alok.giri@ncit.edu.np

What are PM Tools?

Definition: PM tools are digital platforms that help teams plan, execute, monitor, and close projects efficiently.

Purpose: Enhance productivity, visibility, and team

collaboration.

Core Capabilities:

- Task assignment and tracking
- Deadline and milestone planning
- Real-time updates
- Reporting and analytics

Agile Overview

Agile = **iterative**, **incremental**, and **collaborative** approach.

Key practices:

- Sprints

- Daily stand-ups
- Backlogs
- Continuous feedback

Why PM tools? Agile requires structure, visibility, and flexibility—all offered by modern PM tools.

Role of PM Tools in Agile

Sprint Planning: Organize work into defined sprints.
Backlog Management: Prioritize and refine user stories.
Visual Boards: Represent work as tasks/stories visually (Kanban, Scrum boards).

Continuous Feedback Loops: Enable fast tracking of blockers, comments, and changes.

Burndown Charts: Show work completed vs. remaining.

Importance of PM Tools in Collaborative, Distributed, and Iterative Teams

Collaboration in Agile Teams

- Agile thrives on **team interaction**.
- PM tools:
  - Share common boards and dashboards.
  - Enable commenting and real-time updates.
  - Encourage shared ownership of work.

Distributed & Remote Teams

- Increasingly common in global workplaces.
- PM tools support:
    - Async updates across time zones.
    - Cloud-based access and mobile compatibility.
        - Integration with video conferencing and chat (e.g., Zoom, Slack).

Key Features of PM Tools

Core Functionalities

- **Task Creation and Assignment**:
    - Assign users, set due dates, set priorities.
- **Task Hierarchy**:
    - Stories > Tasks > Subtasks (JIRA)

- **Checklists**: Mark off granular steps within tasks.

Time and Sprint Management

- **Sprint Boards**: View current workload.
- **Burndown Charts**: Visualize how much work remains. •

Time Tracking: Log hours, monitor time spent per user/task.

Team Collaboration

- **Mentions**: Alert team members via @mention.
- **Comments & Threads**: Maintain context on task-specific discussions.
- **Activity Logs**: Who did what, when.

Integration Capabilities

- **DevOps**:

    - GitHub/GitLab: commit linking, PR tracking
    - Jenkins: deployment status
- **Comms & Docs**:
    - Slack, Teams: real-time alerts
        - Google Drive, Dropbox: attach files, collaborate in real time

Dashboards and Reports

- Monitor:
    - Sprint velocity
    - Workload by assignee
    - Cycle time, lead time
- Build:
    - Custom dashboards for team leads
    - Automated email reports

Best Practices in Using PM Tools

Align Tool with Your Agile Process

- Match workflows to methodology:
  - Scrum? Use Sprint planning + Review + Retrospective templates.
  - Kanban? Focus on WIP limits and cycle times.
- Avoid complexity—only use what's needed.

Maintain Clean Boards

- Daily updates are essential.
- Groom backlogs regularly.
- Define **Done**: criteria should be clear and enforced.

Best Practices in Using PM Tools

Use Communication Features Strategically

- Don’t overuse notifications—use @mentions meaningfully.
- Encourage in-tool discussions rather than email. ● Keep decisions recorded in comments or attachments.

Monitor and Reflect

- Analyze reports after every sprint:
    - What slowed us down?
    - Were goals met?
- Feed insights into retrospectives and planning.

Collaboration Tools: Enhancing Team Communication

What Are Collaboration Tools?

- Digital platforms that support communication, file sharing, task coordination, and decision-making.
- Used across teams to:
  - Improve responsiveness
  - Reduce miscommunication
  - Document and track decisions

Categories of Collaboration Tools

Category	Example Tools	Use Case
Messaging/Chat	Slack, MS Teams	Quick discussions, updates
Video Conferencing	Zoom, Google Meet	Meetings, demos, interviews
File Sharing	Google Drive, Dropbox	Document collaboration, stora
Project Management	Trello, JIRA, Asana	Task tracking, sprints
Documentation	Confluence, Notion, Docs	Knowledge base, policies
Whiteboarding	Miro, Mural	Brainstorming, visual ideation

Modes of Communication

Type	Examples	Pros	Cons
Synchronous	Chat, Calls, Zoom	Real-time, instant clarification	Requires overlapping availability
Asynchronous	Email, Docs, Comments	Flexibility, thoughtful replies	Slower feedback loops

When to Chat, Call or Document

- **Chat:** Quick updates, informal, non-blocking
- **Call/Video:** Urgent or sensitive topics, team alignment
- **Document:** Long-term reference, processes, decisions

Communication in Agile Teams

Agile Communication Needs

- Short feedback cycles (e.g., daily standups)
- Transparent backlog and sprint progress ●

Real-time updates from all team members ●

Continuous improvement via retrospectives

Key Principles for Agile Collaboration

- Visibility: Everyone sees progress & blockers ●

Alignment: Shared goals, clear responsibilities •

Feedback: Open, frequent, actionable

Common Communication Challenges

Remote Work & Distributed Teams

- Reduced face-to-face cues
- Harder to build trust and rapport
- Need for intentional communication rituals

Time Zones & Availability

- Misaligned working hours → Delays
- Strategies:
    - Use asynchronous updates
    - Shared calendars with working hours
    - Plan meetings inclusively

Cross-Functional Teams

- Different terminologies, priorities
- Need for:
    - Shared documentation

- Common toolset
- Clear communication protocols

Best Practices for Team Communication

- Define preferred channels for each type of message
- Encourage documentation and transparency
- Balance between synchronous & async
- Regularly review and adapt communication norms

Tool Selection Tips

- Match tools to team size and workflow
- Integrate tools for seamless use
- Prioritize usability & adoption

What is Continuous Integration (CI)?

Definition:

Continuous Integration is the practice of **merging all developers' code changes into a shared mainline** several times a day. Each merge is verified by an automated build and tests.

Key Concepts:

- Frequent commits (daily or more)
- Automated testing and build verification
- Catch issues early in development

Benefits:

- Early bug detection
- Faster feedback
- Reduced integration issues

What is Continuous Deployment (CD)?

Definition:

Continuous Deployment is the practice of **automatically deploying every change that passes all stages of the CI pipeline to production**.

Variants:

- **Continuous Delivery:** Automated staging, manual production deployment
- **Continuous Deployment:** Fully automated, including production

Benefits:

- Shorter release cycles
- Increased deployment frequency
- Faster customer feedback

Importance of CI/CD in Agile

Agile emphasizes **rapid, iterative development**

CI/CD supports **quick feedback loops**

Enhances **collaboration between dev, QA, and ops**

Ensures working software is **always available**

Overview of CI/CD Pipeline Tools

Categories:

- **Version Control Systems**
- **CI Servers / Automation Tools**
- **Build Tools**

- **Testing Tools**
- **Deployment Tools**
- **Monitoring and Notifications**

Version Control Systems

Git:

Type: Distributed Version Control System (DVCS)

Features:

- Every developer has a local copy of the full repository. ● Enables branching, merging, rollback, and tracking of code history.
- Fast and efficient collaboration.

Use in CI/CD: Commits and pushes to Git trigger pipeline runs.

GitHub:

Type: Cloud-hosted Git repository
Features:

- Pull requests for code review
- GitHub Actions for CI/CD
- Issue tracking, wikis, and projects

CI/CD Relevance: GitHub Actions offers native automation for builds, tests, and deployments.

GitLab

- **Type:** DevOps platform with Git hosting + built-in CI/CD
- **Features:**
    - Git repository + CI/CD + issue tracking + code review in one
    - YAML-based pipeline configuration
- **CI/CD Relevance:** GitLab CI is integrated; triggers jobs on push, merge, etc.

CI Servers/Automation Tools

Jenkins

- **Type:** Open-source CI server
- **Features:**
    - Plugin-based architecture
    - Automates build, test, deploy tasks
    - Highly customizable
- **Use Case:** Flexible for large projects with custom workflows.

GitHub Actions

- **Type:** CI/CD automation directly in GitHub
- **Features:**
    - Triggers on events (push, pull request, etc.)
    - YAML-based workflows
    - Reusable actions from the GitHub Marketplace ●

Advantage: Easy setup for teams already using GitHub.

GitLab CI

- **Type:** Integrated CI/CD in GitLab
- **Features:**
  - `.gitlab-ci.yml` to define stages/jobs
  - Runs jobs in Docker containers
- **Advantage:** Unified experience from repo to deployment.

CircleCI

- **Type:** Cloud-native CI/CD platform
- **Features:**
  - Fast build speeds with parallelism
  - Supports Docker, macOS, Linux environments ●

Ideal For: Teams needing fast feedback and scaling.

Build Tools

Maven

- **Type:** Build automation tool for Java
- **Features:**
  - Uses `pom.xml` for configuration
  - Manages dependencies and builds
- **CI/CD Use:** Automatically compiles Java code, runs tests, and creates artifacts.

Gradle

- **Type:** Modern build tool for Java, Android
- **Features:**
  - Faster than Maven (uses a build cache)
  - Uses Groovy or Kotlin for scripting

- **CI/CD Use:** Efficient builds, especially in Android projects.

Npm (Node Package Manager)

Type: Package manager for JavaScript

Features:

- Manages libraries/dependencies
- Defines build/test scripts in `package.json`

CI/CD Use: Installs packages, builds front-end apps, runs tests.

Testing Tools

JUnit

- **Type:** Java unit testing framework
- **Features:**
    - Tests individual classes/methods

    - Supports annotations, assertions
- **Use in CI/CD:** Run in CI to catch regressions early.

Selenium

- **Type:** Web browser automation tool
- **Features:**
    - Simulates user interactions in browsers
    - Supports multiple languages
- **Use in CI/CD:** Validates front-end functionality in real browsers.

Cypress

- **Type:** Modern end-to-end testing tool for JavaScript
- **Features:**
    - Fast execution
    - Runs inside the browser
    - Real-time UI during test runs
- **Use in CI/CD:** Ideal for React, Angular, Vue apps.

Postman

- **Type:** API testing tool
- **Features:**
  - Create, send, and validate HTTP requests
  - Automate tests using Newman CLI
- **Use in CI/CD:** Validate API endpoints after every code change.

Deployment Tools

Docker

- **Type:** Containerization platform
- **Features:**
  - Packages app + dependencies into containers
  - Ensures consistency across environments
- **CI/CD Role:** Build Docker images → push to registry → deploy to staging/prod.

Kubernetes

- **Type:** Container orchestration system
- **Features:**
    - Manages clusters of Docker containers
    - Handles scaling, rolling updates, recovery
- **CI/CD Role:** Automate deployment of Docker containers to a scalable cluster.

Monitoring and Notification

Slack

- **Type:** Team communication tool
- **Features:**
    - CI/CD notifications via webhooks or integrations
    - Real-time alerts for build failures, deployments
- **CI/CD Use:** Notifies teams instantly on failures or successes.

Email Alerts

- **Type:** Traditional communication channel
- **Features:**
    - Sends build/test results to developer inbox
    - Configurable in most CI tools (e.g., Jenkins, GitHub Actions) ● **CI/CD Use:** Keeps stakeholders informed even if they aren't using Slack.

Code Commit: Developer pushes code to Git repo (e.g., GitHub)

Trigger Build: CI server (e.g., Jenkins) detects the commit **Run**

Tests: Unit, integration, UI, API tests executed **Build Artifacts:**

App is compiled and packaged

Deploy to Staging: App is deployed in a safe test environment

Approval (optional): Manual gate in Continuous Delivery

Deploy to Production: Automated (CD) or Manual **Monitor &**

Notify: Alerts sent to Slack/Email