AGILE

INTRODUCTION TO AGILE

Definition:
Agile is a way of developing software in **small steps**, with **continuous feedback**, so changes can be made quickly.

Why Agile is Needed (Purpose)

- Customer requirements change frequently
- Faster delivery is expected
- Early feedback reduces project failure
- Better teamwork and transparency

Key Points (Core Concepts)

- Work is divided into **small iterations** (sprints)
- Customer is involved **throughout development**
- Frequent **testing and feedback**
- Focus on **working software**, not heavy documentation
- Flexible to change

Real-Life Example
Think of **building a house room by room**:
Instead of completing the entire house at once, you build one room, show it to the owner, take feedback, then build the next room.

Technical Example (Software)
In a **shopping app**:

- Sprint 1 → Login & Registration
- Sprint 2 → Product listing
- Sprint 3 → Cart & Payment
  After each sprint, the customer reviews and suggests changes.

Interview Explanation (How to Say It)
"Agile is a software development approach where work is delivered in small iterations with continuous customer feedback, allowing teams to adapt to changes quickly and deliver high-quality software."

Common Interview Questions

- What is Agile methodology?
- Why Agile is better than Waterfall?
- What are Agile principles?
- What is an iteration or sprint?

Common Mistakes Freshers Make

- Thinking Agile is only **Scrum**
- Believing Agile has **no planning**
- Ignoring documentation completely
- Thinking Agile means **no deadlines**

One-Line Summary
Agile delivers software incrementally with flexibility, feedback, and faster value.

AGILE MANIFESTO

Definition:
The **Agile Manifesto** is a set of **4 core values and 12 principles** that guide how Agile teams develop software effectively.

Why Agile Manifesto is Needed (Purpose)

- To focus on **what really matters** in software development
- To reduce delays caused by heavy processes
- To improve customer satisfaction
- To help teams adapt to change easily

Agile Manifesto – 4 Core Values
Agile values **individuals on the left**, but **values the right more**

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

Real-Life Example
Instead of writing a long plan and following it blindly, Agile teams:

- Talk directly to people
- Build something usable
- Show it to the customer
- Improve based on feedback

Technical Example (Software)
In a banking app:

- Developers discuss directly with testers
- Deliver login feature early
- Customer reviews and asks for OTP change
- Team adapts without restarting the project

Interview Explanation (How to Say It)
"The Agile Manifesto defines four core values that emphasize people, working software, customer collaboration, and adaptability over rigid processes."

Common Interview Questions

- What is the Agile Manifesto?
- Name the four Agile values
- Explain 'Responding to change over following a plan'
- Does Agile ignore documentation?

Common Mistakes Freshers Make

- Saying Agile has **no documentation**
- Forgetting the word **'over'** in values
- Thinking Agile means **no planning**
- Mixing values with principles

One-Line Summary
Agile Manifesto focuses on people, working software, collaboration, and flexibility.

SCRUM FRAMEWORK

Definition:
Scrum is an **Agile framework** used to manage and complete complex work by delivering software in **short iterations called sprints.**

Why Scrum is Needed (Purpose)

- To deliver working software **frequently**
- To handle changing requirements easily
- To improve **team collaboration**
- To ensure transparency and continuous improvement

Key Components of Scrum (Big Picture)
Scrum has **3 Roles, 5 Events, and 3 Artifacts.**

Scrum Roles
Scrum Roles (Who does what)

1. **Product Owner (PO)**
    - Represents customer/business
    - Maintains **Product Backlog**
    - Prioritizes requirements
2. **Scrum Master (SM)**
    - Facilitates Scrum process
    - Removes team obstacles
    - Ensures Scrum rules are followed
3. **Development Team**
    - Developers, testers, designers
    - Self-organizing team
    - Delivers working increment

Real-Life Example (Roles)
Think of **restaurant**:

- PO → Customer deciding menu
- SM → Manager ensuring smooth process
- Dev Team → Chefs cooking food

Scrum Events
Scrum Events (Meetings)

1. **Sprint**
    - Fixed time (1–4 weeks)
2. **Sprint Planning**
    - Decide what to build in the sprint
3. **Daily Scrum (Stand-up)**
    - 15-minute daily meeting
4. **Sprint Review**
    - Demo to stakeholders
5. **Sprint Retrospective**
    - Discuss improvements

Real-Life Example (Events)
Each week:

- Plan tasks

- Daily short sync
- Show completed work
- Improve team process

Scrum Artifacts
Scrum Artifacts (Documents)

1. **Product Backlog**
   - List of all requirements
2. **Sprint Backlog**
   - Tasks selected for sprint
3. **Increment**
   - Working software at sprint end

Technical Example (Software)
In an e-commerce app:

- Product Backlog → Login, Cart, Payment
- Sprint Backlog → Login & Registration
- Increment → Working login module

Interview Explanation (How to Say It)
"Scrum is an Agile framework that works in time-boxed sprints and defines clear roles, events, and artifacts to deliver incremental value."

Common Interview Questions

- Is Scrum a methodology or framework?
- What are Scrum roles?
- What happens in Daily Scrum?
- What is an increment?

Common Mistakes Freshers Make

- Saying Scrum is a **methodology**
- Confusing Sprint Review with Retrospective
- Thinking Scrum Master is the team boss
- Assuming sprint duration can change mid-sprint

One-Line Summary
Scrum delivers software in short sprints with clear roles, events, and artifacts.

SPRINT PLANNING

Definition:
Sprint Planning is a Scrum meeting where the team decides **what work will be done** and **how it will be completed** in the upcoming sprint.

Why Sprint Planning is Needed (Purpose)

- To clearly understand **sprint goals**
- To select the right amount of work
- To avoid confusion during the sprint
- To align team and Product Owner expectations

Key Points (Core Concepts)

- Happens **at the start of every sprint**
- Time-boxed (max **8 hours** for a 1-month sprint)

- Entire Scrum team participates
- Focuses on **what** and **how**

Two Main Questions Answered
What is done in Sprint Planning?

1. **What can be delivered?**
   → Product Owner explains top backlog items
2. **How will the work be done?**
   → Development team breaks items into tasks

Real-Life Example
Before a **college project submission**:

- Decide features to complete this week
- Assign tasks among teammates
- Set a clear goal to finish on time

Technical Example (Software)
For a **banking app sprint**:

- Selected stories: Login, Forgot Password
- Tasks: UI, backend API, validation, testing
- Sprint Goal: Secure user authentication

Interview Explanation (How to Say It)
"Sprint Planning is a Scrum event where the team decides what backlog items to work on and how to complete them within the sprint."

Common Interview Questions

- Who attends Sprint Planning?
- What are inputs and outputs of Sprint Planning?
- What is a Sprint Goal?
- How long is Sprint Planning?

Common Mistakes Freshers Make

- Thinking only Scrum Master attends
- Over-committing too much work
- Starting sprint without clear sprint goal
- Confusing Sprint Planning with Daily Scrum

One-Line Summary
Sprint Planning sets the sprint goal and defines work before development starts.

AGILE USER STORIES

Definition:
A **User Story** is a short, simple description of a feature written from the **user's point of view.**

Why User Stories are Needed (Purpose)

- To understand **what the user wants**
- To keep requirements **simple and clear**
- To focus on **business value**
- To enable better planning and estimation

Standard User Story Format
As a *[type of user]*,
I want *[some goal]*,
So that *[some benefit]*

Key Points (Core Concepts)

- Written in **simple language**
- Focuses on **user value**
- Small enough to complete in one sprint
- Can change based on feedback

Real-Life Example
As a student,
I want online class recordings,
So that I can revise later.

Technical Example (Software)
As a customer,
I want to reset my password,
So that I can regain access to my account.

Acceptance Criteria
Acceptance criteria define **when a user story is considered complete.**
Example:

- User can enter registered email
- OTP is sent successfully
- Password is updated securely

INVEST Principle (Quality of User Stories)
Good user stories are:

- **I**ndependent
- **N**egotiable
- **V**aluable
- **E**stimable
- **S**mall
- **T**estable

Interview Explanation (How to Say It)
“User stories are lightweight requirements written from the user perspective to describe what the system should do and why.”

Common Interview Questions

- What is a user story?
- What is the user story format?
- What is acceptance criteria?
- What is INVEST principle?

Common Mistakes Freshers Make

- Writing technical tasks instead of user needs
- Missing “So that” (business value)
- Making stories too large
- Confusing user stories with use cases

One-Line Summary
User stories describe features from the user’s viewpoint to deliver real value.

STORY POINTS

Definition:
Story Points are units used in Agile to **estimate the effort** required to complete a user story.

Why Story Pointing is Needed (Purpose)

- To estimate work **without using time**
- To compare story **complexity and effort**
- To help in **sprint planning**
- To improve predictability over time

What Story Points Measure
Story points consider:

- **Complexity**
- **Effort**
- **Risk & uncertainty**

Not hours or days.

Fibonacci Scale (Commonly Used)
Typical story point values:
1, 2, 3, 5, 8, 13, 21
(Larger gaps reflect higher uncertainty)

Real-Life Example
Cleaning your room → 1 point
Shifting house → 13 points
Because shifting has more effort, risk, and uncertainty.

Technical Example (Software)
User Story: *Reset Password*

- UI changes
- Backend logic
- Email/OTP service
- Testing

→ Estimated as **5 story points**

How Story Pointing is Done

- Team discusses the story
- Uses **Planning Poker**
- Agrees on a point value
- Based on past experience

Velocity (Related Concept)
Velocity = Total story points completed in one sprint
Used to plan future sprints.

Interview Explanation (How to Say It)
"Story points are a relative estimation technique used in Agile to measure effort, complexity, and risk of user stories."

Common Interview Questions

- What are story points?
- Why not estimate in hours?
- What scale is used for story points?
- What is velocity?

Common Mistakes Freshers Make

- Converting story points to hours
- Estimating alone (instead of team discussion)
- Changing points mid-sprint
- Comparing points across different teams

One-Line Summary
Story points help teams estimate effort relatively and plan sprints effectively.

AGILE ESTIMATION AND PLANNING

Definition:
Agile Estimation and Planning is the process of **predicting effort** and **planning work in iterations** so teams can deliver value regularly and adapt to change.

Why Agile Estimation & Planning is Needed (Purpose)

- To decide **how much work** can be done in a sprint
- To avoid over-commitment
- To set realistic expectations with stakeholders
- To track progress and improve predictability

Key Characteristics

- Based on **relative estimation** (not exact time)
- Done **continuously**, not once
- Team-driven, not manager-driven
- Improves with experience

Agile Estimation Techniques
Common Estimation Techniques

1. **Story Points** – effort-based estimation
2. **Planning Poker** – team consensus method
3. **T-Shirt Sizes** – XS, S, M, L, XL
4. **Affinity Estimation** – grouping similar stories
5. **Three-Point Estimation** – Optimistic, Pessimistic, Most likely

---

Real-Life Example
Planning a **wedding**:

- Small tasks (invites) → low estimate
- Big tasks (venue, catering) → high estimate
- Adjust plans as things change

Agile Planning Levels
Levels of Agile Planning

1. **Product Planning** – overall roadmap
2. **Release Planning** – features per release
3. **Sprint Planning** – work for next sprint
4. **Daily Planning** – daily scrum updates

Technical Example (Software)
For an e-commerce app:

- Product roadmap → Login, Catalog, Payment
- Release plan → MVP in 3 months
- Sprint plan → Login + Registration
- Daily plan → Today's tasks in stand-up

Velocity in Planning

- Velocity = Story points completed per sprint
- Helps forecast future work
- Used only for **same team**

Interview Explanation (How to Say It)
"Agile estimation and planning use relative sizing and continuous feedback to plan work incrementally and deliver value predictably."

Common Interview Questions

- What is Agile estimation?
- What techniques are used for estimation?
- What is velocity and how is it used?
- Why Agile planning is continuous?

Common Mistakes Freshers Make

- Treating estimates as commitments
- Converting story points into hours
- Planning everything upfront
- Ignoring past velocity

One-Line Summary
Agile planning is iterative, flexible, and based on relative estimation.

AGILE METRICS and REPORTING

Definition:
Agile metrics and reporting are used to **measure team progress, performance, and predictability** in order to improve delivery, not to judge individuals.

Why Agile Metrics Are Needed (Purpose)

- To track **progress and transparency**
- To identify **bottlenecks early**
- To improve future sprint planning
- To give stakeholders clear visibility

Key Rule of Agile Metrics
Metrics are for improvement, not punishment.

Common Agile Metrics
Velocity

- Total story points completed in a sprint
- Used to plan future sprints

Burndown Chart

- Shows **remaining work vs time**

- Helps track if sprint is on schedule

Burnup Chart
- Shows **work completed vs total scope**
- Highlights scope changes clearly

Cycle Time
- Time taken to complete a work item
- Shorter cycle time = faster delivery

Lead Time
- Time from request to delivery
- Used to measure customer waiting time

Defect Density
- Number of defects per feature/module
- Helps assess product quality

Cumulative Flow Diagram (CFD)
- Shows work in different states (To-Do, In-Progress, Done)
- Helps detect bottlenecks

Real-Life Example
Food delivery app:
- Velocity → orders completed daily
- Cycle time → order preparation time
- Burndown → daily pending orders

Technical Example (Software)
Sprint of 2 weeks:
- Planned: 30 story points
- Completed: 28 story points
- Velocity: 28
- Burndown shows slight delay mid-sprint

Interview Explanation (How to Say It)
“Agile metrics like velocity and burndown charts help teams measure progress, predict delivery, and continuously improve.”

Common Interview Questions
- What is velocity?
- Difference between burnup and burndown?
- What is cycle time vs lead time?
- Why metrics should not be misused?

Common Mistakes Freshers Make
- Using metrics to compare teams
- Focusing only on velocity
- Ignoring quality metrics
- Treating metrics as targets

One-Line Summary
Agile metrics provide transparency and guide continuous improvement.

SDLC vs AGILE

Definition:
SDLC (Software Development Life Cycle)
A **traditional, step-by-step process** for building software where each phase is completed one after another.
Agile
A **flexible, iterative approach** where software is developed in **small increments with continuous feedback.**

Why This Comparison Matters (Purpose)

- Interviewers test your **concept clarity**
- Helps you explain **why companies shifted to Agile**
- Important for **Cognizant training & real projects**

SDLC (Traditional Model)
SDLC Phases

1. Requirement Analysis
2. Design
3. Development
4. Testing
5. Deployment
6. Maintenance

Each phase must finish before the next starts.

SDLC Key Characteristics

- Fixed requirements
- Heavy documentation
- Late testing
- Change is costly
- Customer involved mostly at start

Agile Model
Agile Phases (Iterative)

- Plan → Build → Test → Review → Improve
- Done repeatedly in **sprints**

Agile Key Characteristics

- Changing requirements accepted
- Working software delivered frequently
- Continuous testing
- Regular customer feedback
- Flexible and adaptive

SDLC vs Agile (Quick Comparison)

Aspect	SDLC	Agile
Approach	Sequential	Iterative
Flexibility	Low	High
Customer Involvement	Low	High
Testing	After development	Continuous
Delivery	End of project	Every sprint
Documentation	Heavy	Light
Change Handling	Costly	Easy

Real-Life Example
SDLC:
Planning entire college syllabus at start and not changing it.
Agile:
Studying topic by topic, taking feedback, improving weekly.

Technical Example (Software)
Banking App:

- SDLC → Entire app released after months
- Agile → Login first, then transactions, then analytics

Interview Explanation (How to Say It)
"SDLC is a traditional sequential model, while Agile is an iterative approach that delivers working software frequently and adapts to change."

Common Interview Questions

- Why Agile over SDLC?
- Can Agile be part of SDLC?
- Which is better and why?

Common Mistakes Freshers Make

- Saying Agile is not part of SDLC
- Saying SDLC is outdated
- Forgetting testing phase differences
- Saying Agile has no documentation

One-Line Summary
SDLC follows a fixed plan; Agile adapts and delivers continuously.

SOFTWARE SUPPORT and MAINTENANCE

Definition:
Software support and maintenance is the process of **fixing issues, improving performance, and updating software after it is deployed.**

Why Software Support & Maintenance is Needed (Purpose)

- To fix **bugs and errors**
- To adapt software to **new environments**
- To improve performance and security
- To satisfy user needs continuously

Key Activities in Support & Maintenance

- Bug fixing
- Performance tuning
- Security patches
- Feature enhancements
- User support & troubleshooting

Types of Software Maintenance
Corrective Maintenance

- Fixes **defects and bugs**
- Example: Login error fix

Adaptive Maintenance

- Changes due to **environment updates**
- Example: Browser or OS update compatibility

Perfective Maintenance

- Enhances **features or performance**
- Example: Faster search functionality

Preventive Maintenance

- Prevents future issues
- Example: Code refactoring, cleanup

Real-Life Example

Bike service:

- Fix breakdown → corrective
- New road rules → adaptive
- Better mileage → perfective
- Regular servicing → preventive

Technical Example (Software)

In a production application:

- Bug in payment → corrective
- Server upgrade → adaptive
- UI improvement → perfective
- Code optimization → preventive

Support vs Maintenance

Aspect	Support	Maintenance
Focus	User issues	Software improvement
Nature	Reactive	Proactive
Example	Ticket handling	Code enhancement

Interview Explanation (How to Say It)

"Software support handles user-reported issues, while maintenance improves and updates the system post-deployment."

Common Interview Questions

- What is software maintenance?
- Types of maintenance?
- Difference between support and maintenance?

Common Mistakes Freshers Make

- Mixing support and maintenance
- Forgetting preventive maintenance
- Ignoring security updates
- Thinking development ends after deployment

One-Line Summary

Support fixes issues; maintenance improves software continuously.
