CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

NAAC GRADE A+
Accredited University

ASSIGNMENT

AGILE METHODOLOGY

(24CAT-656)

SUBMITTED BY,

Student Name: Devanand Utkarsh
Branch: MCA
Semester: II
Section/Group: 6 (B)
UID: 24MCA20454

SUBMITTED TO,

Priyanka Ma'am

Q 1: What are different types project management approaches in agile?

Agile project management includes several approaches that help teams deliver projects iteratively and efficiently. Here are the main types:

1. Scrum

- Uses fixed-length iterations called **Sprints** (typically 1–4 weeks).
- Roles: **Product Owner, Scrum Master, Development Team**.
- Key meetings: **Sprint Planning, Daily Standup, Sprint Review, Sprint Retrospective**.
- Best for software development and iterative delivery.
- **Example-**
    - A bank develops a **mobile banking app** using an **Agile approach**, releasing features in **iterations**. The process includes:
    - **Sprint Planning:** Define features and select tasks for a **2-week Sprint**.

      Sprint Execution: Developers work on tasks with **daily standups** to track progress.
      Sprint Review: Completed features are **demonstrated** and feedback is collected.

      Sprint Retrospective: The team **reviews** successes and areas for improvement.

      Next Sprints: More features are developed **iteratively** until the app is complete.

        - **Outcome:** The app is continuously tested, improved, and released in stages.

2. Kanban

- Visual workflow management using a **Kanban board** (To Do, In Progress, Done).
- Focuses on **limiting work in progress (WIP)** to avoid bottlenecks.
- Best for continuous delivery and teams with unpredictable workloads.
- **Example –**

- A **software company's** customer support team uses **Kanban** to manage bug fixes and feature requests. They create a **Kanban board** with columns like **To Do, In**

- **Progress, Review, and Done**, set a **Work-in-Progress (WIP) limit**, and prioritize urgent tickets. Fixes are **released continuously** for faster issue resolution.
-
  - **Outcome: Faster** issue resolution and a **smooth workflow** without delays.

3. Lean

- Aims to **reduce waste** and improve efficiency.
- Encourages **continuous improvement (Kaizen)**.
- Best for manufacturing, startups, and service industries.
- **Example –**
  - A **startup** builds an online store using **Lean methodology** by first launching a **Minimum Viable Product (MVP)** with essential features (product catalog, cart, checkout). They **collect customer feedback**, then iteratively **add only necessary features** like reviews and payment integration while **avoiding waste**.
  - **Outcome:** A **quick launch** with **minimal costs**, focusing on customer needs.

4. Extreme Programming (XP)

- Focuses on **high-quality code and customer collaboration**.
- Practices: **Pair Programming, Test-Driven Development (TDD), Continuous Integration**.
- Best for fast-changing software projects.
- **Example –**
  - A tech company builds a real-time chat app using Extreme Programming (XP) techniques like Pair Programming (two developers coding together), Test-Driven Development (TDD) (writing test cases before coding), and Continuous Integration (frequent testing and merging). New features are released weekly.
  - ◇ Outcome: Faster development with fewer bugs due to continuous testing and feedback.

Q 2: Differentiate between CI and CD?

Here are some differences between CI and CD,

Continuous Integration (CI)	Continuous Deployment (CD)
Automates code integration and testing.	Automates software release and deployment.
Automates software release and deployment.	Ensures software is always ready for release (Delivery) or deployed automatically (Deployment).
Automates build and testing processes.	Automates testing and deployment steps.
Requires manual deployment after testing.	Delivery requires manual approval; Deployment is fully automated.
Risks are Low, as it focuses on code integration.	Risk are Higher in Deployment since changes go live automatically.

Q 3: Define Kanban methodology with the help of example?

Kanban Methodology with Example

Definition:

Kanban is an **Agile project management method** that helps teams **visualize tasks**, **limit work in progress (WIP)**, and **improve efficiency** through continuous workflow.

Key Principles of Kanban:

1 . Visualizing Work:

1. Tasks are displayed on a **Kanban board** with columns representing different stages (e.g., **To Do, In Progress, Review, Done**).

2. Limiting Work in Progress (WIP):

- A **task limit** is set for each stage to avoid overload and maintain focus.

3. Continuous Flow:

- Work moves smoothly between stages, preventing bottlenecks.

4 **Continuous Improvement:**

- The team regularly analyzes and optimizes the workflow for better productivity.

Example: Customer Support System

A **software company's** customer support team uses **Kanban** to manage bug fixes and feature requests.

◇ **Implementation:**

☑ **Kanban Board Setup:** Columns are created:

- **To Do** – Pending support tickets
- **In Progress** – Tickets being worked on
- **Review** – Tickets under testing
- **Done** – Resolved tickets

☑ **WIP Limit:** A maximum of **3 tasks per developer** is allowed in the **In Progress** column to avoid overload.

☑ **Prioritization:** Urgent tickets (e.g., **payment issues**) are handled first, while minor UI fixes are scheduled later.

☑ **Continuous Delivery:** Fixes and small updates are **released immediately**, instead of waiting for a full software update.

- **Outcome:**
  - **Faster issue resolution**
  - **Improved workflow efficiency**
  - **Better task management**
    Kanban ensures **smooth progress**, prevents delays, and helps the team **deliver quality support** efficiently.

Q 4: How do we define an item to “done”?

In **Agile**, an item (task, feature, or user story) is considered "Done" when it meets all the **agreed-upon criteria** set by the team. These criteria ensure that the work is complete, high-quality, and ready for use without requiring further changes.

Common Criteria for "Done":

1. **Code is Written and Merged:**
   - The feature or task is **fully developed** and added to the **main codebase**.
2. **Code is Reviewed:**
   - Another team member has **reviewed the code** for potential errors, adherence to standards, and quality.
3. **Testing is Complete:**
   - **Unit tests**, **integration tests**, and **user acceptance tests** have all passed successfully.
4. **No Known Bugs:**
   - All **critical issues** discovered during testing have been **resolved**.
5. **Deployed to the Right Environment:**
   - The feature is deployed and accessible in the **staging** or **production** environment, ready for users.

6. **Meets Requirements:**
    - The task fulfills all the **conditions** set by the **Product Owner** or **client**.
7. **Documented:**
    - Any **necessary documentation** or **user instructions** have been updated to reflect the new changes or features.

Example:

A **development team** is working on a new **login feature**. It is only marked as **"Done"** when:

- The **code** is **written, reviewed**, and **merged** into the main branch.
- **Automated and manual tests** have **passed**.
- The **feature** functions as expected, with **no errors**.
- The feature is **deployed** to the staging or production environment, **ready for use** by the users.

Q 5: Why retrospectic approach is crucial in Agile?

A **retrospective** is a meeting held at the end of each **sprint** or **project phase** where the team reflects on their work: what went well, what didn't, and how to improve. This process is vital in **Agile** for ensuring **continuous improvement** and fostering **team collaboration**.

Importance of Retrospectives in Agile:

1. **Identifying Successes and Challenges:**
    - Helps teams understand **what worked well** and **what needs improvement**.
2. **Improving Workflow Efficiency:**

- Teams find ways to enhance their **workflow**, making future sprints more **efficient** and productive.

3. **Encouraging Open Communication:**

- Provides a platform for **team members** to share feedback, resolve issues, and address concerns in an open manner.

4. l**earning from Past Sprints:**

- Lessons learned help to **avoid similar problems** in future sprints, leading to **better outcomes**.

5. **Fostering Inclusion and Involvement:**

- Everyone gets a chance to **express their thoughts**, making team members feel **valued** and involved in the process.

Example:

A **development team** faces delays due to **unclear requirements**. During the retrospective, the team realizes they need better communication with the **Product Owner**. In the next sprint, they implement **regular check-ins**, which reduces confusion and **speeds up delivery**.