Your job isn't to populate the product backlog with all your stakeholders' wishes. You must ensure your items contribute value and are related to a common goal.
Multiple product backlogs
I mentioned this already, but it's worth emphasizing.

There should be only one product backlog per product. You will be pressured to maintain several, but confusion becomes inevitable once you fall into that.

Extensive requirements
Remember that the product backlog is a vehicle to create value. Unlike waterfall projects, it's not a place to drop extensive requirements and demand people to deliver them.

Meaningful product backlog items describe problems, why they matter, and expected outcomes. In short, the whole team is part of the solution.

No space for learning
The longer your product backlog is, the more pressure your team has to deliver.

An extreme backlog cleaning is essential. It's like your house: if you don't remove the dust, eventually, you cannot get in anymore.

The backlog is just the same: you won't have mental space for new items if you don't remove outdated items.

Strict format
Although I named user stories a type of product backlog item, each team can choose what fits them best.

For example, you can work with job stories, use cases, jobs to be done, etc.

The point is, the team should focus on creating value faster instead of precisely maintaining the product backlog.

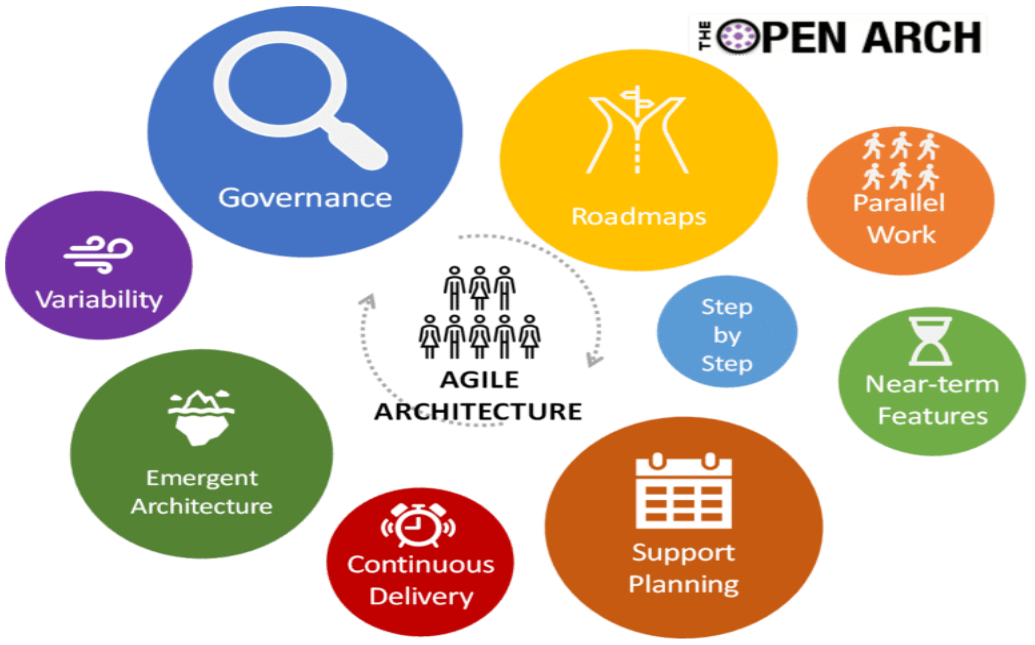
To quote the Agile Product Manifesto again, "Creating value for end-users and businesses is what defines success."

Agile Architecture:

Agile Architecture, as defined in the © Scaled Agile framework (SAFe), "*is defined by a set of values, practices, and collaborations that support the active, evolutionary design and architecture of a system. This approach embraces the DevOps mindset, allowing the architecture of a system to evolve continuously over time, while simultaneously supporting the needs of current users.*"
In comparison to the classic Waterfall delivery approach, the **Agile delivery approach focus is on delivering value fast, through quick and small iterations that maximise return on investment and increase business agility**. When Agile delivery is introduced in an organisation, the maturity level of all business units should be developed including the maturity of the Architecture Team. Agile Architecture is different from the legacy approach for many reasons, some of the most important are:

- Agile architects define "just enough" architectural runway (refer to section below for the definition) to support the evolving business needs.
- Agile development avoids Big Designs Up-front, reducing the design effort to the minimum necessary to iterate the solution.
- The Architecture evolves over time.
- Delivery in production is fast as agile avoids overhead and delays associated with phased/gate approaches.
- The solution is quickly brought live and kept live to maximise its return on investment.

When approaching Agile Architecture for the first time there are some core aspects an architect should consider.



Unit3:

Lean Software Development: Agile with a Focus on Value
Lean Software Development is an Agile approach inspired by the principles of lean manufacturing, which originated from Toyota's production system. It aims to maximize customer value while minimizing waste in the software development process.

Lean principles emphasize efficiency, continuous improvement, and a relentless focus on delivering value to customers.

In this section, we will delve into the core principles of Lean Software Development and explore how it complements Agile methodologies, such as Scrum and Kanban.

Understanding Lean Principles

There are some fundamental principles behind this approach, and they are:

1. **Eliminate Waste:** Lean Software Development advocates the elimination of non-value-adding activities, often referred to as "waste." This includes avoiding unnecessary bureaucracy, reducing delays, and optimizing resource utilization to ensure that valuable work takes precedence.
2. **Amplify Learning:** Lean promotes a learning culture, where teams continuously seek feedback and insights from customers and stakeholders. This learning mindset drives continuous improvement, enabling teams to deliver higher-quality products that better align with customer needs.
3. **Decide as Late as Possible:** Rather than making significant decisions early in the development process when information is limited, Lean encourages postponing decisions until they are necessary. This allows teams to leverage up-to-date information and make informed choices.

4. **Deliver as Fast as Possible:** Lean Software Development places a premium on rapid value delivery. By shortening the cycle time between idea inception and implementation, teams can respond quickly to changes and deliver customer value sooner.
5. **Empower the Team:** Lean emphasizes the importance of empowering and trusting team members to make decisions and contribute to the development process. This autonomy fosters a sense of ownership and accountability, driving motivation and creativity.

Application of Lean in Software Development

Lean principles can be applied in various areas of software development to improve efficiency and value delivery:

- **Value Stream Mapping:** By mapping the entire software development process, teams can identify bottlenecks, inefficiencies, and areas of waste. This helps streamline workflows and optimize the delivery process.
- **Minimal Viable Product (MVP):** The concept of MVP aligns with Lean principles, where the focus is on delivering the smallest set of features that provides value to customers. This enables faster market validation and feedback gathering.
- **Just-In-Time (JIT) Production:** Applying JIT principles in software development means delivering work items when they are needed and avoiding stockpiling uncompleted features. This reduces inventory waste and ensures a more responsive development process.
- **Kaizen:** The principle of Kaizen, or continuous improvement, is central to Lean Software Development. Teams regularly reflect on their processes and practices, seeking ways to optimize and refine their approach.

How Lean Complements Scrum and Kanban

Lean Software Development is highly compatible with other Agile methodologies, such as Scrum and Kanban:

Scrum and Lean

Scrum's iterative and incremental development aligns with Lean's focus on delivering value early and frequently.

By incorporating Lean principles like eliminating waste and amplifying learning, Scrum teams can enhance their effectiveness and responsiveness.

Kanban and Lean

Kanban's emphasis on visualizing work, reducing WIP, and promoting continuous flow aligns seamlessly with Lean's core principles. Kanban's focus on delivering value continuously complements Lean's customer-centric approach.

Lean Software Development enriches the Agile landscape with its value-focused philosophy and waste reduction strategies. By embracing Lean principles, teams can optimize their workflows, foster a culture of continuous improvement, and deliver software products that truly meet customer needs.

Lean's compatibility with other Agile methodologies makes it a powerful complement to approaches like Scrum and Kanban. It provides a holistic and efficient way to drive innovation, reduce waste, and maximize customer value in software development.

Agile Project Management Tools and Software

Agile Project Management tools and software play a pivotal role in streamlining and enhancing the efficiency of Agile development processes. These tools provide teams with a centralized platform to plan, track, and collaborate on projects. They can make it easier to manage tasks, monitor progress, and facilitate seamless communication among team members.

In this section, we will explore some popular Agile Project Management tools and software, along with the benefits they offer to Agile teams.

Project Tracking and Collaboration Tools

Jira was developed by Atlassian, and is one of the most widely used project management tools, particularly in Agile development.
It offers a range of features, including user story and task management, sprint planning, backlog prioritization, and real-time progress tracking.

With customizable workflows and extensive reporting capabilities, Jira provides teams with a comprehensive platform to manage their Agile projects efficiently.

Trello is another offering by Atlassian. It's a visual project management tool that allows teams to organize tasks into boards, lists, and cards. It is simple to use and ideal for small to medium-sized projects.
Trello's intuitive interface and drag-and-drop functionality make it easy to track progress, assign tasks, and collaborate with team members.

Azure DevOps (formerly Visual Studio Team Services) is a comprehensive toolset that includes version control, project planning, continuous integration, and release management capabilities. Its Agile Boards provide flexible backlogs, sprint planning, and real-time task tracking, making it a popular choice for teams following Agile methodologies.

Benefits of Agile Project Management Tools

1. **Improved Transparency:** Agile Project Management tools provide a centralized view of project progress, tasks, and priorities. This transparency enables stakeholders to have a clear understanding of project status and facilitates open communication among team members.
2. **Enhanced Collaboration:** These tools promote seamless collaboration among distributed teams by providing a centralized space to share updates, files, and feedback. Features like commenting and tagging team members make it easier to communicate and address issues effectively.
3. **Streamlined Workflows:** Agile Project Management tools automate repetitive tasks, streamline workflows, and ensure that project tasks flow smoothly from inception to completion. This automation reduces manual overhead and allows teams to focus on delivering value.
4. **Real-time Reporting:** The real-time reporting and data visualization capabilities of these tools provide insights into team performance, sprint progress, and project trends. Teams can use this data to identify bottlenecks, make data-driven decisions, and continuously improve their processes.

5. **Scalability:** Agile Project Management tools cater to projects of varying sizes and complexities, from small startups to large enterprises. They can adapt to different team structures, making them versatile and suitable for diverse Agile implementations.

Popular Tools for Scrum, Kanban, and Other Agile Methodologies

Scrum-Specific Tools

- Targetprocess: A comprehensive tool tailored for Scrum and Agile teams with features like sprint planning, release forecasting, and progress tracking.
- Sprintly: A user-friendly tool that focuses on sprint planning, bug tracking, and team collaboration.

Kanban-Specific Tools

- LeanKit: An advanced Kanban tool with customizable boards, analytics, and workflow automation.
- Kanbanize: A feature-rich platform with analytics, time tracking, and integrations for managing Kanban projects.

All-in-One Agile Tools

- VersionOne: An end-to-end Agile management tool that supports Scrum, Kanban, and SAFe frameworks.
- Monday.com: A versatile collaboration platform that can be customized for various Agile workflows and methodologies.

Agile Project Management tools and software provide indispensable support to Agile development teams, promoting transparency, collaboration, and streamlined workflows.

From Scrum-specific tools to all-in-one Agile platforms, these tools offer a wide range of features and customization options to suit the needs of different teams and projects.

By leveraging these tools, Agile teams can enhance their productivity, drive successful project delivery, and embrace the iterative and customer-focused essence of Agile Software Development.

How to Choose the Right Agile Approach for Your Team

As Agile Software Development has become increasingly popular, various Agile methodologies have emerged, each with its unique strengths and applicability.

The key to successful Agile implementation lies in selecting the approach that best aligns with your team's goals, project requirements, and organizational culture.

In this section, we will explore essential factors to consider when choosing the right Agile approach for your team. I'll also provide insights into tailoring Agile practices to suit your organization.

Factors to Consider When Selecting an Agile Methodology

Project Scope and Complexity: Assess the size and complexity of your project.
Scrum is well-suited for projects with a defined scope and set timelines, making it ideal for product development. On the other hand, Kanban's flexibility works best for projects with constantly changing requirements or continuous flow needs.

Team Structure and Expertise: Consider the composition of your team and their experience with Agile methodologies.
Teams with diverse skills and extensive Agile experience may be more inclined to adopt Extreme Programming (XP) with its emphasis on practices like TDD and pair programming. Conversely, teams with less Agile experience might find Scrum's structured framework easier to implement.

Customer Engagement: The level of customer engagement and the need for constant customer feedback are crucial factors.
If you have direct access to customers and require frequent feedback, Scrum's focus on customer collaboration through ceremonies like Sprint Reviews is advantageous.

Development Environment: Evaluate your organization's development environment. If you work in an environment where tasks continuously arise with no fixed iteration timelines, Kanban's continuous flow model can better accommodate these dynamic workflows.
Organizational Culture: Analyze your organization's culture and willingness to embrace Agile practices. Some Agile methodologies, like Scrum, require significant changes in project management and team dynamics, which may necessitate strong support from management and a cultural shift.

How to Tailor Agile Practices to Suit Your Organization

- **Hybrid Approaches:** Don't be afraid to adopt a hybrid Agile approach that combines elements from different methodologies. For example, you can use Scrum for project planning and sprint-based development while implementing Kanban's continuous flow model for support and maintenance tasks.
- **Iterative Adaptations:** Agile is all about continuous improvement and adaptation. Encourage your team to inspect and adapt their Agile processes regularly. This iterative approach allows you to fine-tune your practices to better suit your team's needs and project requirements.
- **Training and Coaching:** Provide Agile training and coaching to team members, especially if your organization is new to Agile methodologies. Proper education can help teams understand the principles and practices, fostering a smoother adoption process.
- **Flexibility in Scaling:** As your team grows and takes on more significant projects, consider the scalability of your chosen Agile approach. Some methodologies, like Scrum, have well-defined scaling frameworks like SAFe (Scaled Agile Framework) and LeSS (Large-Scale Scrum), which can be tailored to fit larger teams and complex projects.

Choosing the right Agile approach for your team requires a thoughtful analysis of project requirements, team dynamics, and organizational context.

By considering factors such as project scope, team expertise, customer engagement, and organizational culture, you can make an informed decision on which Agile methodology aligns best with your team's needs.

Remember that Agile is not a one-size-fits-all solution. The key to successful Agile implementation lies in adapting and tailoring Agile practices to suit your unique circumstances and goals.

Agile Scaling: Beyond the Team Level

Agile methodologies have proven to be highly effective at the team level, promoting collaboration, flexibility, and value-driven development.

But as organizations grow and undertake more extensive and complex projects, you'll need to scale Agile practices beyond individual teams.

Agile Scaling addresses the challenges of coordinating multiple teams, aligning strategic goals, and ensuring seamless communication across the organization. In this section, we will explore the concept of Agile Scaling and some popular frameworks that facilitate Agile implementation at the enterprise level.

Understanding Agile Scaling

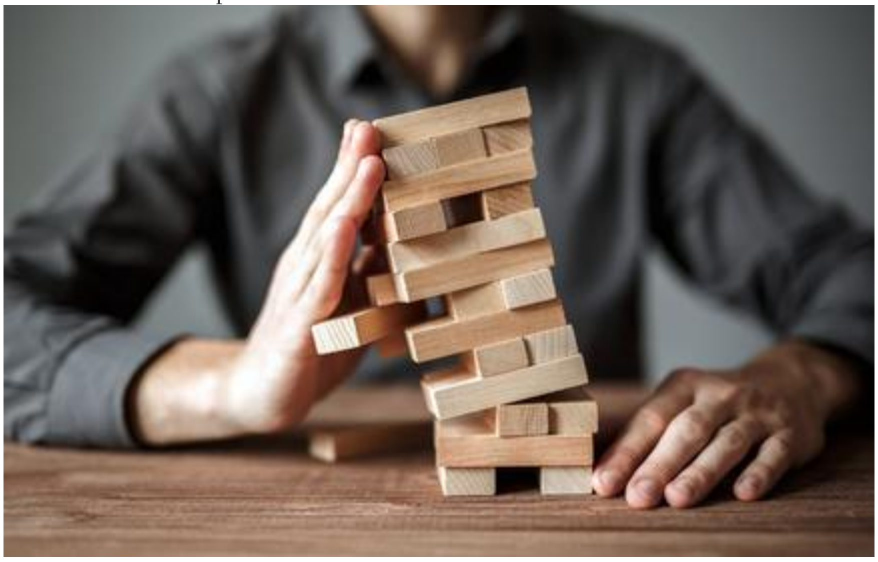
Agile Scaling involves applying Agile principles and practices across an entire organization to ensure that multiple teams work cohesively towards common goals.

At this level, Agile emphasizes cross-team collaboration, continuous integration, and maintaining a cohesive vision throughout the organization. The objective is to extend the Agile mindset beyond the team level and achieve an Agile culture at the enterprise level.

What is Agile Risk Management? How Is It Different?

Some people might think that Agile Risk Management is an oxymoron:

- There is a common stereotype that an Agile project is totally unplanned
- So, why would you take a planned approach to Agile Risk Management if the whole project is unplanned?



There is always some level of planning in an Agile project even though the level of planning may be limited. Here's an article with more detail on that:

Why Is Planning So Difficult? Is It a Waste of Time?

The gist of this is that you have to adapt the planning approach to the level of uncertainty in the project. A similar thing is true regarding risk management:

- There is no single approach to doing risk management and
- It's not a binary choice between zero risk management and a totally rigid and controlled approach to risk management.

You need to fit the risk management approach to the nature of the project:

- For high risk projects where the customer is very sensitive to risk, it makes sense to take a planned approach to risk management
- For lower risk projects , a more informal approach to risk management may be appropriate.

Agile Risk Management Process

The overall process for doing risk analysis in an Agile environment is generally the same as a traditional, plan-driven project; however, it may not be as formal and it may not be as disciplined. The general approach follows these stages:

Phase	Description
Risk Identification	This might consist of a brainstorming session to identify potential risks in the project
Risk Analysis	This involves further study to determine the probability and impact of each risk
Risk Response	This phase involves determining what, if anything, should be done to mitigate the risk
Monitoring and Control	Finally during the course of the project, the risks are monitored and controlled

Advantages of an Agile Risk Management Approach

An Agile approach is inherently well-designed for dealing with risks:

- Risks are generally directly related to uncertainty in a project and an Agile approach is intended to be flexible and adaptive in order to deal with uncertainty
- For that reason, it is easier to adapt to risks in an Agile environment as the project is in progress

Risk Management in a Plan-driven Environment

In a traditional, plan-driven project:

- A considerable amount of re-planning may be necessary to adapt to risks as the project is in progress and
- For that reason, it may be more important to plan for risks upfront in a plan-driven environment.

Structuring an Agile Project for Risk Management

Another factor is due to the iterative and incremental nature of development in an Agile project:

- It's not too difficult to structure the Product Backlog to address high risk items early in the project and,
- If there is a lot of uncertainty associated with those risks, a "spike" can be performed to evaluate the risk without having a major impact on the project.

Responsibility for Agile Risk Management

It's easy to lose focus on risk management in an Agile environment because there is no well-defined focal point of responsibility for risk management. Risk management is normally a project management responsibility and there is typically no project manager at the team level in an Agile project:

- In an Agile environment, the entire team owns responsibility for risk management. In a similar way, the the entire team owns responsibility for project management
- Another factor is that because an Agile approach is more adaptive to risks, there tends to be a “cavalier” approach to not worry about risks but it doesn’t have to be that way
- You can do as much (or as little) risk management as necessary depending on the nature of the project and the sensitivity to risk

Unit4:

Agile Project Tools:

Agile is a project management methodology used to break complex projects into smaller, more focused chunks so teams can work in short, incremental phases. As its name suggests, “agile” means moving fast and managing shifting priorities. With smaller projects known in software development as sprints, businesses deliver greater value to customers through an iterative approach with continuous releases.

To support the agile approach, there is a wide array of agile project management tools available. These tools help teams with planning, visualization, collaboration, process streamlining, and maintaining focus throughout the entire project journey.

If you're in the process of selecting the right agile project management tool for your growing business, this guide will introduce you to the top nine options. It aims to provide you with the information you need to make an informed decision that will contribute to your team's success.

1. Best for issue tracking and sprint planning: Jira

Jira is purpose-built for software teams, who rely on agile ways of working. It keeps teams focused, motivated, and on schedule by breaking large, complex projects into smaller, more manageable sprints.

As a sprint planning tool, it supports refining product backlogs, generating and prioritizing ideas, documenting user stories, and defining the tasks required to achieve a successful release. Project managers can create a continuous delivery pipeline with agile project plan templates, as well as templates for agile collaboration at scale. And, it’s free for up to 10 users.

2. Best for aligning with cross-functional teams: Jira

Coordinating projects that involve multiple teams, such as software development and marketing, can be a complex task. Each team has its own way of working based on its specific discipline and focus. When non-software teams adopt agile project management methods, it’s especially important to recognize and address the differences in how each team works.

With Jira, agile project management is synchronized across the organization to encompass all resources and departments involved. Work management solutions allow cross-business teams to identify dependencies between business areas, manage resource allocation, and track tasks related to sprints, and even epics, in one visual workflow.

3. Best for sprint retrospectives: Confluence

Sprint retrospectives are one of the most important continuous improvement tools available to an agile development team. They provide an opportunity for everyone to share what went well and where there’s room for improvement. The most successful teams look beyond a single sprint and instead, view trends across multiple projects. An anomaly observed in one sprint may actually be a recurring issue that becomes more apparent when examined across projects.

Confluence provides a real-time, centralized knowledge management platform for collaborating on everything from customer feedback to sprint retrospectives. With the sprint retrospective template, Confluence gives agile project teams the tools to capture important feedback for both immediate and future improvement.

4. Best for backlog management: Jira

Ongoing agile project management involves effectively managing the product backlog. This involves organizing work into logical groups, prioritizing tasks based on the project roadmap, and documenting requirements to ensure the team understands what to build and why. This is often referred to as Scrum backlog refinement.

Think of Agile as the overarching methodology guiding your work and Scrum as the structured framework of activities that facilitates its successful implementation. Jira includes Scrum templates for backlog refinement, task prioritization, dependency mapping, and resource optimization, making backlog management easier. When the team is working from a well-organized and prioritized backlog, sprint planning is easier and delivery is faster.

5. Best for project tracking and reporting: Jira

When it comes to real-time insights, Jira provides what you need when you need it. During sprint planning, insights into the backlog help teams plan smarter sprints. During the development process, agile metrics, dashboards, and insights are available right on the board, giving teams the information they need to make real-time, data-driven decisions. Then, development frequency and cycle time reports help optimize the delivery pipeline.

6. Best for daily standups: Zoom

Today's teams are often highly dispersed. For agile teams who use a scrum standup as a daily check-in to discuss the sprint progress, address roadblocks, and collaborate, this can present a significant challenge.

Zoom video hosting brings the team together as if they were all in the same room. Screen sharing allows the team to discuss the project board or capture decisions, the recording feature allows absent team members to catch up on discussions at a later time, and the chat function allows for the real-time sharing of links and other essential information.

7. Best for collaboration and knowledge management: Confluence

Agile projects include their own internal documentation, which team members use to meet the sprint goals. Whether it's capturing user stories, defining requirements and product specifications, or external information like product certifications, information must be linked to the project, readily available, and always discoverable by all stakeholders.

Confluence is a central repository that brings knowledge management into the product development lifecycle. It houses relevant project information and supports easy collaboration. Contributors can add text, images, code, and tables and attach files for true collaboration.

8. Best for task and workflow automation: Jira

Jira is an effective tool for task management because it provides real-time visual status updates so agile teams can respond to changes as they arise. Jira automation lets people focus on what's important by automating repetitive tasks. Anyone on the agile team can sync tasks across projects and products because it doesn't require special coding. Jira templates make it easy to get started quickly, and Jira automation integrates with other commonly used tools such as Slack, Open Devops, Bitbucket, GitHub, and more.

9. Best for async video collaboration: Loom

Task switching and interruptions slow down work. Informing team members about project status, roadmap updates, or new strategies doesn't always require immediate disruptions. Asynchronous communication allows team members to engage with new information when it best suits their workflow.

Loom lets agile project managers turn statuses and other information into videos, allowing team members to view them at their own convenience. This tool is especially helpful for geographically dispersed teams to remain informed and included, no matter what time zone they are in.

Continuous Integration in Agile

The quality of software depends on the quality of its development. However, to reach the desired product quality, development plans and methods may deviate from the original estimations. Customer requirements also tend to change with time. Any system that successfully accommodates and implements these changes is bound to create better software.

This is precisely what the Agile development methodology does. This article will focus on a pivotal stage in agile development – Continuous Integration.

Let's begin with a glimpse of pre-Agile models, AKA the traditional Software Development Life Cycle (SDLC) models.

Table of Contents

Traditional SDLC models

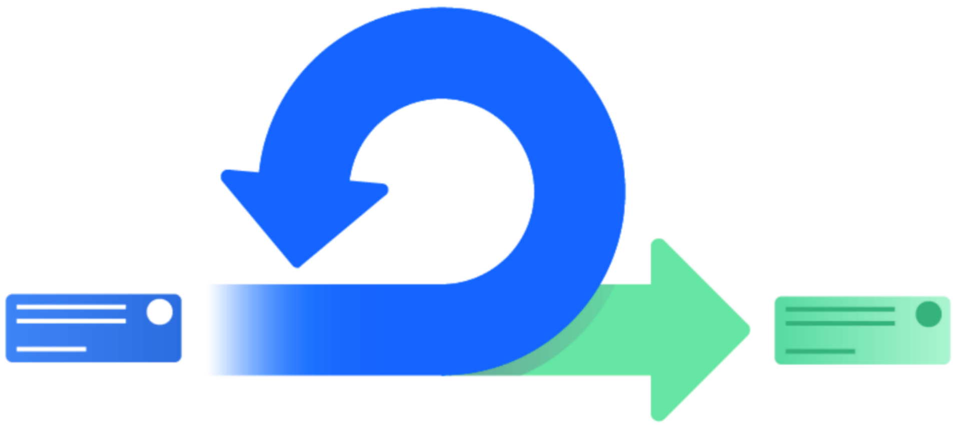
The process through which software is developed is termed the SDLC – Software Development Life Cycle. The blueprint defines the various steps to be followed to build software.

- The **Waterfall Model** and **Spiral Model** were two frequently used traditional SDLC models.
- The Waterfall Model comprises a step-by-step approach – linearly moving through requirements gathering, design, implementation, testing, and maintenance. This model did not work well when customers expected changes because it provided little scope to include modifications once the design phase was finished.
- To counter this, many organizations started following the **Spiral Model of SDLC.** In this model, the first cycle began with building an initial prototype: the **Minimum Viable Product (MVP)**. Based on this, the customer would be given a chance to suggest any necessary changes. All the SDLC stages would be followed if changes were recommended to build a second prototype.

The advent of Agile

Until the early '80s, the software would be developed based on the principles of physical engineering. During this time, the demand for automation in mechanical processes skyrocketed, leading to fierce competition among software development companies. Traditional SDLCs were inadequate, as they often took too much to meet deviations from initial requirements.

End products were not stable enough and kept failing to meet customer expectations. Naturally, the software development fraternity required an SDLC model that solved the problems mentioned above. The new model would have to maintain the positives of traditional SDLCs and adapt and adjust to changing requirements at short notice. Simply put, the SDLC model had to be more "**Agile**."



The linearity of the Waterfall Model and the iterative nature of the Spiral Model along with a few other concepts, formed the basic framework for Agile.

Learn More: Agile Development Methodologies: An Essential Guide

What is Agile?

Agile is an iterative and incremental approach to software development with a baseline of adaptability for customer requirements. To adapt to changes, tasks had to be broken into smaller modules. Using smaller tasks with lesser iterations means devs do not have to change large chunks of code if requirements alter or deviate. Milestones are set to mark the completion of a development phase. A constant customer review process helps shape the product to a desirable state. This method appealed to development organizations. Many started implementing it and found it provided better results than previous methods.

Read More: What is Agile Testing?

Continuous Integration

Continuous Integration (CI) can be considered as a pillar of the Agile process. It is developing software iteratively with small parts of code being integrated into the main code body repeatedly following all SDLC phases. The general approach is described below:

- Requirements are broken down into small tasks and assigned to team members
- The developer works on code and tests the logic of the feature on his system via **Unit testing**
- The code is committed regularly to a repository empowered with a Version Control System of some kind
- This kicks off a series of tests that can be either automated or manual. These tests aim to check the product’s health whenever new code is added via Integration Testing.
- The code is certified as ready for deployment once all tests pass

The above cycle continues until the desired product has been built. This also allows for easy feature enhancements after the initial release.
Every company modifies and adds steps to the above process based on their needs. Most companies work with three environments wherein the code is deployed and tested.

1. It begins with the Development Environment (DEV), a playground for developers to test their code.
2. After local unit testing, code is deployed onto the DEV, and a combined unit test suite is run.
3. The code is then deployed to the **UAT/Integration environment**. The frequency of this deployment varies based on the project. In this environment, code is tested through integration to check for compatibility of newly developed features against existing code. This environment is also called a Sanity environment.
4. After integration tests are passed, the code is tagged and marked as deployment-ready.
5. Continuous Integration implements an iterative and incremental approach to software development. It reduces the feature development cycle time while ensuring the product is stable.
6. CI is so widely used these days that CI and Agile are often considered synonymous, which is incorrect. But it signifies the effectiveness with which CI helps implement the Agile methodology.

Implementation of a CI pipeline for Agile Teams

As discussed earlier, CI is the process of building and testing code repeatedly to get it deploy-ready. Below is a sample CI process that many organizations follow:

The image part with relationship ID rId108 was not found in the file.

1. The developers develop the code and push it onto a central source code management system
2. This code is built and deployed onto the DEV
3. Tests are triggered to check if the code works as expected
4. Regularly – nightly or weekly or fortnightly – this code is freshly constructed to include the latest code commit to the existing repository.
5. This build artifact is deployed onto the UAT or the Sanity environment, wherein the code is tested, as a whole, for its functionality.
6. Integration and regression test suites are run against the deployed code. Post that, the code is declared deploy-ready

These steps can be configured using multiple CI orchestration tools such as Jenkins, Bamboo, Teamcity, etc. Let’s consider an example of how this configuration can be performed using Jenkins for Agile Teams.

Step 1: A Jenkins job can be created to build the code. The source code repo needs to be added in the SCM section as shown below:

The image part with relationship ID rId111 was not found in the file.

Step 2: Additional build environment configurations can be done as required. Under the build section of the configuration, the build command can be issued. This can be done by using plugins or by executing shell commands directly. The image below uses Gradle to build the project:

The image part with relationship ID rId112 was not found in the file.

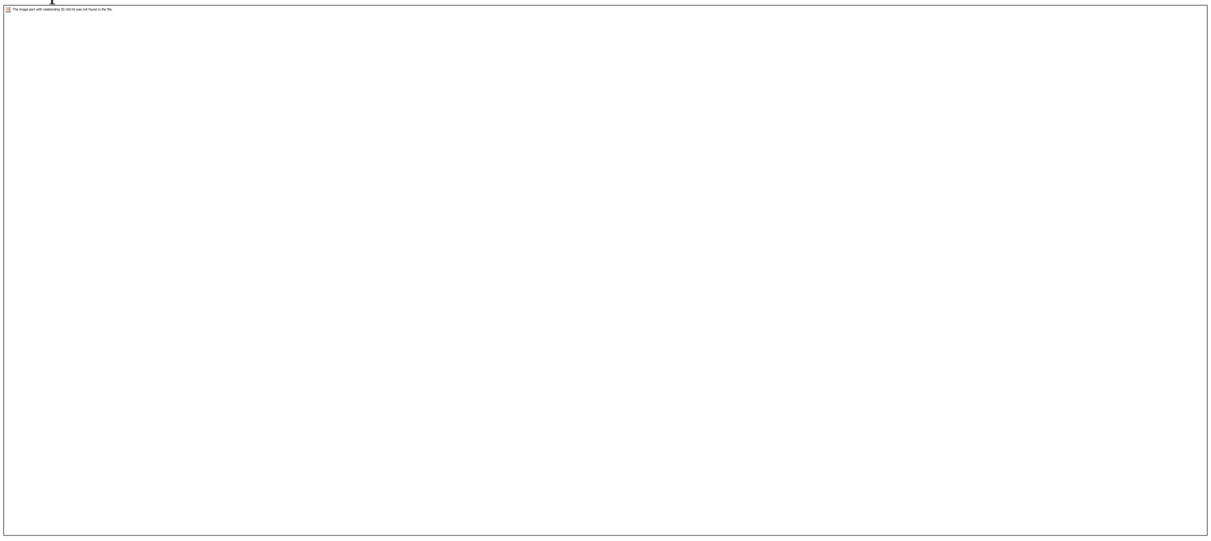
Step 3: Configure the post-build section. Mails can be sent to notify the status of the job. Other builds can be triggered. Reports can be published. Jenkins offers numerous plugins for integrations with multiple tools. The image below depicts how to send a mail and also trigger another job once the current job has succeeded

The image part with relationship ID rId114 was not found in the file.

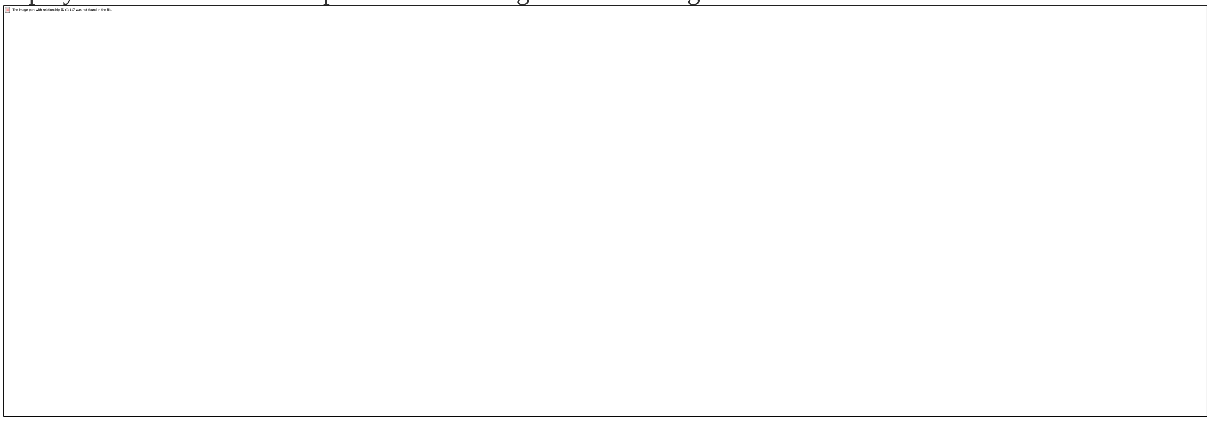
Step 4: Similarly configure the deploy job. Tools like Ansible, Chef, and Puppet are generally used by organizations to handle deployments to multiple servers. The image below shows how one can configure the build part of the job configuration to run an Ansible playbook:

The image part with relationship ID rId115 was not found in the file.

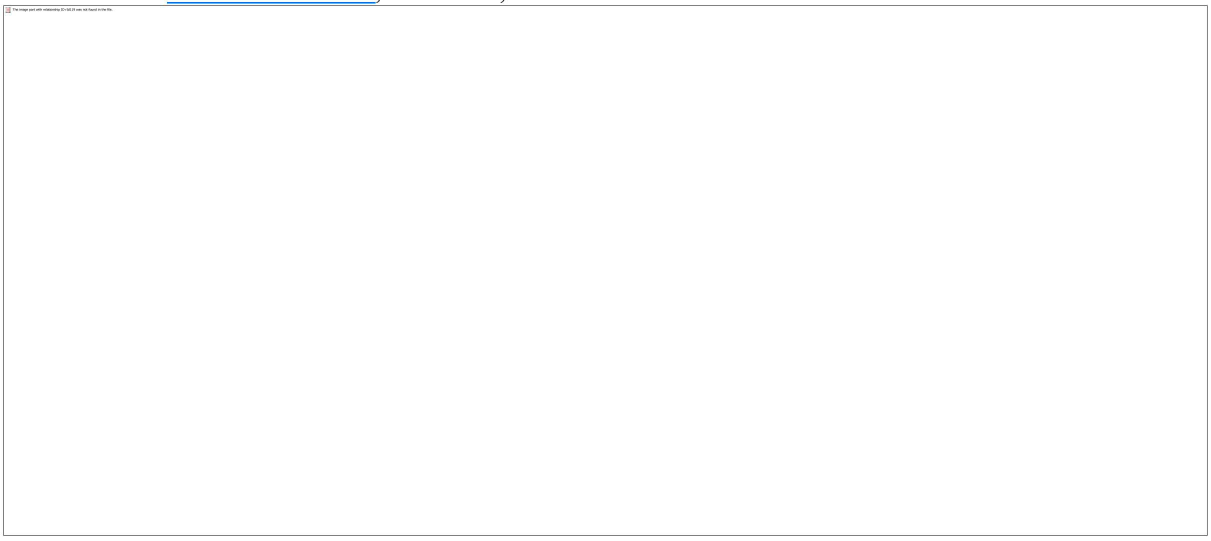
Step 5: Like the build job, the deploy job can now trigger another job (Test Job is this case) upon its successful completion



Step 6: Once the code Is deployed onto the environment, a test job can be run to execute test cases against the deployed code. Another example of Gradle being used for testing:



Step 7: Once tests are run, reports are generated. Multiple plugins are available in Jenkins to publish reports from tools like TestNG framework, Cucumber, etc.



Unit5:

Agile Testing is a type of software testing that follows the principles of agile software development to test the software application. All members of the project team along with the special experts and testers are involved in agile testing. Agile testing is not a separate phase and it is carried out with all the development phases i.e. requirements, design and coding, and test case generation. Agile testing takes place simultaneously throughout the Development Life Cycle. Agile testers participate in the entire development life cycle along with development team members and the testers help in building the software according to the customer requirements and with better design and thus code becomes possible. The agile testing team works as a single team towards the single objective of achieving quality. Agile Testing has shorter time frames called iterations or loops. This methodology is also called the delivery-driven approach because it provides a better prediction on the workable products in less duration time.

- Agile testing is an informal process that is specified as a dynamic type of testing.
- It is performed regularly throughout every iteration of the Software Development Lifecycle (SDLC).
- Customer satisfaction is the primary concern for agile test engineers at some stage in the agile testing process.

Features of Agile Testing

Some of the key features of agile software testing are:

- **Simplistic approach:** In agile testing, testers perform only the necessary tests but at the same time do not leave behind any essential tests. This approach delivers a product that is simple and provides value.
- **Continuous improvement:** In agile testing, agile testers depend mainly on feedback and self-learning for improvement and they perform their activities efficiently continuously.
- **Self-organized:** Agile testers are highly efficient and tend to solve problems by bringing teams together to resolve them.
- **Testers enjoy work:** In agile testing, testers enjoy their work and thus will be able to deliver a product with the greatest value to the consumer.
- **Encourage Constant communication:** In agile testing, efficient communication channels are set up with all the stakeholders of the project to reduce errors and miscommunications.
- **Constant feedback:** Agile testers need to constantly provide feedback to the developers if necessary.

Agile Testing Principles

- **Shortening feedback iteration:** In Agile Testing, the testing team gets to know the product development and its quality for each and every iteration. Thus continuous feedback minimizes the feedback response time and the fixing cost is also reduced.
- **Testing is performed alongside** Agile testing is not a different phase. It is performed alongside the development phase. It ensures that the features implemented during that iteration are actually done. Testing is not kept pending for a later phase.
- **Involvement of all members:** Agile testing involves each and every member of the development team and the testing team. It includes various developers and experts.
- **Documentation is weightless:** In place of global test documentation, agile testers use reusable checklists to suggest tests and focus on the essence of the test rather than the incidental details. Lightweight documentation tools are used.
- **Clean code:** The defects that are detected are fixed within the same iteration. This ensures clean code at any stage of development.
- **Constant response:** Agile testing helps to deliver responses or feedback on an ongoing basis. Thus, the product can meet the business needs.
- **Customer satisfaction:** In agile testing, customers are exposed to the product throughout the development process. Throughout the development process, the customer can modify the requirements, and update the requirements and the tests can also be changed as per the changed requirements.
- **Test-driven:** In agile testing, the testing needs to be conducted alongside the development process to shorten the development time. But testing is implemented after the implementation or when the software is developed in the traditional process.

Agile Testing Methodologies

Some of the agile testing methodologies are:

1. **Test-Driven Development (TDD):** TDD is the software development process relying on creating unit test cases before developing the actual code of the software. It is an iterative approach that combines 3 operations, programming, creation of unit tests, and refactoring.

2. **Behavior Driven Development (BDD):** BDD is agile software testing that aims to document and develop the application around the user behavior a user expects to experience when interacting with the application. It encourages collaboration among the developer, quality experts, and customer representatives.
3. **Exploratory Testing:** In exploratory testing, the tester has the freedom to explore the code and create effective and efficient software. It helps to discover the unknown risks and explore each aspect of the software functionality.
4. **Acceptance Test-Driven Development (ATDD):** ATDD is a collaborative process where customer representatives, developers, and testers come together to discuss the requirements, and potential pitfalls and thus reduce the chance of errors before coding begins.
5. **Extreme Programming (XP):** Extreme programming is a customer-oriented methodology that helps to deliver a good quality product that meets customer expectations and requirements.
6. **Session-Based Testing:** It is a structured and time-based approach that involves the progress of exploratory testing in multiple sessions. This involves uninterrupted testing sessions that are time-boxed with a duration varying from 45 to 90 minutes. During the session, the tester creates a document called a charter document that includes various information about their testing.
7. **Dynamic Software Development Method (DSDM):** DSDM is an agile project delivery framework that provides a framework for building and maintaining systems. It can be used by users, developers, and testers.
8. **Crystal Methodologies:** This methodology focuses on people and their interactions when working on the project instead of processes and tools. The suitability of the crystal method depends on three dimensions, team size, criticality, and priority of the project.

Agile Testing Strategies

1. Iteration 0

It is the first stage of the testing process and the initial setup is performed in this stage. The testing environment is set in this iteration.

- This stage involves executing the preliminary setup tasks such as finding people for testing, preparing the usability testing lab, preparing resources, etc.
- The business case for the project, boundary situations, and project scope are verified.
- Important requirements and use cases are summarized.
- Initial project and cost valuation are planned.
- Risks are identified.
- Outline one or more candidate designs for the project.

2. Construction Iteration

It is the second phase of the testing process. It is the major phase of the testing and most of the work is performed in this phase. It is a set of iterations to build an increment of the solution. This process is divided into two types of testing:

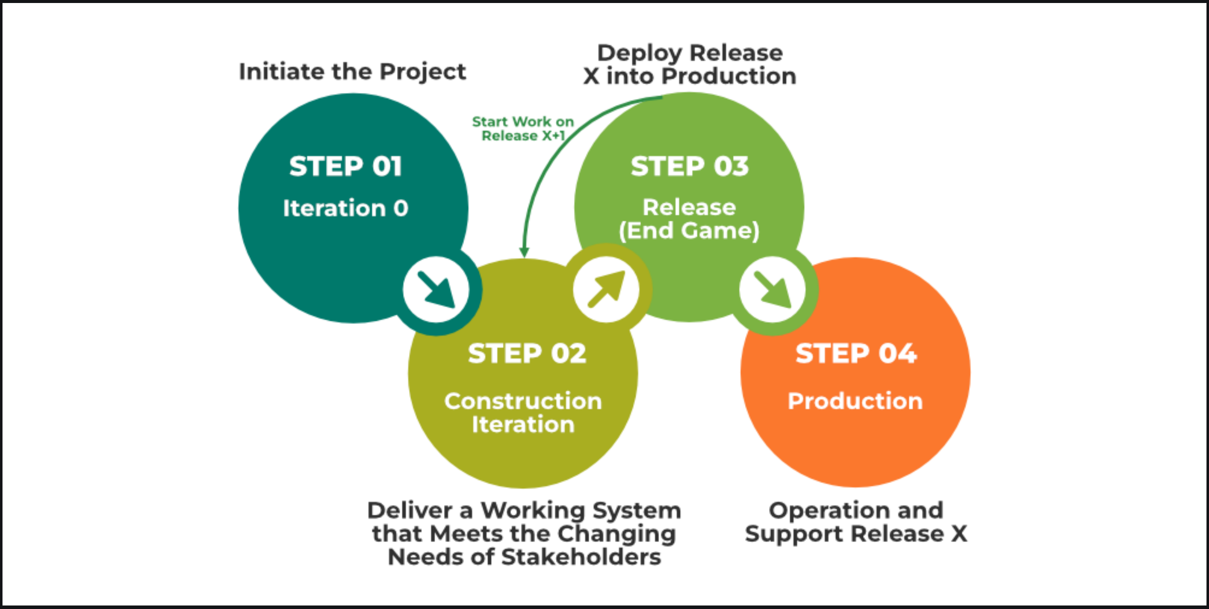
- **Confirmatory testing:** This type of testing concentrates on verifying that the system meets the stakeholder's requirements as described to the team to date and is performed by the team. It is further divided into 2 types of testing:
  - **Agile acceptance testing:** It is the combination of acceptance testing and functional testing. It can be executed by the development team and the stakeholders.
  - **Developer testing:** It is the combination of unit testing and integration testing and verifies both the application code and database schema.
- **Investigative testing:** Investigative testing detects the problems that are skipped or ignored during confirmatory testing. In this type of testing, the tester determines the potential problems in the form of defect stories. It focuses on issues like integration testing, load testing, security testing, and stress testing.

3. Release End Game

This phase is also known as the transition phase. This phase includes the full system testing and the acceptance testing. To finish the testing stage, the product is tested more relentlessly while it is in construction iterations. In this phase, testers work on the defect stories. This phase involves activities like:

- Training end-users.
- Support people and operational people.
- Marketing of the product release.
- Back-up and restoration.
- Finalization of the system and user documentation.

4. Production
It is the last phase of agile testing. The product is finalized in this stage after the removal of all defects and issues raised.



Initiate the Project
Deploy Release X into Production
Start Work on Release X+1
STEP 01
Iteration 0
STEP 03
Release (End Game)
STEP 02
Construction Iteration
STEP 04
Production
Deliver a Working System that Meets the Changing Needs of Stakeholders
Operation and Support Release X

Agile Testing Quadrants
The whole agile testing process is divided into four quadrants:

1. Quadrant 1 (Automated)
The first agile quadrat focuses on the internal quality of code which contains the test cases and test components that are executed by the test engineers. All test cases are technology-driven and used for automation testing. All through the agile first quadrant of testing, the following testing can be executed:

- Unit testing.
- Component testing.

2. Quadrant 2 (Manual and Automated)
The second agile quadrant focuses on the customer requirements that are provided to the testing team before and throughout the testing process. The test cases in this quadrant are business-driven and are used for manual and automated functional testing. The following testing will be executed in this quadrant:

- Pair testing.
- Testing scenarios and workflow.
- Testing user stories and experiences like prototypes.

3. Quadrant 3 (Manual)

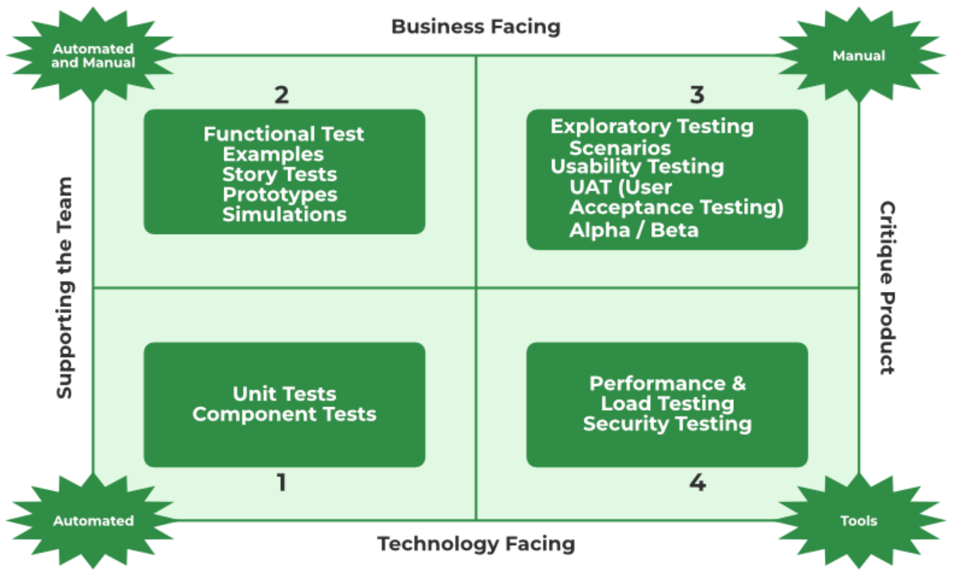
The third agile quadrant provides feedback to the first and the second quadrant. This quadrant involves executing many iterations of testing, these reviews and responses are then used to strengthen the code. The test cases in this quadrant are developed to implement automation testing. The testing that can be carried out in this quadrant are:

- Usability testing.
- Collaborative testing.
- User acceptance testing.
- Collaborative testing.
- Pair testing with customers.

4. Quadrant 4 (Tools)

The fourth agile quadrant focuses on the non-functional requirements of the product like performance, security, stability, etc. Various types of testing are performed in this quadrant to deliver non-functional qualities and the expected value. The testing activities that can be performed in this quadrant are:

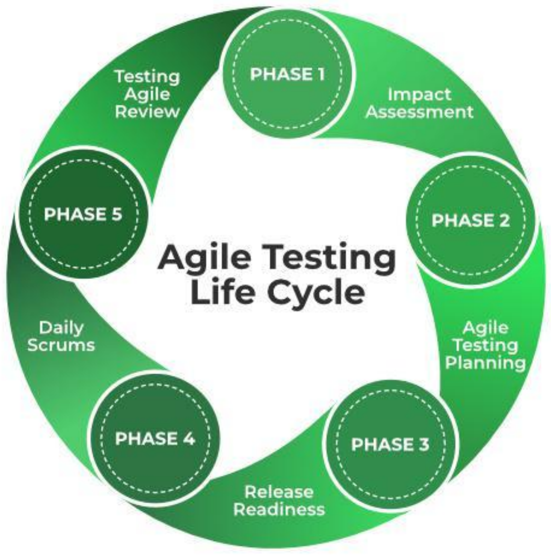
- Non-functional testing such as stress testing, load testing, performance testing, etc.
- Security testing.
- Scalability testing.
- Infrastructure testing.
- Data migration testing.


Business Facing
Automated and Manual
Manual
2
Functional Test Examples Story Tests Prototypes Simulations
3
Exploratory Testing Scenarios Usability Testing UAT (User Acceptance Testing) Alpha / Beta
Supporting the Team
Critique Product
Unit Tests Component Tests
Performance & Load Testing Security Testing
1
4
Automated
Tools
Technology Facing

Agile Testing Life Cycle
The agile testing life cycle has 5 different phases:

1. **Impact Assessment:** This is the first phase of the agile testing life cycle also known as the feedback phase where the inputs and responses are collected from the users and stakeholders. This phase supports the test engineers to set the objective for the next phase in the cycle.
2. **Agile Testing Planning:** In this phase, the developers, customers, test engineers, and stakeholders team up to plan the testing process schedules, regular meetings, and deliverables.
3. **Release Readiness:** This is the third phase in the agile testing lifecycle where the test engineers review the features which have been created entirely and test if the features are ready to go live or not and the features that need to be sent again to the previous development phase.

4. **Daily Scrums:** This phase involves the daily morning meetings to check on testing and determine the objectives for the day. The goals are set daily to enable test engineers to understand the status of testing.
5. **Test Agility Review:** This is the last phase of the agile testing lifecycle that includes weekly meetings with the stakeholders to evaluate and assess the progress against the goals.



Agile Test Plan

An agile test plan includes types of testing done in that iteration like test data requirements, test environments, and test results. In agile testing, a test plan is written and updated for every release. The test plan includes the following:

- Test Scope.
- Testing instruments.
- Data and settings are to be used for the test.
- Approaches and strategies used to test.
- Skills required to test.
- New functionalities are being tested.
- Levels or Types of testing based on the complexity of the features.
- Resourcing.
- Deliverables and Milestones.
- Infrastructure Consideration.
- Load or Performance Testing.
- Mitigation or Risks Plan.

Benefits of Agile Testing

Below are some of the benefits of agile testing:

- **Saves time:** Implementing agile testing helps to make cost estimates more transparent and thus helps to save time and money.
- **Reduces documentation:** It requires less documentation to execute agile testing.
- **Enhances software productivity:** Agile testing helps to reduce errors, improve product quality, and enhance software productivity.
- **Higher efficiency:** In agile software testing the work is divided into small parts thus developer can focus more easily and complete one part first and then move on to the next part. This approach helps to identify minor inconsistencies and higher efficiency.
- **Improve product quality:** In agile testing, regular feedback is obtained from the user and other stakeholders, which helps to enhance the software product quality.

Limitations of Agile Testing

Below are some of the limitations of agile software testing:

- **Project failure:** In agile testing, if one or more members leave the job then there are chances for the project failure.
- **Limited documentation:** In agile testing, there is no or less documentation which makes it difficult to predict the expected results as there are explicit conditions and requirements.
- **Introduce new bugs:** In agile software testing, bug fixes, modifications, and releases happen repeatedly which may sometimes result in the introduction of new bugs in the system.
- **Poor planning:** In agile testing, the team is not exactly aware of the end result from day one, so it becomes challenging to predict factors like cost, time, and resources required at the beginning of the project.
- **No finite end:** Agile testing requires minimal planning at the beginning so it becomes easy to get sidetracked while delivering the new product. There is no finite end and there is no clear vision of what the final product will look like.

Challenges During Agile Testing

Below are some of the challenges that are faced during agile testing:

- **Changing requirements:** Sometimes during product development changes in the requirements or the specifications occur but when they occur near the end of the sprint, the changes are moved to the next sprint and thus become the overhead for developers and testers.
- **Inadequate test coverage:** In agile testing, testers sometimes miss critical test cases because of the continuously changing requirements and continuous integration. This problem can be solved by keeping track of test coverage by analyzing the agile test metrics.
- **Tester's availability:** Sometimes the testers don't have adequate skills to perform API and Integration testing, which results in missing important test cases. One solution to this problem is to provide training for the testers so that they can carry out essential tests effectively.
- **Less Documentation:** In agile testing, there is less or no documentation which makes the task of the QA team more tedious.
- **Performance Bottlenecks:** Sometimes developer builds products without understanding the end-user requirements and following only the specification requirements, resulting in performance issues in the product. Using load testing tools performance bottlenecks can be identified and fixed.
- **Early detection of defects:** In agile testing, defects are detected at the production stage or at the testing stage, which makes it very difficult to fix them.
- **Skipping essential tests:** In agile testing, sometimes agile testers due to time constraints and the complexity of the test cases put some of the non-functional tests on hold. This may cause some bugs later that may be difficult to fix.

Risks During Agile Testing

- **Automated UI slow to execute:** Automated UI gives confidence in the testing but they are slow to execute and expensive to build.
- **Use a mix of testing types:** To achieve the expected quality of the product, a mixture of testing types and levels must be used.
- **Poor Automation test plan:** Sometimes automation tests plan is poorly organized and unplanned to save time which results in a test failure.
- **Lack of expertise:** Automated testing sometimes is not the only solution that should be used, it can sometimes lack the expertise to deliver effective solutions.
- **Unreliable tests:** Fixing failing tests and resolving issues of brittle tests should be the top priority to avoid false positives.

Introduction

Scaling Agile for large organizations is a complex process that requires careful planning and execution. Scaling Agile involves adapting agile methodologies to the needs of larger organizations with more complex structures, processes, and stakeholders.

Why Is Scaling Agile Important for Large Organizations?

Organizations constantly seek ways to improve their agility and adaptability to meet changing customer needs as the world becomes more volatile, uncertain, complex, and ambiguous.

Scaling Agile enables large organizations to respond more quickly to changing customer needs, market trends, and business requirements. Agile methodologies focus on delivering value to customers in small increments, allowing organizations to continuously test and adapt their products or services.

Agile promotes collaboration and communication across teams, departments, and stakeholders. Large organizations often face silos and communication barriers, leading to misunderstandings, delays, and rework. Scaling Agile emphasizes the importance of organizing around value delivery, implementing focused cross-functional teams, regular synchronization, and feedback loops, which can help break down silos and improve communication.

The top reasons large organizations choose to implement Agile

- **Speed to market:** This refers to the time it takes to deliver a product, service, feature, or idea. It is essential to minimize time to market to stay competitive and meet customer demands.
- **Delivery predictability:** A predictable delivery cadence enables the business to make informed decisions; everyone in the organization is aligned and can plan accordingly.
- **Market responsiveness:** Enabling organizations to pivot quickly to respond to ever-changing market demands is essential in today's fast-paced business environment. Adapting to new trends and customer needs is crucial for survival.
- **Innovation:** New ideas, creative thoughts, or novel imaginations provide better solutions to meet new requirements, unarticulated needs, or known market needs. Fostering a culture of innovation can help businesses stay ahead of the curve and differentiate themselves from competitors.
- **Continuous improvement:** The ability of the organization to relentlessly pursue optimizations in all aspects of business functions is critical for long-term success; it involves constantly analyzing and improving processes, products, and services.
- **Productivity:** Increasing the business value realized while maintaining or reducing costs is a crucial objective for any organization. Improving productivity can help achieve this goal by optimizing resources and streamlining operations.
- **Employee engagement:** Employees who are more satisfied in their work are more willing to go the extra mile, passionate about the purpose of their jobs, and committed to the organization. Employee engagement is critical in driving productivity and improving overall business performance.
- **Customer satisfaction**: Customers are satisfied with your products and service's experience, benefits, and outcomes. Ensuring organizations meet and exceed customer needs and expectations is crucial for building brand loyalty and driving revenue.
- **Quality:** The product or service meets the market's expectations for usability and reliability. Maintaining high-quality standards is essential for building a positive reputation and ensuring customer satisfaction. Maintaining high quality also supports multiple other objectives, including speed to market, productivity, and delivery predictability.

How to Scale Agile for Large Organizations?

Step 1: Identify your business objectives

Agile is not an end but a means to an end. Before choosing to implement Agile, you need to know why you are doing it so that you can focus your efforts in the right direction.

Step 2: Secure leadership support and buy-in

Reasons for gaining leadership support and buy-in

- **Resources allocation:** Leadership support is needed to allocate the necessary resources for Agile implementation, such as training, tools, and time for Agile teams to work.
- **Organizational alignment:** Implementing Agile in large organizations requires processes, procedures, and role changes. Leadership support must ensure these changes align with the organization's goals and vision.
- **Change management:** Agile implementation involves a significant shift in mindset and culture. Leaders must support this change and ensure the organization is ready to embrace Agile. Leaders also

need to lead the change by shifting their mindset and enabling a culture of change by exhibiting the behaviors of an Agile leader.

- **Priority setting:** Leadership support is necessary to prioritize the implementation of Agile for the organization. Without leadership support, you may experience resistance or delays.
- **Team empowerment:** Agile is a team-based methodology that empowers teams to make decisions and take ownership of their work. Leadership support is necessary to create an environment that fosters team empowerment and autonomy.
- **Stakeholder communication:** Agile implementation involves stakeholders from different parts of the organization. Leadership support is necessary to facilitate stakeholder communication and ensure everyone is aligned and informed.
- **Scaling:** Implementing Agile in large organizations requires scaling Agile practices across different teams and departments. Leadership support is necessary to ensure that scaling is done effectively and efficiently.

How to get leadership support and buy-in

- **Educate leadership:** Educate leaders about Agile methodology and how it can benefit the organization. Provide case studies and success stories of other organizations successfully implementing Agile.
- **Demonstrate value:** Show how Agile can help the organization achieve its business goals and objectives. Explain how Scaling Agile can improve productivity, reduce costs, and increase customer satisfaction.
- **Start small:** Implement Agile on a small scale and demonstrate its success. Start with a pilot project and show how Agile can deliver value quickly.
- **Involve leadership in the process:** Involve leadership in the Agile implementation process. Assign an executive sponsor to the project and involve them in regular status updates and decision-making.
- **Communicate effectively:** Communicate regularly with leadership about the progress of the Agile implementation. Be transparent about the challenges and successes of the implementation process.
- **Provide training:** Provide Agile training for leadership and team members to ensure everyone understands the methodology and can effectively work within an Agile framework.
- **Adapt to the organization's culture**: Customize Agile to fit the organization's culture and needs. When implementing Agile, consider the organization's size, structure, and existing processes.
- **Address concerns:** Address any concerns that leadership may have about Agile. Be prepared to answer questions about how Agile will impact existing processes and how it will integrate with other business functions.

Step 3: Use an external Agile coach

Getting an Agile coach to support a pilot implementation of Agile in a large organization can provide several benefits:

- **Expertise and Experience:** An Agile coach can provide guidance on best practices, identify potential challenges, and help teams to overcome them.
- **Accelerated Learning:** An Agile coach can help teams learn and adopt Agile practices more quickly. They can provide training and coaching to team members, allowing them to understand and embrace Agile principles.
- **Reduced Risk:** An Agile coach can help reduce the risk of failure by identifying potential issues and providing recommendations for mitigation. They can also help teams to adjust their approach based on feedback and metrics.
- **Better Communication:** An Agile coach can facilitate better communication between team members and stakeholders. They can help teams understand stakeholder needs and ensure everyone is aligned.
- **Improved Team Dynamics:** An Agile coach can help improve team dynamics by fostering collaboration, encouraging feedback, and promoting a culture of continuous improvement.
- **Scalability:** An Agile coach can help ensure that the pilot implementation of Agile is scalable. They can guide Agile practices and help teams adjust their approach.
- **Objective Perspective:** An Agile coach can provide an objective perspective on the pilot implementation of Agile. They can identify areas for improvement and provide recommendations for addressing them.

Hiring an Agile coach with the proper experience and credentials is essential to support an Agile pilot for a large organization. They can provide expertise, accelerate learning, reduce risk, improve communication and team dynamics, ensure scalability, and provide an objective perspective.

Step 4: Start small

Implementing Agile in a large organization can be a challenging task. Starting small can help ease the transition and ensure a successful Agile implementation.

Why Start Small?

- **Reduced Risk:** Starting small minimizes the risk of failure. By implementing Agile on a small scale, you can identify and address any issues before scaling up.
- **Quick Wins:** Implementing Agile on a small scale allows for quick wins, which can build momentum and support for Agile across the organization.
- **Learning and Improvement:** Starting small allows for learning and improvement. You can assess what worked well and what didn't and make adjustments before scaling up.
- **Better Stakeholder Management:** Implementing Agile on a small scale enables better stakeholder management. It allows for greater visibility and feedback from stakeholders.

How to Start Small?

- **Select a Pilot Implementation:** Select a small project to pilot Agile. Ensure that the project is simple and has clearly defined goals.
- **Build a Cross-Functional Team:** Build a cross-functional team with members from different departments, ensuring that the team can deliver the product or service without any external assistance; this will enable better collaboration and communication, reduce dependencies, support faster time to market, and more predictable outcomes.
- **Define Roles and Responsibilities:** Define roles and responsibilities for the team members. Ensure that everyone understands their role and is aligned with the project goals.
- **Provide Agile Training:** Provide Agile training to the team members to ensure everyone understands the Agile methodology and can effectively work within an Agile framework.
- **Establish Agile Ceremonies:** Agile ceremonies such as daily stand-up meetings, sprint planning meetings, and retrospectives form the cornerstones of a successful pilot. Ensure that everyone understands the purpose of these ceremonies and is committed to participating.
- **Monitor Progress:** Monitor the progress of the pilot project. Use metrics such as velocity and burndown charts to track progress and identify any issues.
- Communicate Results: Communicate the results of the pilot project to the organization. Share the successes and challenges and provide recommendations for scaling Agile.

Starting small can be an effective way to implement Agile in a large organization. It reduces the risk of failure, allows for quick wins, enables learning and improvement, and facilitates better stakeholder management.

Step 5: Involve the team in defining the experiment.

Agile works under the assumption that “people closer to the problem understand it best.” You must involve the cross-functional team in defining the Agile experiment. They will provide valuable insights into the problem you are trying to solve. Also, involving them in the experiment definition ensures that you have their full buy-in and support for the process – they believe the experiment you want to run will have the desired results.

Step 6: Measure your results

When initiating your pilot,

- **Set a clear benefit hypothesis statement.** What exactly will you do, and what results do you want to achieve?

- **Choose your leading indicators:** Leading indicators give you early signals about whether you are on track to achieve your benefit hypothesis. It allows you to make adjustments early in the process if necessary.
- **Choose your lagging indicators:** Lagging indicators give you an empirical measure of success – allowing you to prove that you have achieved the specific objectives you set for yourself.
- **Decide how long to run the experiment:** this enables you to set a definite timeline for stopping and reflecting on your experiment. There is no perfect duration, but typically it should run up to two months.

Step 7: Implement Agile iteratively

Many professional service organizations, including large consulting firms, like to sell the idea that you should take a “big bang” approach to implement Agile. The truth is that this serves their purposes and not yours. They want to maximize their short-term revenue to meet their short-term financial goals; this typically results in organizational resistance and failed implementations.

A more low-risk approach, with a much higher probability of success, is to continue rolling Agile out iteratively across teams, using the success of each experiment to drive the impetus for the next. It also enables you to customize Agile to your organization’s specific needs.

Thank you