Agile Methodology: -

What Is Agile Methodology in Project Management?

The Agile methodology is a way to manage a project by breaking it up into **several phases.** It involves constant collaboration with **stakeholders** and continuous improvement at every stage. Once the work begins, teams cycle through a process of **planning**, **executing**, and **evaluating**. Continuous collaboration is vital, both with team members and project stakeholders.

Agile is a time-bound, **iterative** approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver all at once.

Agile: is a **framework** that defines how software development needs to be carried on. Agile is not a single method, it represents the various collection of methods and practices that follow the value statements provided in the **manifesto**. Agile methods and practices do not promise to solve every problem present in the software industry (No Software model ever can). But they sure help to establish a culture and environment where solutions emerge.

Why Agile?

Technology in this current era is progressing faster than ever, enforcing the global software companies to work in a fast-paced changing environment. Because these businesses are operating in an **ever-changing environment**, it is impossible to gather a complete and exhaustive set of software requirements. Without these requirements, it becomes practically hard for any conventional software model to work. The conventional software models such as **Waterfall Model** that depends on completely specifying the requirements, designing, and testing the system are not geared towards rapid software development. As a consequence, a conventional software development model fails to deliver the required product.

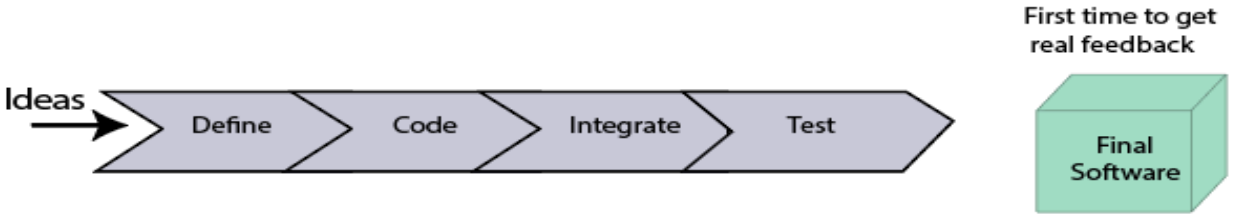
This is where agile software development comes to the rescue. It was specially designed to curate the needs of the rapidly changing environment by embracing the idea of incremental development and developing the actual final product.

What is Agile Methodology?

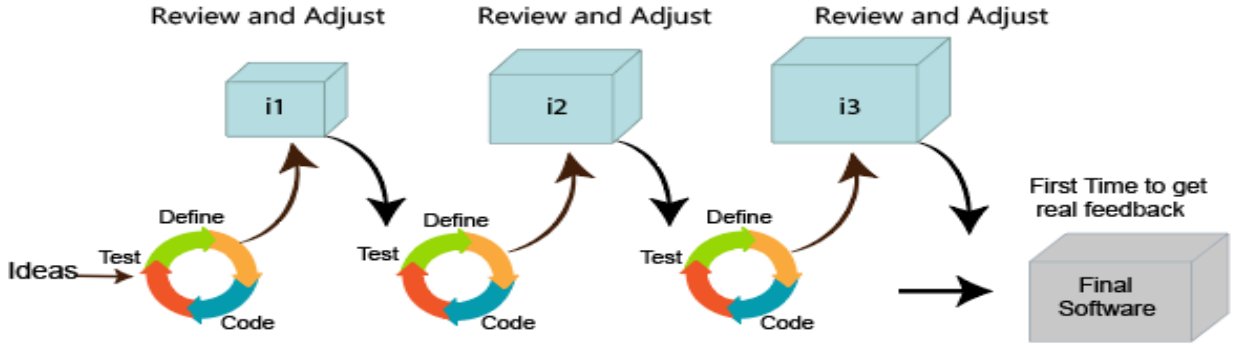
An agile methodology is an iterative approach to software development. Each **iteration of agile methodology** takes a short time interval of 1 to 4 weeks. The agile development process is aligned to deliver the **changing business requirement**. It distributes the software with faster and fewer changes.

The **single-phase software development** takes 6 to 18 months. In single-phase development, all the requirement gathering and risks management factors are predicted initially.

The agile software development process frequently takes the **feedback of workable product**. The workable product is delivered within 1 to 4 weeks of iteration.



Traditional Method



Agile Method

Understanding the Agile values:

The Agile was founded on **four key values**, which are as follows:

1. Individuals and interactions over processes and tools:

The first value gets straight to the core of Agile methodology**:** the focus is on the people. You can use the best processes and tools available to support your projects, but they will only work, if your people are doing their best work. Your team is your most valuable resource. Communication plays a key role here — when people interact with each other regularly and share their ideas, they build better products.

2. Working software over comprehensive documentation

Before Agile practices were fully implemented, teams would spend hours creating **exhaustive** documents with technical **specifications**, **requirements**, and more. These lists would be prepared before developers started to write the

code, meaning the whole process was delayed as the documentation took so long to compile.
The **Agile philosophy** is to **streamline** these documents and **condense** the information into **user stories.** These stories **equip** the developer with all the details they need to start working on the software and get it ready for release. The idea is to accelerate the launch process and make product **tweaks** in the early stages, improving the software in future iterations.

3. Customer collaboration over contract negotiation
The third value of the Agile Manifesto highlights the importance of customer collaboration. This is viewed as superior to contract negotiation, which involves outlining product requirements with the customer before commencing the project and then renegotiating these deliverables at a later stage.

When you bring your customers into the development process, you can ask for their opinions regularly and take their suggestions on board. By **delving into** their specific needs while the software is still being built, developers can gain valuable insights to create the ultimate user experience.

4. Responding to change over following a plan
Traditional methodologies advocated for as little change as possible, recognizing that significant alterations could cost time and money. The aim was to create a **comprehensive** plan that followed a structured, linear path and avoided obstacles where possible.

The Agile mentality turns this **rigid method** on its head, arguing that change can benefit the software development process. When you **embrace** change, you open yourself up to new possibilities and ways to improve. Agile teams work in short, iterative cycles, meaning they can react quickly and implement changes on a continuous basis. This ultimately leads to better products.
These four values form the basis of Agile software development, highlighting key priority areas for teams to focus their energies on. The core values are supported by the 12 Agile Manifesto principles.

Agile Manifesto:

What is the Agile Manifesto?

The Agile Manifesto is a document that identifies **four key values and 12 principles** that its authors believe software developers should use to guide their work. Formally called the *Manifesto for Agile Software Development*, it was produced by 17 developers.
The developers called themselves the **Agile Alliance**. They were seeking an alternative to the existing software development processes that they saw as complicated, unresponsive and too focused on documentation requirements.

Furthermore, the developers sought to find a balance between the existing ways of development and the new alternatives. They admitted to accepting modelling and documentation, but only when it had a clear, beneficial use.
The developers also explained that while planning is important, it's necessary to accept that plans change and flexibility is needed for these modifications. Overall, the manifesto focuses on valuing individuals and interactions over processes and tools.

THE TWELVE PRINCIPLE OF AGILE MANIFESTO

1. **Customer Satisfaction**: Manifesto provides high priority to satisfy the costumer's requirements. This is done through early and continuous delivery of valuable software.
2. **Welcome Change**: Making changes during software development is common and inevitable. Every changing requirement should be welcome, even in the late development phase. Agile process works to increase the customers' competitive advantage.
3. **Deliver the Working Software**: Deliver the working software frequently, ranging from a few weeks to a few months with considering the shortest time period.
4. **Collaboration**: Business people (**Scrum Master** and **Project Owner**) and **developer**s must work together during the entire life of a project development phase.
5. **Motivation**: Projects should be build around motivated team members. Provide such environment that supports individual team members and trust them. It makes them feel responsible for getting the job done thoroughly.
6. **Face-to-face Conversation**: Face-to-face conversation between **Scrum Master** and **development team** and between the **Scrum Master** and **customers** for the most efficient and effective method of conveying information to and within a development team.
7. **Measure the Progress as per the Working Software**: The working software is the key and primary measure of the progress.
8. **Maintain Constant Pace**: The aim of agile development is sustainable development. All the businesses and users should be able to maintain a constant pace with the project.
9. **Monitoring**: Pay regular attention to technical excellence and good design to maximize agility.
10. **Simplicity**: Keep things simple and use simple terms to measure the work that is not completed.
11. **Self-organized Teams**: The Agile team should be self-organized. They should not be depending heavily on other teams because the best architectures, requirements, and designs emerge from self-organized teams.

12. **Review the Work Regularly**: The work should be reviewed at regular intervals, so that the team can reflect on how to become more productive and adjust its behaviour accordingly.

What are the Different Types of Agile Methodologies?

Agile refers to the **methods** and **best practices** for organizing projects based on the **values** and **principles** documented in the **Agile Manifesto**. However, there's no one right way to implement Agile and many different types of methodologies from which to choose. Here are some of the most common Agile frameworks.

Kanban
Kanban is a simple, visual means of managing projects that enables teams to see the progress so far and what's coming up next. Kanban projects are primarily managed through a **Kanban board**, which **segments** tasks into three columns: "**To Do**", "**Doing**" and "**Done**".

Scrum
Scrum is similar to Kanban in many ways. Scrum typically uses a Scrum board, similar to a Kanban board, and groups tasks into columns based on progress. Unlike Kanban, Scrum focuses on breaking a project down into **sprints** and only planning and managing one sprint at a time. Scrum also has unique project roles: Scrum master and product owner.

Extreme Programming (XP)
Extreme Programming (XP) was designed for Agile software development projects. It focuses on **continuous development and customer delivery** (CDCD) and uses intervals or sprints, similar to a Scrum methodology.

However, XP also has 12 supporting processes specific to the world of software development:

- Planning game
- Small releases
- Customer acceptance tests
- Simple design
- Pair programming
- Test-driven development
- Refactoring
- Continuous integration
- Collective code ownership
- Coding standards
- Metaphor
- Sustainable pace

Feature-driven development (FDD)

Feature-driven development is another software-specific Agile framework. This methodology involves creating software models every two weeks and requires a development and design plan for every *model feature*. It has more **rigorous** documentation requirements than XP, so it's better for teams with advanced design and planning abilities. FDD breaks projects down into five basic activities:

- Develop an overall model
- Build a feature list
- Plan by feature
- Design by feature
- Build by feature

Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method (DSDM) was born of the need for a common industry framework for rapid software delivery. Rework is to be expected, and any development changes that occur must be reversible. Like Scrum, XP, and FDD, DSDM uses sprints. This framework is based on eight fundamental principles:

- Focus on the business need
- Deliver on time
- Collaborate
- Never compromise quality
- Build incrementally from firm foundations
- Develop iteratively
- Communicate continuously and clearly
- Demonstrate control

Crystal

Crystal is a family of Agile methodologies that includes **Crystal Clear**, **Crystal Yellow**, Crystal Orange, Crystal Red, etc. Each has a unique framework. Your choice depends on several project factors, such as your team size, priorities, and project criticality.

Lean

Lean development is often grouped with Agile, but it's an entirely different methodology that happens to share many of the same values. The main principles of the Lean methodology include:

- Eliminating waste
- Build quality in
- Create knowledge
- Defer commitment
- Deliver fast
- Respect people
- Optimize the whole

What are the Advantages and Disadvantages of Agile?

Agile methodology has earned huge popularity recently. It is very famous for its capacity to provide high-quality software solutions conveniently. It has been a **buzzword** in software development. The methodology Agile process is an iterative & collaborative process that emphasizes the flexibility to adjust to changing conditions, customer feedback, and market trends. However, like any other methodology, Agile also has its **advantages** & **disadvantages**. Here let's discuss the details about the advantages and disadvantages of Agile;

Advantages of Agile:

Flexibility & Adaptability

If you need a flexible & adaptable system for software development, then Agile is for you. This process allows the team to respond to changes in requirements, client feedback & market trends quickly. The principle of continuous progress is the base of Agile methodology. The adaptability of Agile methodology assures that the final product meets the customer's expectations & satisfies their needs.

Increased Collaboration

The agile methodology encourages co-operation between the **development team, stakeholders & customers**. The development team works closely with stakeholders & customers to assure that the software needs meet their expectations. This approach guarantees that the software solution is **user-centric** and focuses on meeting the end users' needs.

Early and Continuous Delivery

The agile methodology highlights the early & continuous delivery of operating software solutions. This process confirms that the development team can provide high-quality software solutions to customers quickly. The early delivery of software solutions lets the team receive early feedback from the customers & make necessary modifications to the solution.

Transparency

The Agile methodology encourages clarity in the software development process. The development team regularly shares with stakeholders & customers to deliver updates on the growth cycle. This approach guarantees that stakeholders & customers are aware of the progress made on the software solution. They can deliver feedback & suggestions to the development team.

Disadvantages of Agile:

Lack of Predictability

This methodology depends heavily on **flexibility** and **adaptability**. It can result in a shortage of **predictability** in the development procedure. The team may struggle to calculate the time needed to finish a task & the final product may not satisfy the initial requirements. The lack of predictability can generate frustration among stakeholders & customers who expect a predictable timeline for the delivery of the software.

Overemphasis on Documentation

The agile methodology highlights working on software solutions over documentation. However, this process can guide to a lack of documentation, which can generate issues in the future. The lack of documentation can create it hard to maintain & update the software solution, particularly if the original development team is no longer available.

Limited Scope

Agile methodology focuses on delivering small increments of the software solution at a time. This approach can be limiting in terms of the scope of the software solution. The development team may struggle to deliver a comprehensive solution that meets all the customer's requirements within the given timeline.

Overreliance on the Development Team

The agile methodology requires a highly skilled and experienced development team to be successful. The team must have a **profound** knowledge of the software development methodology & be able to work collaboratively with stakeholders & customers. However, this reliance on the development unit can be a disadvantage, particularly if the team lacks the required skills or experience.

Agile methodology has many advantages & disadvantages. The flexibility & adaptability of Agile methodology provides that the product team can respond to changes in requirements, customer feedback & demand quickly. The collaborative approach of Agile methodology confirms that the software solution is user-centric & concentrates on meeting the end-user's needs. However, the lack of predictability, fixed scope, overemphasis on documentation, and overreliance on the development team are some of the disadvantages of Agile methodology. To get success in the project, the team must carefully major the advantages & disadvantages of Agile methodology. They also have to determine whether it is the correct path for their projects.

What are Anti-Patterns in Agile?

Anti-patterns in **agile** or **scrum anti-patterns** are (bad) practices that you follow to improve the process. Still, they do the opposite by **hampering** your efforts and slowing your progress towards achieving Agile goals.

It is a **disguised** form of Agile Development practice that poses to be a solution but creates negative consequences that you might realize later. That is why **retrospectives** need to be taken seriously so that the development team can identify potential problems with the existing process (based on past mistakes) and can introduce improvements.

Top three Agile Anti-Patterns in Software Development Process

Here are the top **three agile anti-patterns** that can de-accelerate time to market:

1. Miscommunication:

The first value of the Agile Manifesto recommends preferring individuals and interactions over processes and tools. Yet, organizations fail to implement this very basic rule when creating software and instead are seen working in a somewhat **mushroom** management setting. In fact, the daily **stand-ups** that the **scrum teams** conduct to understand the progress of the teams is an **agile anti-pattern** too.

Solution:

a. Mob Programming

Mob programming is a practice where the entire agile team works on the same thing, at the same time, and on the same system. This is a great practice to follow where the **UX** designers, architects, developers, and testers can communicate and collaborate to understand each other's processes and solve a problem at hand (collectively).

b. Rely on Collaboration Tools

Remote work or no remote work, collaboration tools can undoubtedly lend a hand at filling the communication gap. You can conduct brainstorming sessions and plan around a remote stack that suits your business needs. Ensure you do not invest much in boat anchors (a piece of software or hardware that is not useful) as it will add up to the costs.

c. Get Over Progress Status

Scrum meetings are generally about following up on the progress status, i.e., where are you on your way to completing tasks assigned to you in that particular **sprint**. They are considered **long**, **flow breakers**, and **meaningless** at times. But that should not be the perspective around it! Daily stand-ups should be serving the purpose well.

2. Unclear Requirements and Expanding Scope Creep

The **product owner** is the bridge between the **agile** teams and the **client** and ensures that the requirements are communicated as it is, i.e., the ideas match with the final product. However, often the authenticity of the requirements goes amiss.

Be it the **sprint planning** meetings or the **product backlog meetings**, if the requirements are not clearly discussed — the **MVP** and the corresponding final product suffers. A non-details requirement can be perceived in many ways, and hence its feasibility can vary for different members.

The negative consequences of it lead to a lot of reworks, and thus piling up technical debts and delayed time to market.

Moreover, the expanding **scope creep** or the **feature creep** during the Agile process adds to the confusion, which further leads to requirement mismatch with the deliverables.

Solution:

- Document the requirements. Do not rely on verbal communication of the requirements alone. Take the product backlog meetings seriously. Ensure all the three scrum roles attend these meetings so that no scope of confusion exists.
- Use collaboration tools such as **Click Up** or **Trello** to maintain transparency of tasks to complete in a particular sprint. This way, everyone can stay on the same page while knowing what they need to be doing.
- Communicate with the stakeholders to verify the documented requirements. Their go-ahead matters when finalizing the requirements.

3. Gold Plating or Scope Stretching

Gold plating or scope stretching is the act of extending the workload unnecessarily and working on something, which was not even required in the **first place**.

A lot many times, the **agile** team extends its responsibilities unnecessarily. In the process of overdoing it, they are seen diverting from the initially defined scope, thus leading to inconsistencies and delays with the deliverables — in turn, unhappy clients and customers.

In most cases, neither the product owner nor the scrum master is aware of the increasing (not needed) burden that the team adds to their shoulders. So, how do you avoid gold plating in the agile development process?

Solution:

- Results should be mapped against the underlying requirements at the end of every sprint.
- There needs to be a constant communication channel between the **product owner** and the **Agile Development team** (UX designers + Developers + Agile Testers).
- The development team should be strictly instructed to stick to the documented requirements. Nothing except the aforementioned requirements will be accepted. If they feel otherwise and feel it necessary to work on something out of the scope, they need to discuss it first with the concerned parties.

Scaled Agile Framework:

Scaled Agile Framework is a knowledge base of **proven**, **integrated principles**, **practices**, and **competencies** for achieving business agility using Lean, Agile, and DevOps. It is built around the **Seven Core** Competencies of Business Agility that are critical to achieving and sustaining a competitive advantage in an increasingly digital age:

1. **Lean-Agile Leadership** – Advancing and applying Lean-Agile leadership skills that drive and sustain organizational change by empowering individuals and teams to reach their highest potential
2. **Team and Technical Agility** – Creating high-performing Agile teams as well as sound technical practices including Built-in Quality, implementing and measuring flow, and more
3. **Agile Product Delivery** – Building high-performing teams-of-teams that use design thinking and **customer-centricity** to provide a continuous flow of valuable products using DevOps, the Continuous Delivery Pipeline, and Release on Demand
4. **Enterprise Solution Delivery** – Building and sustaining the world's largest software applications, networks, and cyber-physical solutions
5. **Lean Portfolio Management** – Executing **portfolio vision** and strategy formulation, **chartering portfolios**, creating the Vision, Lean budgets and **Guard rails**, as well as portfolio prioritization, and road mapping
6. **Organizational Agility** – **Aligning** strategy and execution by applying Lean and systems thinking approaches to strategy and investment funding, Agile portfolio operations, and governance
7. **Continuous Learning Culture** – Continually increasing knowledge, competence, and performance by becoming a learning organization committed to relentless improvement and innovation.

Why Lean UX

The increasingly competitive nature of the digital product development space means there's more pressure on designers than ever before — not just to deliver better products than everyone else, but also to deliver them faster than everyone else.

Based on the lean-agile methods of making the design process incremental, **Lean UX** is a design strategy focused around **minimizing wasted time and effort during the design process**.

Three foundations of Lean UX:

1. **Design Thinking:** thinks like a designer, drive the business direction by observing what people want and need, like or dislike and refine accordingly

2. **Agile Software Development**: Lean UX applies the four core principles of Agile development
    1. Individuals and interactions over processes and tools
    2. Working software over comprehensive documentation
    3. Customer collaboration over **contract negotiation**
    4. Responding to change over following a plan

3. **Lean Startup Methods:** build the Minimum Viable Products (MVPs) for rapid learning of market response

The core principles of Lean UX

Lean UX has 5 core principles:

- Act as one team
- Solve the right problem
- Design collaboratively
- Exercise flexibility
- De-emphasize deliverables

Let's take a look at each of these principles in more detail.

Act as one team:

Act as one team

The first key principle of Lean UX is acting **as one team**. This means going against the tradition of delegating tasks based on skill and keeping design teams separate, instead bringing together designers,

product designers, engineers, and developers to work together. Having a diverse team with a multitude of backgrounds and disciplines in the mix means you'll come up with more diverse and well-rounded solutions to the problems brought up by the user testing.

Solve the right problem

The second core principle of Lean UX is solving the right problems over the wrong ones. Lean UX relies heavily on a loop of continuous learning facilitated by continuous feedback. By receiving this feedback, the team will be able to hone in and focus on solving the right problem with a deeper understanding of the issue at hand. By working in small increments, i.e. a weekly feedback session that informs a week of work, you'll be able to focus on finding a solution without any large risk. If your solution isn't working and you need to reframe the problem, this way you will have only wasted a week — rather than, say, 3 or 4 months.

Design collaboratively

The third and perhaps most important principle of Lean UX is collaboration. In the design world, there's a growing step towards 'many hats' job specs and away from limiting yourself with specific and specialized design roles. This increasing demand for designers who can work across disciplines is paving the way towards a collaborative future — a future where Lean UX will thrive. One way to facilitate collaboration between teams is to have a shared whiteboard or painted white wall that allows anyone to add their ideas and thoughts to.

Be flexible

The **penultimate** core principle of Lean UX is flexibility. In traditional UX, you're always working with a high level of uncertainty — but the traditional approach is to try and change that. This is a different story in Lean UX, where you should always assume (no matter how well it's going) that the plan is going to change — and therefore you need to plan accordingly. For example, your second round of user testing might mean that your initial assumptions are proved wrong, and you have to go back to square one. This might go

against some people's instinct, and many may instead be tempted to make some minor changes and continue on as normal. But being flexible is critical to the Lean UX process.

De-emphasize deliverables:

The fifth and final core principle of Lean UX is a de-emphasis on designing for deliverables. Where traditional design favours upfront work, Lean UX focuses on adjusting your design process as you go along in order to become more agile compatible. That's not to say that Lean UX is devoid of deliverables; rather, it asserts that you should work out what conversation needs to be had in order to move forward to the next step. In short, Lean UX places emphasis on conversations rather than deliverables in the design process.