Software Project Management

Tuesday, December 09, 2025 3:42 PM

Unit 06: Applications of Software Project Management in Industry

Nov – Dec 2022

Q7) a) What is visibility in Devops? What are the different ways to enable the visibility in Azure Devops?
Ans.

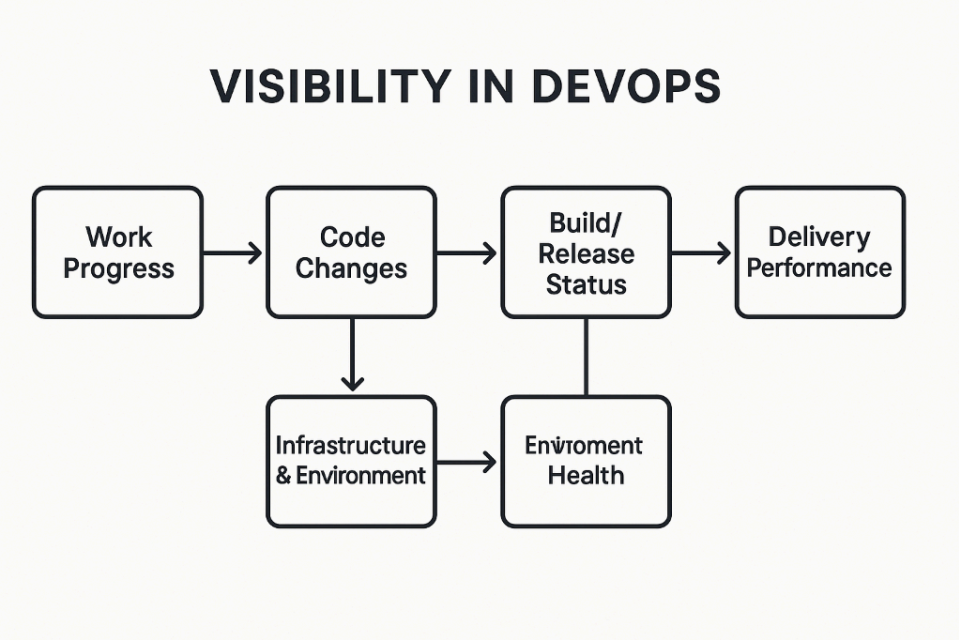
What Is Visibility in DevOps?

Visibility in DevOps means having clear, real-time insight into every stage of software delivery — from planning and coding to deployment and operations.

It ensures teams can see:

- Work progress
- Code changes
- Build and release status
- Infrastructure and environment health
- Delivery performance (KPIs, metrics)

Goal: Improve collaboration, reduce bottlenecks, speed up problem resolution, and increase overall delivery quality.



Ways to Enable Visibility in Azure DevOps

Azure DevOps provides several built-in tools and practices to improve visibility across the entire DevOps lifecycle.

1. Dashboards

Dashboards allow teams to see high-level project indicators.

What they show:

- Work item progress
- Build/Release success rates
- Test results
- Deployment pipeline status
- Code coverage

How it enhances visibility:

Provides a **single-pane-of-glass** view for team performance and project health.

2. Azure Boards (Work Items, Backlogs, Kanban Boards)

Azure Boards enable visibility into planning and work tracking.

Features:

- Product backlog & sprint backlog
- User stories, bugs, tasks tracking
- Kanban boards with custom swimlanes
- Analytics views

Benefit:

Teams always know **what is being worked on**, **who is responsible**, and **what remains**.

3. Pipelines (CI/CD Visibility)

Azure Pipelines give insight into build, test, and deployment workflows.

Examples of visibility:

- Build duration & history
- Test failures
- Deployment logs
- Approval gates & release progress

Benefit:

Immediate feedback on code quality and deployment status.

4. Repository Visibility (Azure Repos)

Azure Repos provides visibility into code practices.

Features:

- Commit history
- Branch policies
- Pull requests with reviewers
- Code search

Benefit:

Team members can easily track code changes and enforce quality through policies.

5. Test Plans

Azure Test Plans increase visibility into testing activities.

Shows:

- Manual & exploratory test status
- Test runs & failures
- Traceability to requirements and bugs

Benefit:

Helps assess test coverage and release readiness.

6. Analytics & Reporting

Azure DevOps includes built-in analytics plus Power BI integration.

You can track:

- Lead time & cycle time
- Build failure rates
- Deployment frequency
- Work item trends

Benefit:

Data-driven visibility for better decision-making.

7. Audit Logs and Compliance

Audit logs show who changed what and when.

Useful for:

- Compliance
- Security
- Change tracking

Benefit:

Visibility into operational and administrative activities.

8. Notifications & Alerts

Azure DevOps can notify users about:

- Build failures
- PR updates
- Work item changes
- Deployment issues

Benefit:

Real-time visibility into events requiring attention.

9. Integration with Monitoring Tools

Azure DevOps integrates with external monitoring platforms:

- **Azure Monitor**
- **Application Insights**
- **Grafana dashboards**
- **ELK/Datadog/Splunk**

  Benefit:

  End-to-end visibility from code to production performance.

Summary Table

Area	Visibility Feature	Purpose
Planning	Boards, Backlogs, Kanban	Track work & progress
Code	Repos, Pull Requests	Track changes & enforce quality
Build/Release	CI/CD Pipeline Logs	Monitor automation and deployments
Testing	Test Plans	Ensure test coverage & quality
Monitoring	Azure Monitor, App Insights	Observe runtime behavior
Reporting	Dashboards, Power BI	Analytics & decision support
Governance	Audit logs, Policies	Compliance & security visibility

Q7) b) Define Application Lifecycle Management (ALM) tools? What feature should be considered while choosing an ALM Tools? List some examples of ALM tools?
Ans.

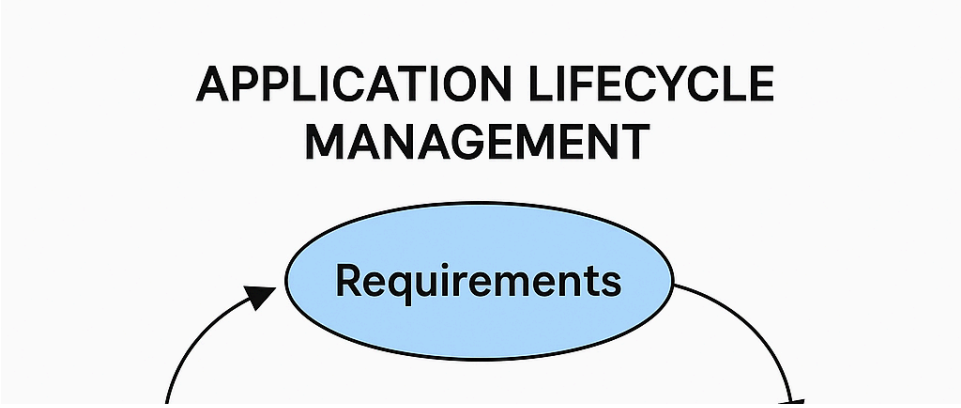
What Are Application Lifecycle Management (ALM) Tools?

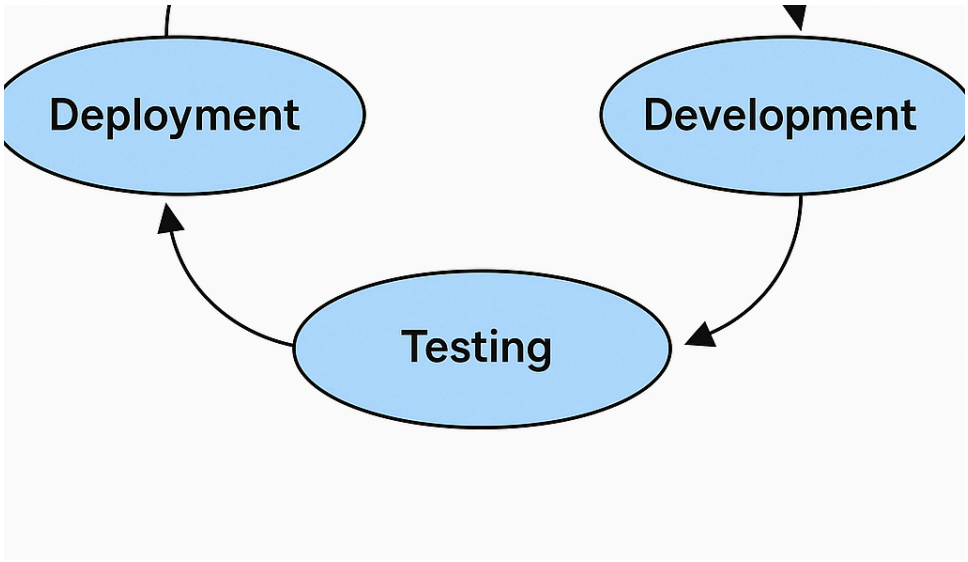
Application Lifecycle Management (ALM) tools are software platforms that help manage the entire lifecycle of an application—from planning and requirements gathering through development, testing, deployment, maintenance, and retirement.

They integrate people, processes, and tools to ensure:

- Better collaboration
- Increased transparency
- Traceability across the lifecycle
- Improved quality and faster delivery

In short: ALM tools provide an end-to-end framework to manage and track all aspects of software development and delivery.





Features to Consider When Choosing an ALM Tool

When selecting an ALM tool, the following features are important:

1. Requirements Management

- Ability to capture, track, and maintain requirements
- Traceability between requirements → code → tests → releases

2. Project / Work Management

- Support for Agile, Scrum, Kanban, or Waterfall
- Backlogs, sprints, task boards, dashboards

3. Version Control Integration

- Native or integrated Git, TFVC, SVN, etc.
- Branching, merging, pull requests, policies

4. CI/CD Pipeline Support

- Ability to automate builds, tests, and deployments
- Integration with Jenkins, Azure Pipelines, GitHub Actions, etc.

5. Test Management

- Manual and automated test management
- Test planning, coverage reports, defect tracking

6. Reporting and Analytics

- Real-time dashboards
- Metrics like velocity, cycle time, defect density

7. Collaboration Features

- Wiki, documentation
- Notifications, discussions, comments, approvals

8. Integration Capabilities

- Connectors for DevOps, cloud providers, IDEs, automation tools
- APIs and plugins

9. Security and Access Control

- Role-based permissions
- Audit logs
- Compliance support

10. Scalability & Cloud Support

- Ability to handle large projects and teams
- SaaS or on-premise options

11. Cost and Licensing

- Pricing model (per user, per project, enterprise)
- Support & maintenance costs

Examples of Popular ALM Tools

Here are widely used ALM tools in the industry:

Microsoft Azure DevOps

- Work tracking, repos, CI/CD, test plans
- Full ALM + DevOps suite

Atlassian Jira + Confluence + Bitbucket

- Requirements, work tracking, documentation, version control

GitLab

- Integrated Git repo, CI/CD, issue tracking, security scanning

Rally Software (CA Agile Central)

- Enterprise Agile planning and lifecycle management

IBM Engineering Lifecycle Management (formerly Rational)

- Requirements management, testing, workflow automation

Micro Focus ALM / ALM Octane

- Strong test management + requirements tracking

ServiceNow DevOps & Agile

- Enterprise workflow & change management

VersionOne (Digital.ai Agility)

- Agile/SAFe lifecycle management

Polarion ALM

- Traceability and requirements management

Summary

Topic	Explanation
ALM Tools	Software that supports management of the full application lifecycle
Features to Consider	Requirements management, version control, CI/CD, testing, reporting, integrations, scalability
Examples	Azure DevOps, Jira, GitLab, Rally, IBM ELM, Micro Focus ALM, VersionOne, Polarion

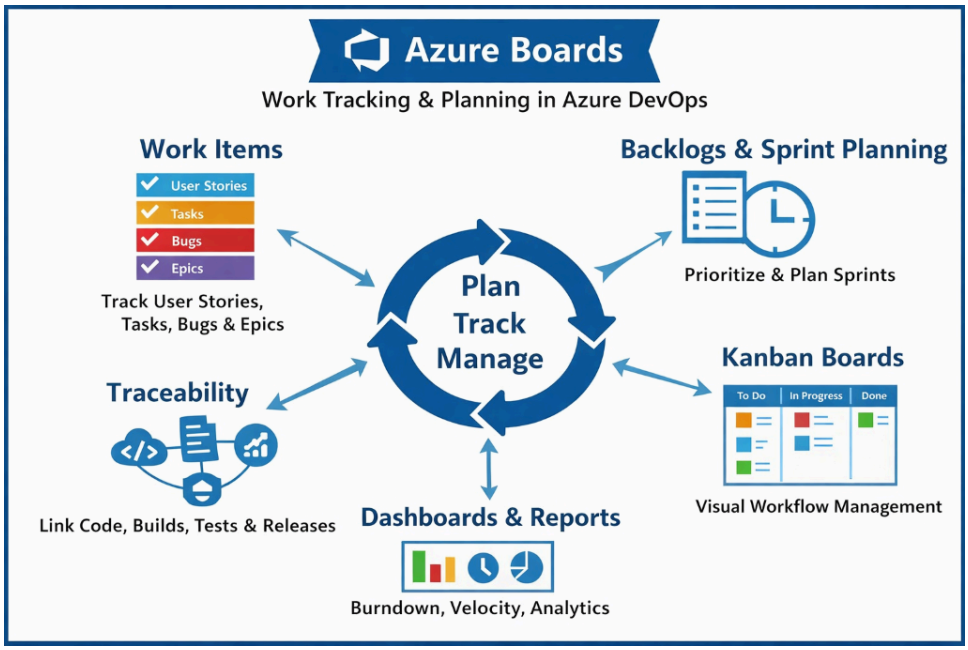
Q7) c) What is Azure Board? Explain with suitable example?
Ans. **Azure Boards** is a work-tracking service in **Azure DevOps** that helps teams plan, track, and manage their work using:

- **Work items** (User Stories, Tasks, Bugs, Epics, Features)
- **Backlogs** (Product backlog, Sprint backlog)
- **Kanban boards**
- **Sprint planning tools**
- **Dashboards and analytics**

Azure Boards is widely used by Agile, Scrum, and Kanban teams.

✓ Purpose of Azure Boards

It provides **visibility**, **collaboration**, and **traceability** across the entire development process.



Key Features of Azure Boards

- **Work Item Tracking** – Create items such as User Stories, Tasks, Bugs, Features.
- **Customizable Kanban Boards** – Drag-and-drop workflow visualization.

- **Backlogs & Sprint Planning** – Prioritize work, estimate, assign tasks.
- **Dashboards & Reporting** – Burndown charts, velocity charts, analytics.
- **Collaboration Tools** – Comments, attachments, linking items, @mentions.
- **End-to-end Traceability** – Link work items to builds, code commits, test cases, releases.

Example to Explain Azure Boards

Scenario

A software team is developing a new feature called **"User Login Functionality"** for a web application.

They use Azure Boards to manage their work.

1. Product Owner creates a User Story

User Story:

As a user, I want to log into the application so that I can securely access my account.

In Azure Boards, this becomes a **Work Item**.

2. The development team breaks the story into tasks

Inside the User Story, the team adds tasks:

Task	Assigned To
Create login UI page	Frontend Developer
Implement authentication API	Backend Developer
Connect API to UI	Developer
Write unit tests	QA
Perform functional testing	QA

Each task shows progress, status, and assignee.

3. Work is tracked on the Kanban Board

The Kanban board might have columns like:

- **To Do**
- **In Progress**
- **Code Review**
- **Testing**
- **Done**

Developers drag tasks across the board as work progresses.

4. Sprint Planning

During sprint planning:

- The User Story is selected for Sprint 1
- Story points are assigned (e.g., 5 points)
- Tasks are given estimates

Azure Boards automatically generates:

- Sprint **burndown chart**
- Team **capacity planning**

5. Traceability

As developers commit code, they link commits to the work item:

```
git commit -m "Implemented authentication API #1234"
```

Work Item **#1234** (User Story) now has traceability to:

- Code changes
- Build results
- Testing results
- Deployment logs

Simple Example Summary

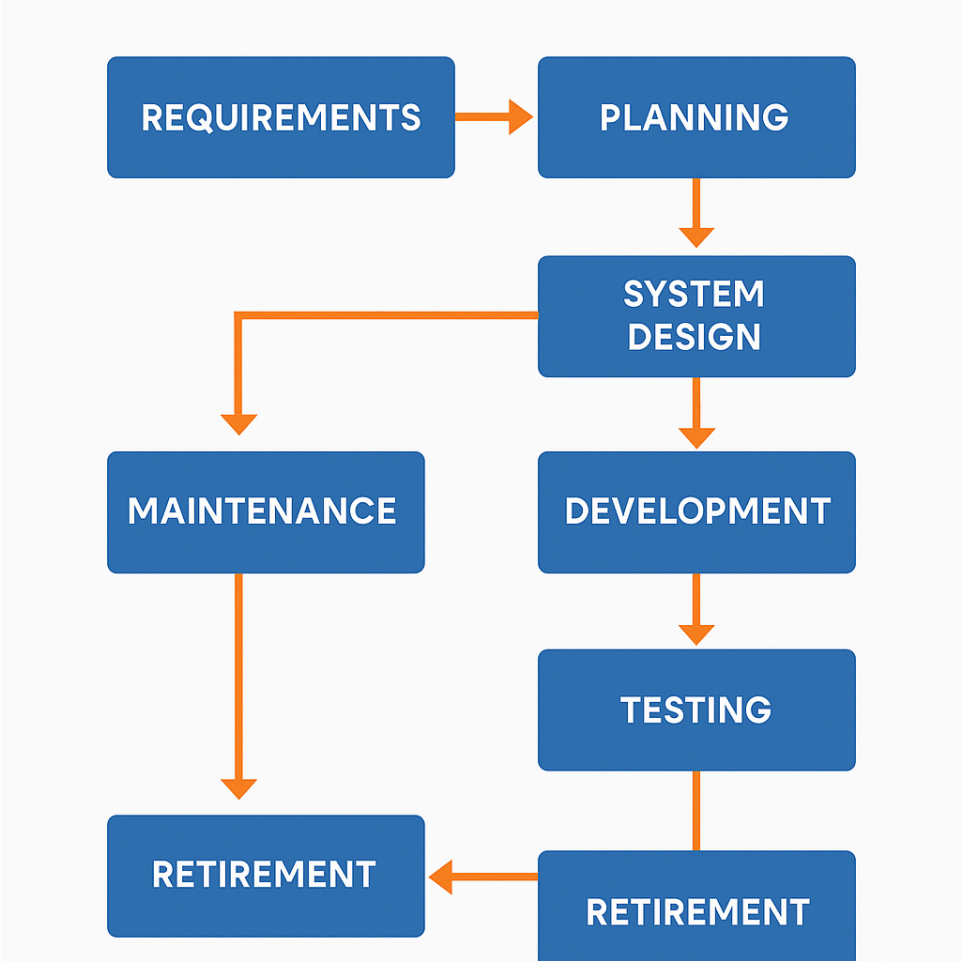
Step	Azure Boards Feature Used
Create login feature	Work Items (User Story)
Break down tasks	Task items
Track progress	Kanban Boards
Plan sprint	Backlogs & Sprint Planning
Monitor progress	Dashboards & Analytics
Trace work → code → test	End-to-end Traceability

In One Sentence

Azure Boards helps teams organize and track their work efficiently using Agile methodologies through work items, boards, backlogs, and reporting tools.

Q8) a) Explain application life cycle with its phases?
Ans. The **Application Life Cycle** (also called **Software Development Life Cycle – SDLC**) refers to the complete process involved in building, deploying, maintaining, and eventually retiring a software application.

It provides a structured approach to delivering high-quality software efficiently.



Phases of the Application Life Cycle

Different models (Agile, Waterfall, DevOps) may name the stages slightly differently, but the core lifecycle typically includes the following phases:

1. Requirements Gathering & Analysis

This is the first and most critical phase.

Purpose:

- Understand the business needs
- Identify user expectations
- Document functional and non-functional requirements

Output:

- Requirements Specification Document
- User stories (in Agile)

2. Planning

This stage defines how the project will be executed.

Activities:

- Estimating time and cost
- Resource allocation
- Risk assessment

- Choosing tools, technologies, and methodologies

Output:

- Project plan
- High-level roadmap

3. System Design

Here, the architecture and detailed technical design are created.

Includes:

- Database design
- UI/UX design
- System architecture
- Security design
- API + module blueprints

Output:

- Design specifications
- Wireframes, ER diagrams, architecture diagrams

4. Development (Coding)

Developers write the actual code based on the design.

Activities:

- Frontend & backend coding
- API development
- Database creation
- Configuring hosting or cloud services

Outputs:

- Working application modules
- Version-controlled source code

5. Testing

The application is tested to ensure it meets requirements and is free from defects.

Types of testing:

- Unit testing
- Integration testing
- Functional testing
- Performance testing
- Security testing
- User Acceptance Testing (UAT)

Outputs:

- Test reports
- Defect logs
- Verified build ready for deployment

6. Deployment

The tested application is deployed to the live environment.

Activities:

- Release planning
- CI/CD pipeline execution
- Production deployment
- Post-deployment validation

Outputs:

- Live application
- Deployment logs

7. Maintenance & Support

Once the application is running, it enters the maintenance phase.

Activities:

- Fixing bugs discovered in production
- Enhancing performance
- Adding new features
- Security updates
- Monitoring application health

Outputs:

- Updated versions
- Patches/releases

8. Retirement (Optional but Important)

When the application is no longer needed, it's retired.

Activities:

- Data archiving
- Migration to new systems
- Decommissioning servers or cloud resources

Output:

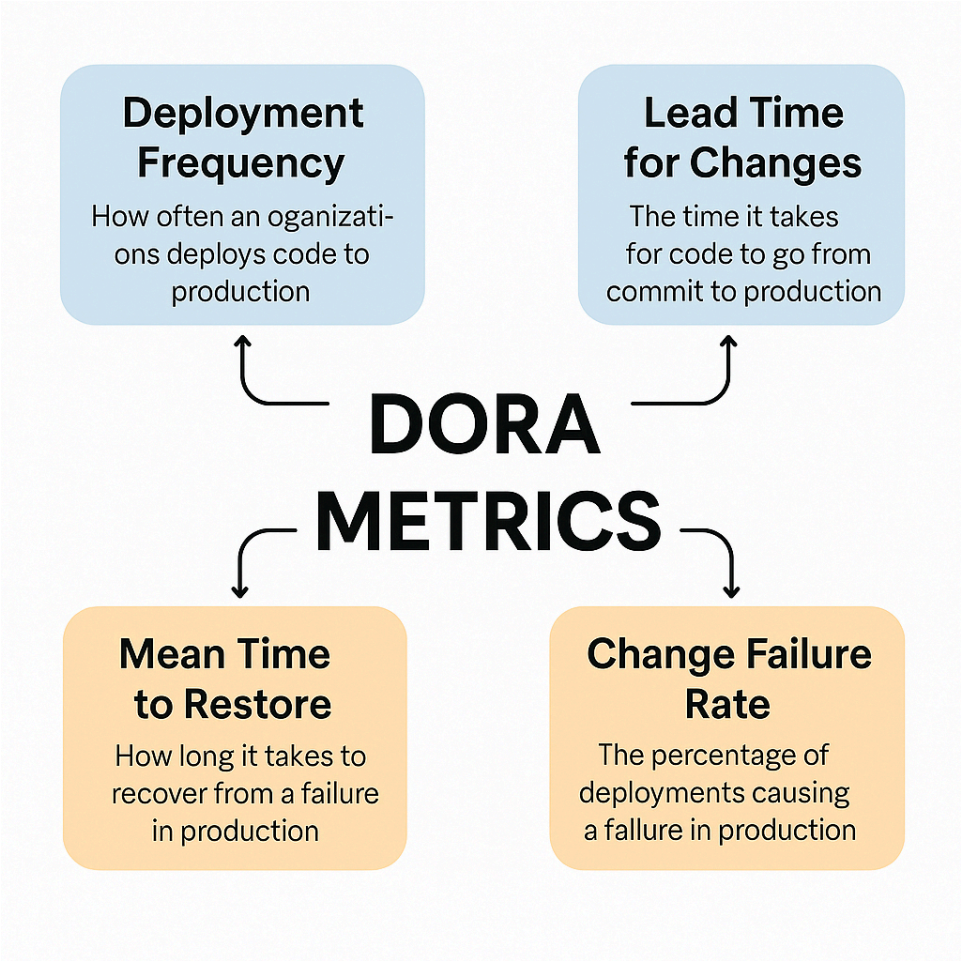
- Application gracefully shut down
- Replacement or upgraded system in place

Summary of Application Lifecycle Phases

Phase	Purpose
Requirements	Understand what needs to be built
Planning	Define scope, cost, and schedule
Design	Create technical and architectural blueprint
Development	Build the software
Testing	Verify quality and correctness
Deployment	Release software to users
Maintenance	Support, upgrade, and improve the software

Retirement	Decommission the application

Q8) b) Explain any four metrics used for developer practices?
Ans.



1. Deployment Frequency

What it measures:

How often code is deployed to production or released to users.

Why it matters:

- Indicates how quickly the team can deliver value
- High deployment frequency = mature DevOps practices
- Helps detect bottlenecks in build, test, or release processes

Example:

A team deploying updates **multiple times per day** shows strong automation and agility.

2. Lead Time for Changes

What it measures:

The time taken for a code change to move from commit → production.

Why it matters:

- Shorter lead time means faster feedback and higher responsiveness
- Long lead time reveals delays in code review, testing, or CI/CD pipelines

Example:

If a feature takes **2 hours from commit to deployment**, the team has great efficiency.

3. Change Failure Rate

What it measures:

Percentage of deployments that result in failures, such as incidents, rollbacks, or hotfixes.

Why it matters:

- Indicates code quality and effectiveness of testing
- Helps teams focus on improving validation and reducing production bugs

Example:

If out of 20 deployments, 2 cause issues → Change Failure Rate = **10%**.

4. Mean Time to Restore (MTTR)

What it measures:

How long it takes to recover from a failure in production.

Why it matters:

- Shows how resilient and well-prepared the team is during incidents
- Low MTTR means faster issue resolution and better user experience

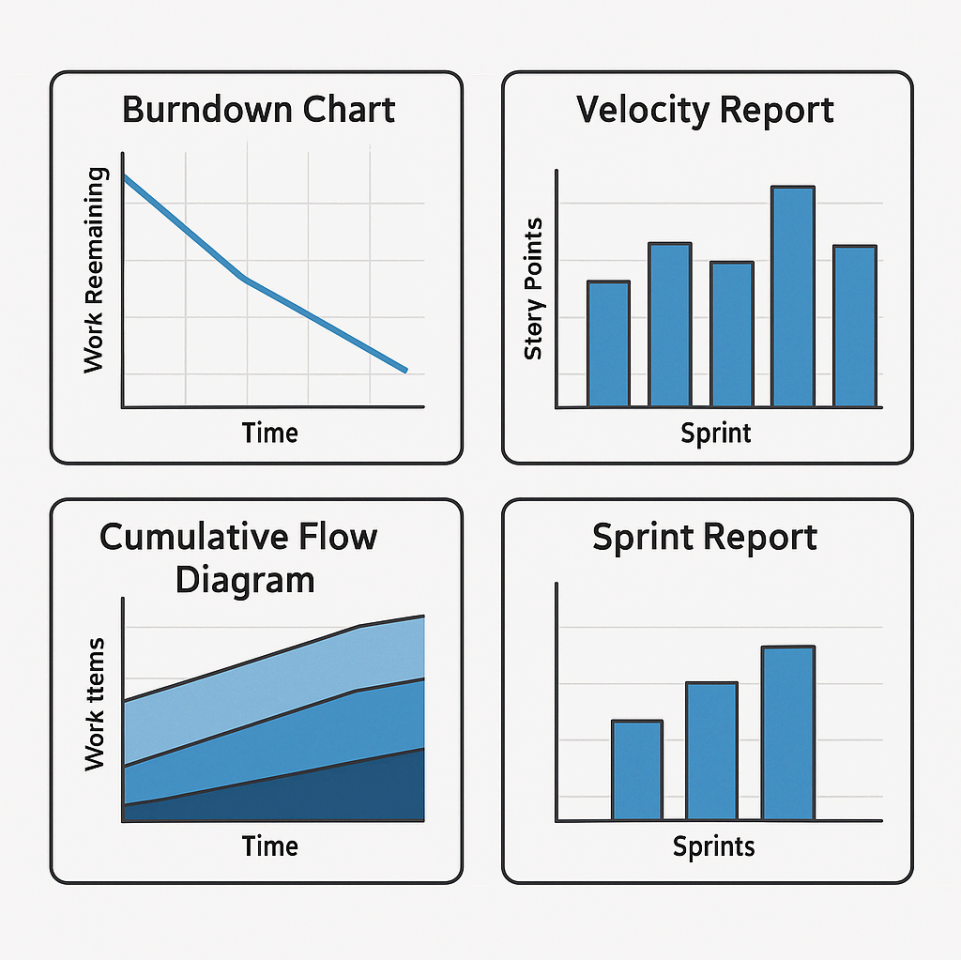
Example:

If a service outage is fixed in **20 minutes**, MTTR is 20 minutes—an indicator of strong operational maturity.

Summary Table

Metric	What It Shows	Why It's Important
Deployment Frequency	How often the team deploys	Measures speed of delivery
Lead Time for Changes	Time from commit → production	Indicates efficiency and agility
Change Failure Rate	% of deployments that fail	Reflects code quality & testing effectiveness
MTTR	Time to fix production issues	Measures system reliability & response capability

Q8) c) List any four examples of reports for metrics in agile projects.
Ans.



1. Burndown Chart

What it shows:

Tracks the amount of remaining work in a sprint over time.

Purpose:

- Helps teams see if they are on track to complete sprint goals
- Visualizes progress and identifies delays early

2. Velocity Report

What it shows:

Amount of work (story points) completed by the team in each sprint.

Purpose:

- Helps forecast future sprint capacity
- Useful for sprint planning and release planning

3. Cumulative Flow Diagram (CFD)

What it shows:

Visual representation of work items in different workflow stages (To Do → In Progress → Done).

Purpose:

- Identifies bottlenecks
- Shows stability of flow
- Helps evaluate cycle time and throughput

4. Sprint Report / Sprint Progress Report

What it shows:

Summarizes all work completed, not completed, and carried forward from a sprint.

Purpose:

- Provides transparency for sprint review
- Helps identify issues that blocked progress
- Supports continuous improvement

Summary Table

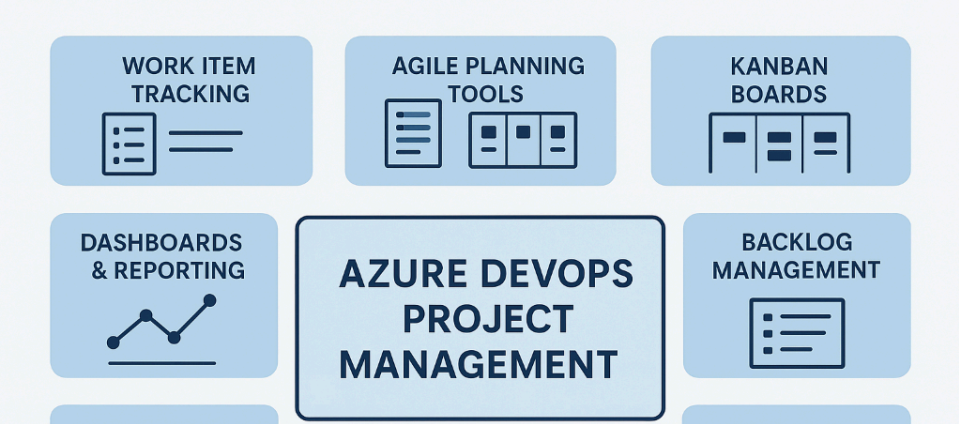
Report	What It Measures	Why It's Useful
Burndown Chart	Remaining work vs. time	Sprint tracking
Velocity Report	Story points completed per sprint	Forecasting & planning
Cumulative Flow Diagram	Flow of work across stages	Detect bottlenecks & improve flow
Sprint Report	Completed vs. pending sprint work	Review & improvement

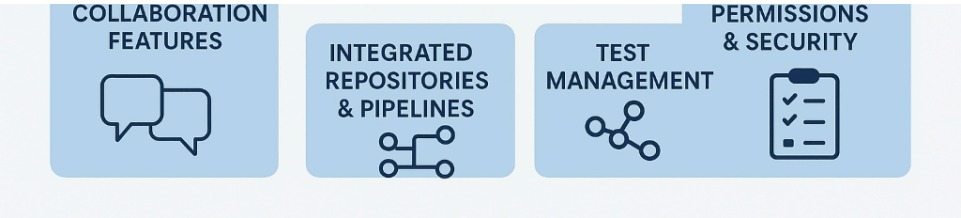
May – Jun 2023

Q7) a) Feature of Azure DevOps Project Management.
Ans.

Features of Azure DevOps Project Management

Azure DevOps provides a powerful set of tools that help teams plan, track, collaborate, and deliver software efficiently. Its project management capabilities primarily come from **Azure Boards**, but extend across the entire DevOps ecosystem.





1. Work Item Tracking

Azure DevOps uses work items to track all types of work.

Supports:

- User Stories
- Tasks
- Bugs
- Epics
- Features
- Issues

Features:

- Customizable fields and workflows
- Acceptance criteria, attachments, tags
- Audit history

2. Agile Planning Tools

Azure DevOps supports Agile, Scrum, and Kanban methodologies.

Includes:

- Product Backlog
- Sprint Backlog
- Sprint Planning Tools
- Agile Templates (Scrum, Basic, CMMI)

Benefits: Teams can plan iterations, manage priorities, and track progress easily.

3. Kanban Boards

Visual boards for managing the flow of work.

Features:

- Drag-and-drop cards
- Custom swimlanes
- Work-in-progress (WIP) limits
- Card customization

Benefit: Helps teams visualize and optimize workflow.

4. Dashboards & Reporting

Azure DevOps provides customizable dashboards for tracking metrics.

Examples of widgets:

- Burndown charts
- Velocity charts
- Cumulative flow diagrams
- Work item trends
- Build/Release status

Benefit: Improves visibility and data-driven decision-making.

5. Backlog Management

Backlogs help teams organize and prioritize work.

Features:

- Hierarchical work items (Epic → Feature → User Story → Task)
- Bulk update
- Estimation (Story points, Effort)
- Prioritization tools

6. Collaboration Features

Azure DevOps enhances teamwork through integrated communication tools.

Includes:

- Comments with @mentions
- Attachments, links, and updates
- Wiki and documentation
- Discussion threads on work items

7. End-to-End Traceability

Everything in Azure DevOps can be linked.

You can link:

- Work items ↔ Commits
- Work items ↔ Pull Requests
- Work items ↔ Test Cases
- Work items ↔ Builds & Releases

Benefit: Full traceability from planning → development → testing → deployment.

8. Integrated Repositories & Pipelines

Project management connects seamlessly with DevOps tools.

Examples:

- Code changes automatically linked to work items
- Builds triggered from commits
- Deployment status visible in dashboards

9. Test Planning and Test Management

Azure Test Plans integrates testing into project management.

Features:

- Manual test execution
- Test case management
- Traceability to requirements
- Tracking bugs & test results

10. Permissions & Security

Role-based access controls ensure secure collaboration.

Features:

- Team-based permissions
- Area paths & iteration paths
- Branch policies

Summary: Azure DevOps Project Management Features

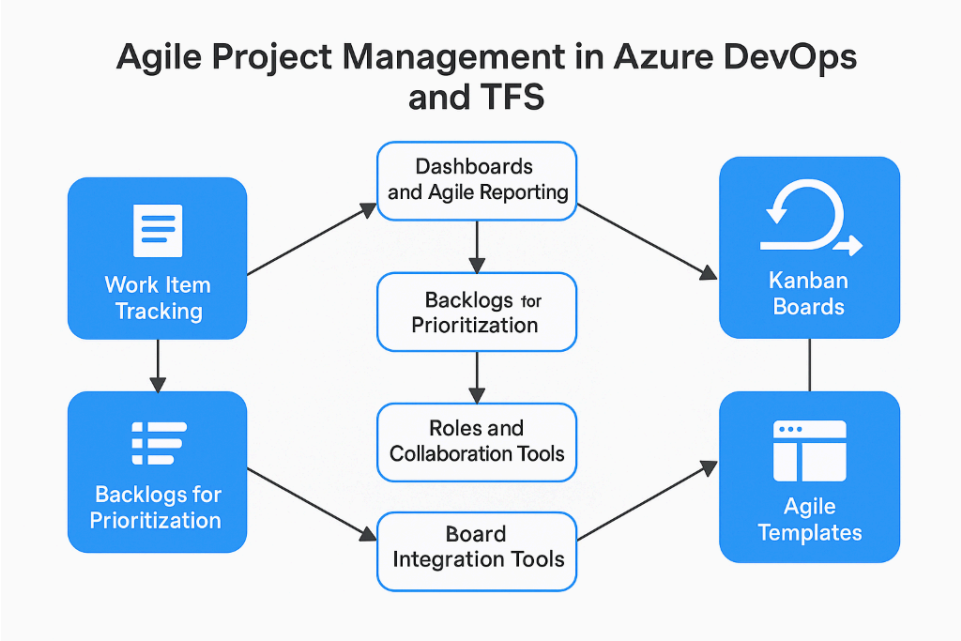
Feature	What It Enables
Work Item Tracking	Manage and track all work types
Agile Planning Tools	Support for Scrum/Kanban processes
Kanban Boards	Visual workflow management
Dashboards & Reports	Monitoring metrics and progress
Backlog Management	Prioritize and organize work
Collaboration Tools	Team communication & documentation
End-to-End Traceability	Link work → code → tests → release
Repos & Pipelines Integration	Seamless DevOps workflow
Test Management	QA planning and execution
Security & Permissions	Controlled access and governance

Q7) b) Explain the Agile Project Management in Azure DevOps and TFS.
Ans.

Agile Project Management in Azure DevOps and TFS

Azure DevOps (cloud-based) and TFS/Azure DevOps Server (on-premises) both support **Agile project management** by providing tools that help teams plan, track, and deliver work using Agile principles such as **Scrum**, **Kanban**, and **Iterative development**.

Both platforms share almost the same features — the main difference is **Azure DevOps is cloud-first**, while **TFS is server-based**.



1. Work Item Tracking (Core of Agile in Azure DevOps/TFS)

Azure DevOps and TFS use **work items** to track all types of work:

- **Epics** – Large business initiatives
- **Features** – Functional components
- **User Stories / Product Backlog Items (PBIs)** – Customer requirements
- **Tasks** – Development activities
- **Bugs** – Defects
- **Test Cases** – Quality checks

Each work item includes fields like *effort*, *priority*, *acceptance criteria*, and *status*.

Benefit:

Provides end-to-end traceability and transparency.

2. Backlogs for Prioritization

Both systems support **product and sprint backlogs**.

Backlog features:

- Drag-and-drop prioritization
- Story point estimation
- Hierarchical structure (Epic → Feature → User Story → Task)
- Filtering and tagging

Product Owners maintain this backlog to define the roadmap.

Benefit:

Clear visibility of future work and priorities.

3. Sprint Planning and Iterations

Azure DevOps and TFS support **Scrum-based sprint cycles**.

Sprint features:

- Assign work to a sprint
- Estimate effort (story points, hours)
- Allocate team capacity
- Forecast workload
- Track sprint goals

Teams plan work at the start of each sprint using built-in tools.

Benefit:

Structured iterative development with measurable progress.

4. Kanban Boards for Workflow Visualization

Both platforms include customizable **Kanban boards**.

Features:

- Columns (To Do → In Progress → Done)
- Swimlanes for priority work
- WIP (Work-In-Progress) limits
- Card customization
- Rules and automation

Excellent for Kanban teams or Scrum teams managing daily workflow.

Benefit:

Helps teams optimize flow and eliminate bottlenecks.

5. Dashboards and Agile Reporting

Azure DevOps and TFS offer built-in reporting for Agile metrics like:

- Burndown charts
- Velocity charts
- Cumulative flow diagrams (CFD)
- Work in progress reports
- Test case progress
- Build & Release summaries

Benefit:

Data-driven decision making and improved transparency.

6. Boards Integration with Code, Builds, and Tests

Azure DevOps/TFS allows linking:

- Work items ↔ Commits
- Work items ↔ Pull Requests
- Work items ↔ Test Cases
- Work items ↔ Builds & Releases

This allows teams to see:

✓ Why a change was made

✓ Which code belongs to which task
✓ What tests validate the work
✓ Deployment history

Benefit:

Complete traceability across the entire DevOps lifecycle.

7. Roles and Collaboration Tools

Both systems support Agile team collaboration:

- @mentions and comment threads
- Wiki for documentation
- Team dashboards
- Notifications and alerts
- Shared queries for tracking work

Benefit:

Improves communication and aligns the team.

8. Agile Templates (Scrum, Agile, CMMI)

Azure DevOps/TFS offers three process templates:

Agile Template

- User Stories + Tasks
- Best for teams using Agile/Kanban

Scrum Template

- Product Backlog Items (PBIs)
- Best for teams strictly following Scrum

CMMI Template

- Formal requirement + change management
- Best for enterprise/regulated environments

Benefit:

Teams can choose the methodology that fits their workflow.

Azure DevOps vs TFS (Short Comparison)

Aspect	Azure DevOps	TFS/Azure DevOps Server
Hosting	Cloud (SaaS)	On-premises
Updates	Continuous	Manual upgrades
Scalability	High	Depends on hardware
Integration	Cloud services	Local/enterprise systems
Features	Latest features first	Features released later

Agile functionality is almost identical, but Azure DevOps gets updates faster.

Summary

Agile Project Management in Azure DevOps and TFS includes:

1. **Work item tracking** for stories, tasks, bugs
2. **Backlogs** for prioritization
3. **Sprint planning tools** for Scrum teams
4. **Kanban boards** for workflow visualization
5. **Dashboards & reporting** for Agile metrics
6. **End-to-end traceability** from story → code → build → release
7. **Team collaboration tools**
8. **Support for Agile, Scrum, and CMMI processes**

Both tools help teams deliver iterative, high-quality software with transparency and efficiency.

Q7) c) Which methodologies are supported by Azure DevOps server to implement Agile project practices?
Ans. Azure DevOps Server (formerly TFS) supports Agile project practices through **three main process methodologies**, each implemented using **process templates**. These templates define how work is tracked, planned, and visualized.

Here are the **methodologies supported**:

1. Agile Methodology (Agile Process Template)

Designed for teams practicing general Agile or Kanban-style development.

Key Features:

- User Stories, Tasks, Bugs
- Story points for estimation
- Product & Sprint backlogs
- Kanban boards

Best For:

Teams wanting flexibility with modern Agile approaches.

2. Scrum Methodology (Scrum Process Template)

Fully aligned with the Scrum framework.

Key Features:

- Product Backlog Items (PBIs)
- Tasks
- Sprints & Iterations
- Burndown charts
- Velocity tracking

Best For:

Teams following formal Scrum ceremonies and iterative delivery.

3. CMMI Methodology (CMMI Process Template)

Supports more structured, documentation-driven project management.

Key Features:

- Requirements
- Change Requests
- Risks & Issues
- Formal workflows and approvals
- High traceability

Best For:

Enterprises requiring compliance, audits, or heavyweight processes.

Summary

Azure DevOps Server supports Agile project management through three methodologies:

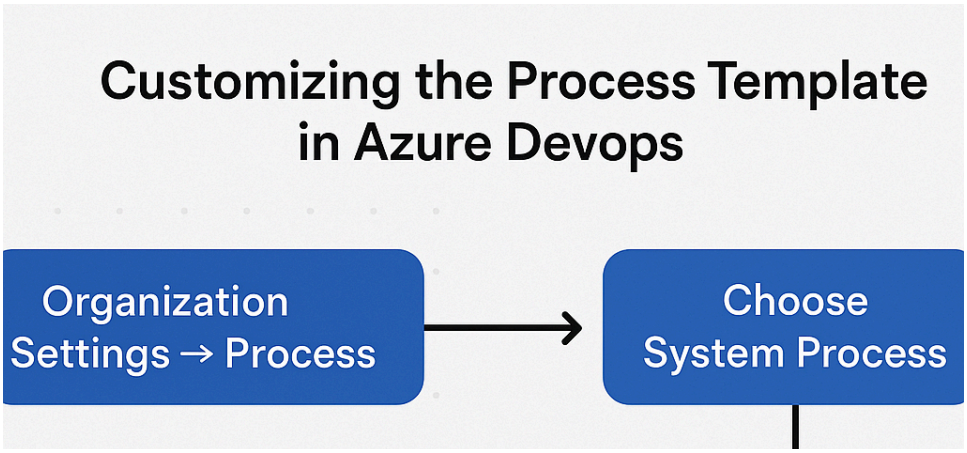
Methodology	Process Template	Description
Agile	Agile	Flexible, lightweight Agile/Kanban approach
Scrum	Scrum	Iterative development using Scrum framework
CMMI	CMMI	Structured, enterprise-grade process model

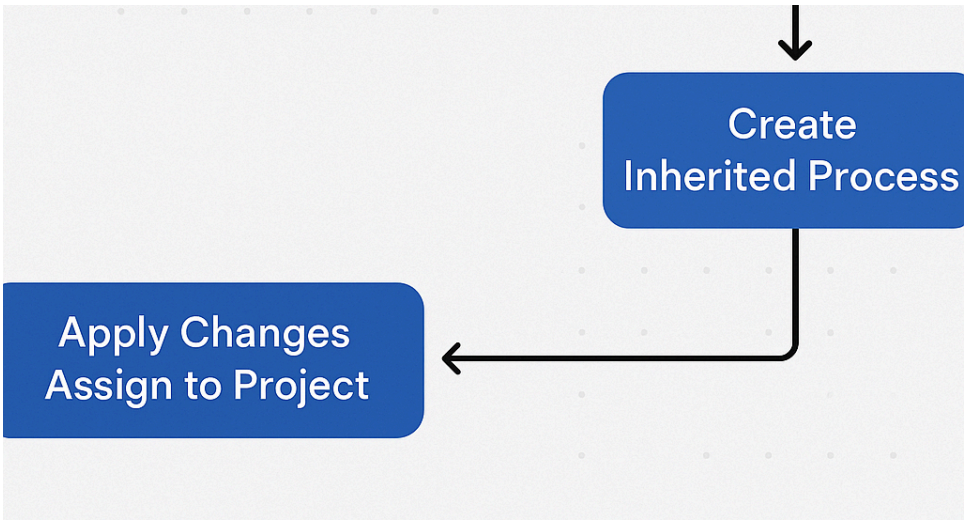
Q8) c) Explain the Customizing the Process Template in Azure DevOps.
Ans.

Customizing the Process Template in Azure DevOps

In Azure DevOps, a **process template** defines how work is tracked. It includes work item types, workflows, states, fields, rules, forms, and backlogs. Customizing the process template allows organizations to tailor Azure DevOps to their own project, compliance, or business needs.

Azure DevOps offers **two ways** to customize depending on the type of process:





1. Customizing Inherited Processes (Cloud – Azure DevOps Services)

This is the *recommended* and easiest method.

✓ What You Can Customize:

- Add or modify **Work Item Types**
- Add new **fields**
- Change **state workflows** (e.g., New → Active → Done)
- Customize **layouts/forms** (tabs, groups, controls)
- Customize **rules** (required fields, default values)
- Add/remove **custom backlogs and boards columns**
- Add **tags**, **custom states**, and **custom transitions**

✓ How It Works:

Go to **Organization Settings** → **Process**
Choose a system process (Agile, Scrum, Basic, CMMI)
Create Inherited Process
Apply changes visually, without XML editing
Assign the inherited process to your project

✓ Benefits:

- No downtime
- Immediate effect
- Safe and easy (GUI-based)
- No need to export/import templates
- Supported for cloud environments

2. Customizing XML-Based Process Template (Azure DevOps Server / TFS – On-Premises)

Used only for **on-prem Azure DevOps Server/TFS**.

✓ What You Can Customize:

- Work item type definitions (WITD)
- Process template configuration
- Categories
- Portfolio backlogs
- Reports
- Security settings

- Workflow states & transitions
- Field rules and validation

✓ How It Works:

Export the process template using `witadmin` tool
Modify XML files (work item types, categories, workflow)
Import the updated template back into the server
Apply it to one or more projects

✓ Tools Used:

- **witadmin.exe**
- XML editors
- Process Template Editor (older versions of VS)

✓ Use Cases:

- Highly regulated industries
- Complex workflows
- Advanced automation rules
- Enterprise-grade process control

Common Customizations Teams Perform

☐ Add new Work Item Types

Examples:

- Risk
- Change Request
- Test Scenario
- Stakeholder Requirement

Add Custom Fields

Examples:

- Customer Priority
- Business Value
- Effort Remaining
- Risk Level

☐ Modify Workflow States

Examples:

- “Ready for QA”
- “In Review”
- “Pending Approval”
- “Blocked”

☐ Customize Kanban Columns

Align board columns with actual team workflow.

☐ Add New Backlog Levels

Examples:

- Initiative
- Capability

☐ Add Rules

Examples:

- Auto-populate values
- Make fields required on state change
- Enforce business policies

Inherited vs XML Process Customization (Quick Comparison)

Feature	Azure DevOps Services (Cloud)	Azure DevOps Server/TFS
Customization Type	Inherited (GUI)	XML-based
Complexity	Easy	Advanced
Risk	Low	Medium/High
Flexibility	Medium	Very High
Downtime	None	Possible
Custom WITs	Yes	Yes
Workflow changes	Yes	Yes

Summary

Azure DevOps allows customizing process templates to adapt work tracking to organizational needs. In the cloud version, you customize using inherited processes through a GUI, modifying work item types, fields, states, and rules. In Azure DevOps Server/TFS, customization is done by editing and importing XML-based process templates, allowing deeper control over workflows, backlogs, and work item definitions.

Nov– Dec 2023

Q7) a) What is Agile Project Management? List two benefits of using Azure DevOps for Agile Project Management

Ans. **Agile Project Management** is an iterative and flexible approach to managing software development projects.

It focuses on:

- Delivering work in small, incremental cycles (iterations or sprints)
- Continuous feedback and improvement
- Collaboration between teams and stakeholders
- Adapting to changes quickly

Agile emphasizes **frequent delivery**, **customer involvement**, and **responding to change** rather than following rigid plans.

Two Benefits of Using Azure DevOps for Agile Project Management

1. Excellent Tools for Planning & Tracking Work

Azure DevOps provides **Boards**, **Backlogs**, **Sprints**, and **Kanban boards** that help teams:

- Prioritize user stories
- Track tasks and bugs
- Plan sprints efficiently
- Visualize work using Kanban

This makes project management easier and more transparent.

2. End-to-End Traceability Across the Development Lifecycle

Azure DevOps links **work items → commits → builds → tests → releases**.

This allows teams to:

- Track why a change was made
- Validate work with connected test cases
- Monitor deployment progress
- Improve accountability and quality

It provides a unified view of the entire development process.

Short Answer (Quick Revision)

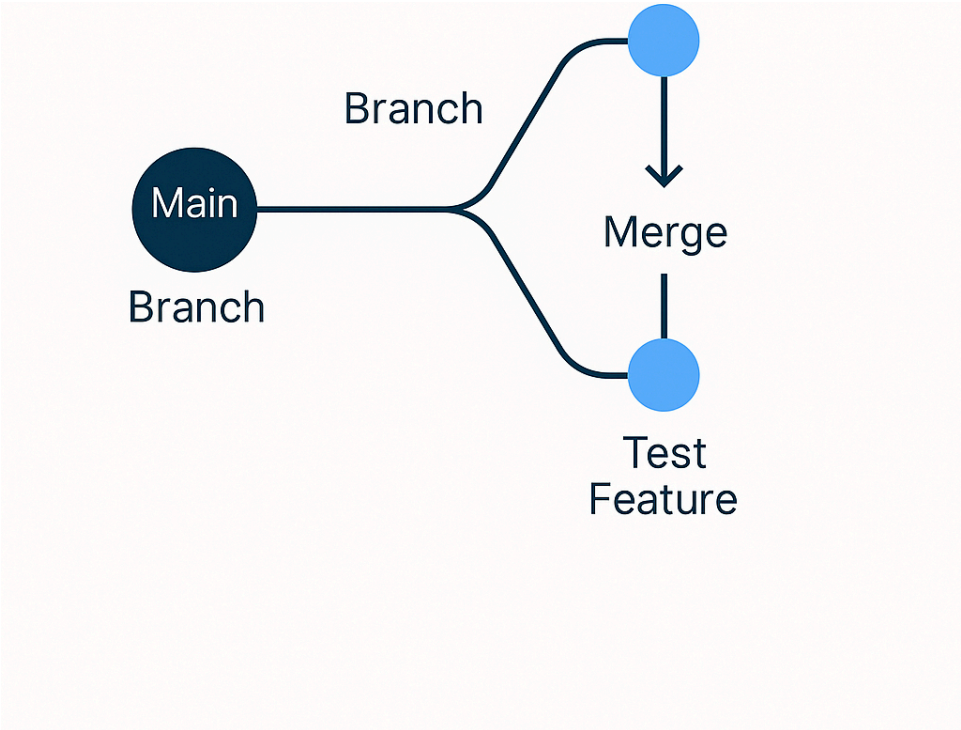
Agile Project Management is an iterative approach that promotes flexibility, collaboration, and continuous improvement.

Two benefits of using **Azure DevOps** for Agile Project Management are:
Powerful work tracking tools (Boards, Backlogs, Sprints, Kanban)
Full traceability from planning to deployment.

Q7) b) Explain the concept of branching and merging in Azure DevOps and how it is used to support Agile Project Management.
Ans.

Branching and Merging in Azure DevOps

In Azure DevOps (using Git), **branching and merging** are essential version control practices that help teams work in parallel, manage code changes, and deliver features incrementally—perfectly aligning with Agile principles.



Branching

Branching means creating a separate copy of the code where developers can work independently without affecting the main codebase.

Purpose of Branching:

- Isolate work on new features
- Prevent unfinished code from impacting production
- Allow multiple developers or teams to work in parallel
- Support experiments, bug fixes, and enhancements safely

Common Branch Types in Azure DevOps:

Main (or Master) Branch

- Stable production-ready code

Develop Branch

- Integration branch for ongoing work

Feature Branches

- Used for user stories or enhancements

Bugfix Branches

- Used to fix defects

Release Branches

- Prepare for a software release

Hotfix Branches

- Quick fixes for production emergencies

Merging

Merging combines changes from one branch into another branch, usually after the work is completed.

Purpose of Merging:

- Integrate new features into the main codebase
- Bring updates from the main branch into working branches
- Keep code consistent and synchronized

In Azure DevOps, merging typically happens through a **Pull Request (PR)**.

How Branching & Merging Support Agile Project Management

Agile requires teams to deliver features iteratively, collaborate frequently, and maintain a stable product. Branching and merging make this easier:

1. Supports Parallel Work on Multiple User Stories

Each **user story** or **task** can have its own branch.

Benefit:

- Developers work independently
- No code conflicts during sprint execution
- Reduces integration issues

2. Provides Safe Isolation for Feature Development

Feature branches prevent incomplete or unstable code from affecting the main application.

Benefit:

- The sprint deliverable remains stable
- Teams commit frequently without breaking builds

3. Enables Continuous Integration (CI)

Azure DevOps Pipelines automatically build and test the code whenever a branch is updated.

Benefit:

- Immediate feedback
- Higher code quality
- Early defect detection

4. Pull Requests Support Code Reviews

Before merging, PRs allow team members to:

- Review changes
- Add comments
- Enforce policies (build success, approvals, tests)

Benefit:

- Ensures quality and learning
- Enhances team collaboration

5. Facilitates Release Planning

Branches like **release** and **hotfix** align with Agile release strategies.

Benefit:

- Teams can finish sprints while handling production issues in parallel
- Releases can be stabilized without interrupting new development

Example (Agile Sprint Workflow in Branching)

Sprint starts
– Developer creates a feature branch:
`feature/user-login`
Developer codes and commits frequently
CI pipeline builds and tests the branch
Work is completed → PR created to merge into `develop`
Review + approval + automated tests
Merge into `develop` → integrated
At end of sprint → merge into `main` or release branch

This ensures **clean, continuous delivery** aligned with Agile.

Summary

Branching in Azure DevOps allows developers to work independently on features, bugs, or releases without affecting the main code.

Merging integrates those changes back into shared branches using Pull Requests.

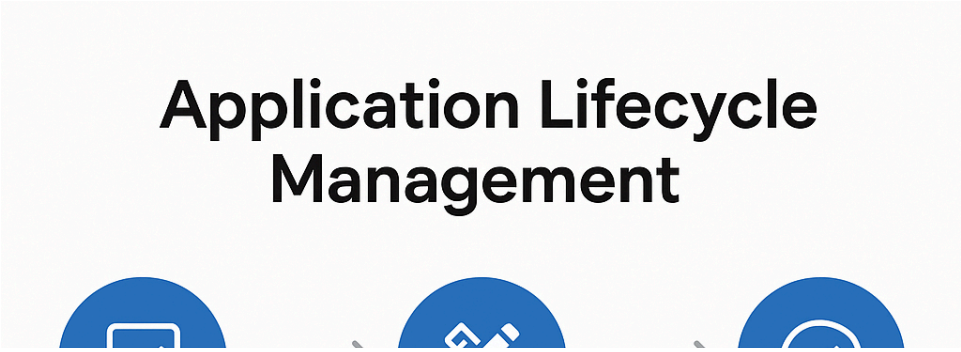
These practices support Agile project management by enabling:

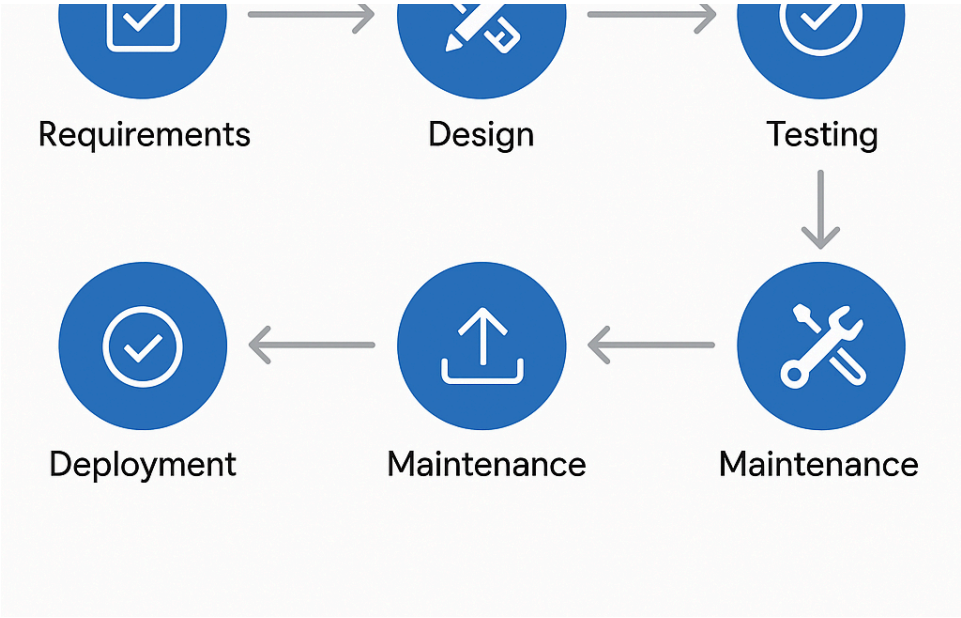
- Parallel work on multiple user stories
- Stable code during sprints
- Continuous integration and testing
- Collaboration through PR reviews
- Controlled and predictable releases

Q8) a) What are the different stages of the application lifecycle management (ALM) process? What are the key features of Azure DevOps that support ALM?
Ans.

Stages of the Application Lifecycle Management (ALM) Process

ALM covers the entire lifespan of a software application — from idea to retirement. The major stages include:





1. Requirements Management

- Collecting, documenting, and prioritizing business needs
- Creating user stories, acceptance criteria, functional specs

Goal: Ensure clarity of what needs to be built.

2. Project Planning

- Estimating effort
- Creating schedules, sprints, and roadmaps
- Assigning responsibilities

Goal: Plan how and when work will be delivered.

3. Design

- System architecture design
- UI/UX design
- Database and API design

Goal: Provide a technical blueprint for the solution.

4. Development (Coding)

- Implementing features
- Writing code, unit tests
- Committing and version controlling code

Goal: Build the software according to design specs.

5. Testing

- Functional testing
- Unit, integration, regression testing
- User acceptance testing (UAT)

Goal: Ensure quality, performance, and correctness.

6. Deployment / Release Management

- Automating builds and releases
- Deploying to staging and production environments
- Release versioning

Goal: Deliver the application to users reliably.

7. Maintenance & Support

- Fixing bugs in production
- Improving performance
- Enhancing features
- Monitoring system health

Goal: Ensure stability and continual improvement.

8. Retirement (End of Life)

- Decommissioning the application
- Migrating users/data
- Archiving documentation

Goal: Safely phase out outdated applications.

Key Azure DevOps Features That Support ALM

Azure DevOps provides an integrated toolset that supports every stage of ALM:

1. Azure Boards – Planning & Requirements Management

- Work items (Epics, Features, Stories, Tasks, Bugs)
- Product & sprint backlogs
- Kanban boards
- Dashboards and analytics

Supports: Requirements, planning, progress tracking.

2. Azure Repos – Version Control

- Git repositories
- Branching, merging, pull requests
- Code reviews & policies
- Traceability from code → work items

Supports: Development, collaboration, code management.

3. Azure Pipelines – CI/CD Automation

- Build automation
- Automated testing
- Release pipelines
- Multi-platform support (Windows, Linux, containers, cloud)

Supports: Development, testing, deployment.

4. Azure Test Plans – Manual & Automated Testing

- Manual and exploratory test execution
- Test case management
- Bug tracking
- Traceability to user stories

Supports: Quality assurance and testing.

5. Azure Artifacts – Package Management

- Store and share packages: NuGet, npm, Maven, Python
- Versioning and dependency management

Supports: Build & release management, package reuse.

6. Monitoring & Feedback Integration

- Integration with Azure Monitor, App Insights
- Telemetry, logs, performance metrics
- Continuous feedback loops

Supports: Maintenance, improvement, system monitoring.

Summary

Stages of ALM

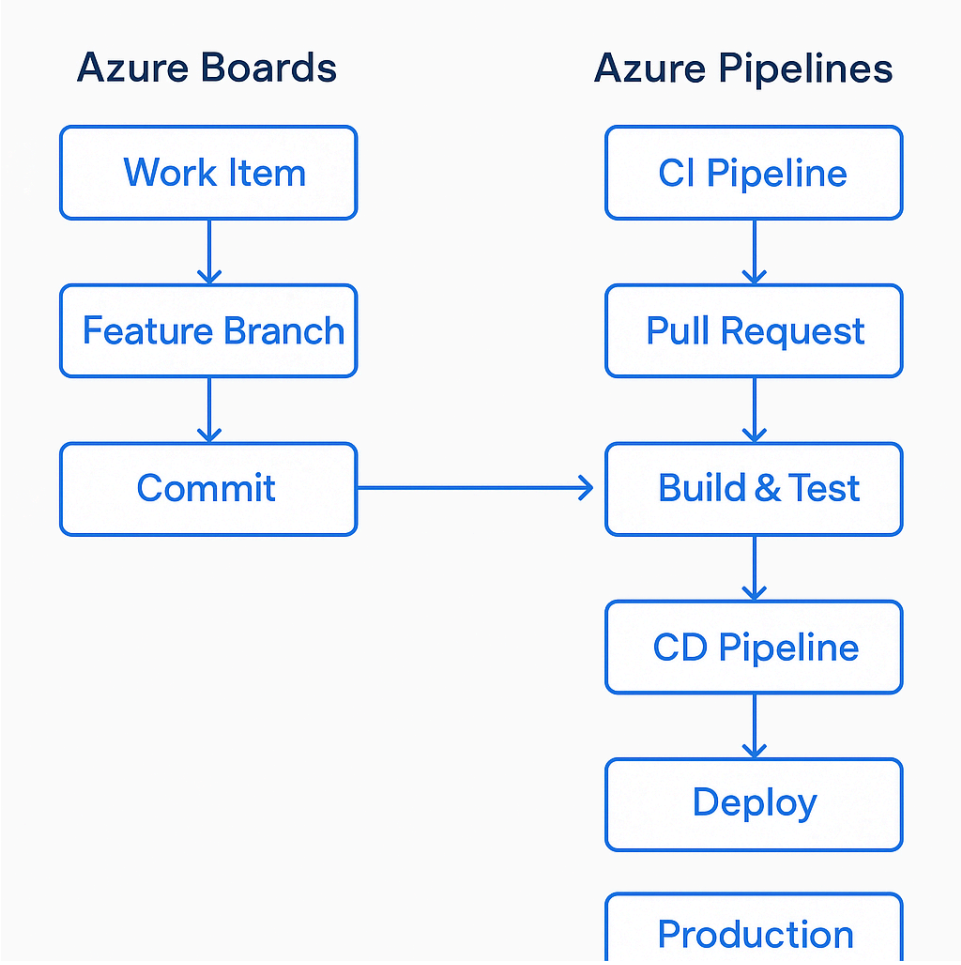
- Requirements
- Planning
- Design
- Development
- Testing
- Deployment
- Maintenance
- Retirement

Azure DevOps Features Supporting ALM

- **Azure Boards** → Requirements & project planning
- **Azure Repos** → Version control & collaboration
- **Azure Pipelines** → CI/CD automation
- **Azure Test Plans** → Testing & QA
- **Azure Artifacts** → Package management

- **Monitoring integrations** → Feedback & operations

Q8) b) Explain how to use Azure Pipelines to implement continuous integration and continuous delivery (CI/CD) in an Agile project. Discuss the different ways to manage work items in Azure Boards.
Ans.



Part 1 — Using Azure Pipelines for CI/CD in an Agile project

Goal: get code from developer machines into production quickly, safely, and repeatedly — aligned to short Agile iterations.

High-level CI/CD flow

9. Developer works on a **feature/bugfix branch** for a work item.
10. Commits → **CI pipeline** runs (build + unit tests + static analysis).
11. Developer opens a **Pull Request (PR)**; PR triggers validation builds and required checks.
12. After PR review + approvals, merge into `develop`/`main` (or release branch).
13. Merge triggers further pipeline stages: integration tests, packaging, publish artifacts.
14. CD pipeline deploys to environments (staging → canary → production), with approvals/gates and automated smoke tests.
15. Monitoring/telemetry feed back to team; incidents create work items for fixes.

Key Azure Pipelines capabilities to use

1. Pipeline-as-code (YAML)

- Store pipeline definition in the repo (`azure-pipelines.yml`).
- Versioned with code, repeatable and reviewable.

2. CI triggers

- `trigger:` for branch CI builds (on push).
- `pr:` to run pipeline on PRs.

3. Build steps

- Compile, run unit tests, run lint/static analysis, create build artifacts.

4. Pull Request validation & branch policies

- Require successful build, reviewers, work item linking, and passing policies before merge.
- Helps enforce quality and traceability.

5. Continuous Delivery (multi-stage pipelines)

- Define stages: `build` → `test` → `deploy-staging` → `deploy-prod`.
- Use approvals, checks, and environment-level gates.

6. Environments & deployment targets

- Environments represent *staging, QA, production*; support approvals, deployment history, and resource views.
- Deploy to VMs, containers, AKS, serverless, or cloud services.

7. Release strategies

- Support for rolling, blue/green, canary, and A/B (usually via pipeline tasks or deployment tooling).
- Feature flags + dark launches for safer releases.

8. Artifacts & package feeds

- Publish build artifacts and use Azure Artifacts (NuGet, npm, Maven) for dependency management.

9. Secrets / service connections

- Use variable groups linked to Azure Key Vault or pipeline-level secure variables for credentials.
- Configure service connections for cloud providers, Kubernetes, container registries.

10. Gates & checks

- Approval gates, automated checks (e.g., query Azure Monitor, run integration smoke tests) before promotion.

11. Integration with test tools

- Run automated test suites (unit, integration, UI), publish test reports and code coverage.

12. Monitoring & rollback

- Integrate with Application Insights / monitoring tools and automate rollback or alerts for failed health checks.

Recommended CI/CD best practices (Agile-friendly)

- **Small frequent merges** (short-lived feature branches).
- **Fast CI feedback** (unit tests first; heavy tests in later stages).
- **Protect main branch** with PR validation and branch policies.
- **Pipeline as code** for repeatability and review.
- **Automate promotions** with manual approvals only where required.
- **Use feature flags** to release incomplete features safely.

- **Traceability:** link commits/PRs/builds/releases to work items so each change has context.
- **Monitor & iterate**: use telemetry to feed backlog items for improvements/bugs.

Minimal example: a simple YAML CI pipeline

```
# azure-pipelines.yml
trigger:
  branches:
    include:
      - main
      - develop
pr:
  branches:
    include:
      - develop

pool:
  vmImage: 'ubuntu-latest'

variables:
  buildConfiguration: 'Release'

stages:
- stage: Build
  jobs:
  - job: Build
    steps:
    - task: UseDotNet@2
      inputs:
        packageType: 'sdk'
        version: '7.x'
    - script: dotnet build --configuration $(buildConfiguration)
      displayName: 'Build'
    - script: dotnet test --configuration $(buildConfiguration) --no-build --logger trx
      displayName: 'Run unit tests'
    - publish: $(Build.BinariesDirectory)
      artifact: drop
```

```
- stage: DeployToStaging
  dependsOn: Build
  condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/develop'))
  jobs:
  - deployment: Deploy
    environment: 'staging'
    strategy:
      runOnce:
        deploy:
          steps:
          - download: current
          - script: echo "Deploying to staging..."
```

Use additional stages for integration tests, canary, and production with approvals.

Traceability: linking work items to code & pipelines

- Include the **work item ID** in commit messages or PRs so Azure Boards links them to commits and builds (e.g., mention the ID in the PR description). This creates an automatic trace from work item → commit → build → release.
- Use the PR work item linking UI to explicitly associate work items with PRs and builds.
- Configure branch policies to **require** a linked work item for PR completion (enforces traceability).

Part 2 — Different ways to manage work items in Azure Boards

Azure Boards is flexible. Here are the main ways teams manage work items, with use-cases and tips.

1. **Work item types & hierarchy**

- **Work items:** Epics → Features → User Stories (or PBIs) → Tasks → Bugs.
- Use the hierarchy to model product roadmap → releases → sprint backlog → execution tasks.
- **Tip:** keep stories small and link tasks to their parent story.

2. **Backlogs**

- **Product backlog:** prioritize features/stories for the product.
- **Sprint backlog:** subset of the product backlog committed for a sprint.
- **Features:** reorder by priority; use effort/story points for planning.

3. Boards (Kanban)

- Visualize flow: To Do → In Progress → Review → Done.
- **Swimlanes** for expediting or separating classes of work (bugs, high priority).
- **WIP limits** to control in-progress work and expose bottlenecks.
- Customize card fields and rules to display important info at a glance.

4. Sprint planning & capacity

- Use the **sprint planning** tool to assign stories/tasks to team members.
- Set team capacity and use effort estimates to avoid overcommit.
- Burndown charts show sprint progress.

5. Queries & saved views

- Custom **queries** for status, release, owner, or SLA.
- Use query-based charts and add to dashboards for visibility.
- Examples: “Open bugs by area”, “Work items without owner”, “Work items blocked > 3 days”.

6. Boards rules & automation

- Add **rules** to set field values on state changes, enforce fields, or auto-assign reviewers.
- Use service hooks / Azure Logic Apps to automate external workflows (e.g., Slack notifications on state change).

7. Linking & traceability

- Link work items to **commits**, **PRs**, **builds**, **test cases**, and **releases**.
- This creates end-to-end traceability and helps in audits and postmortems.

8. Tags, area paths & iteration paths

- **Tags**: flexible grouping (e.g., `customerX`, `security`).
- **Area paths**: logical subsystems or teams; useful for permissions/reporting.
- **Iteration paths**: sprints/releases; used for planning and sprint backlogs.

9. Templates & process customization

- Create work item templates for recurring tasks (pre-filled fields).
- Customize processes (inherited process for cloud or XML templates on server) to add fields, states or rules.

10. Boards for Kanban vs Scrum

- **Scrum teams** use sprint backlogs, PBIs, and burndown.
- **Kanban teams** prefer continuous flow via a single backlog and Kanban board with WIP limits.
- Azure Boards supports both; choose the view/process that fits your team.

11. Test plans & linking to QA

- Link test cases to requirements (work items) and associate test runs with builds.
- Use Test Plans to manage manual exploratory testing during iterations.

12. Dashboards & reporting

- Dashboards aggregate charts, pipeline status, queries, and burndowns for stakeholders.
- Use **Analytics** (Power BI integration) for advanced reporting (lead time, cycle time, throughput).

Practical examples / recommended patterns

- **Feature work**: Create a Feature → split into Stories → create Feature branches for each Story → run CI for each branch → PR links to Story → merge to `develop` → automated integration + deployment to staging.
- **Hotfix**: Create Hotfix work item + hotfix branch from `main` → fix → CI + quick release to production (CD with approval).

Quick recap (one-liner)

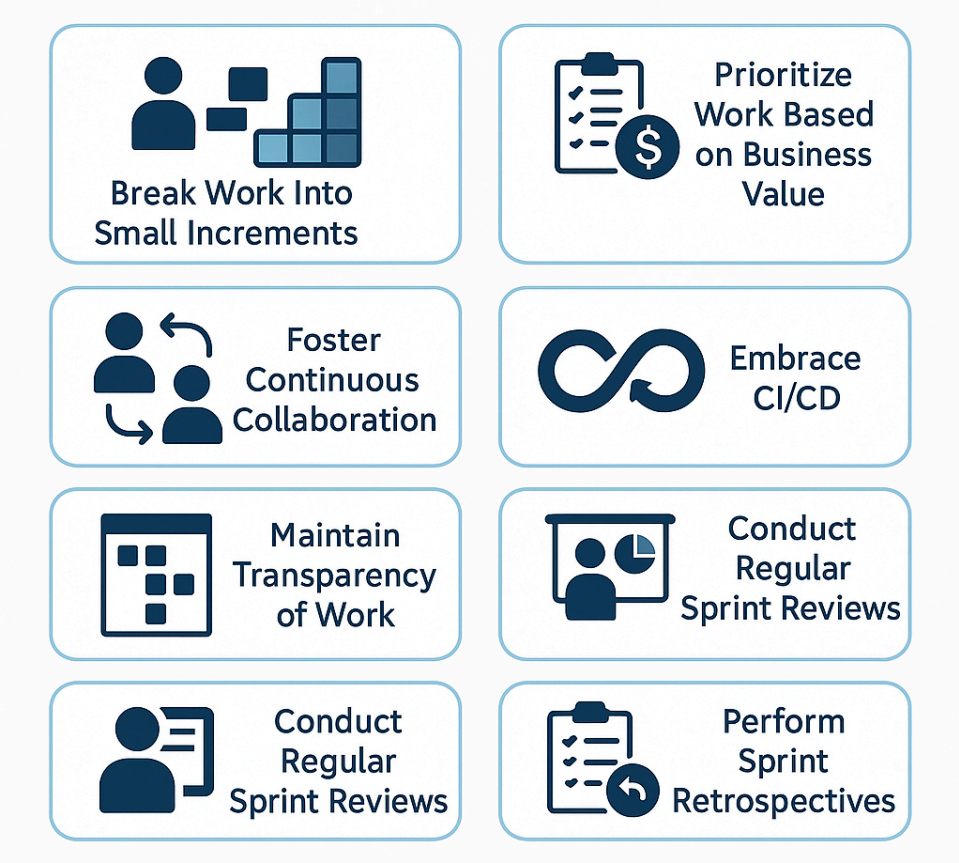
- **Azure Pipelines** implements CI/CD by automating builds, tests, and staged deployments (pipeline-as-code, PR validation, environments, approvals, and deployment strategies).
- **Azure Boards** manages work items with hierarchical work items, backlogs, Kanban/Scrum boards, rules, queries, links to code & pipelines, and dashboards for traceability and planning.

Nov– Dec 2024

Q7) a) Explain best practices for Agile management.
Ans.

Best Practices for Agile Management

Agile management focuses on delivering value iteratively, collaborating effectively, and adapting to change quickly. The following best practices help teams achieve consistent and predictable results:



1. Break Work into Small, Manageable Increments

Agile teams should divide features into small user stories that can be completed within a sprint.

Benefits: Faster delivery, easier planning, quicker feedback.

2. Prioritize Work Based on Business Value

Use a well-maintained product backlog where items are ranked by customer value, risk, and strategic importance.

Benefits: Ensures teams work on the most impactful items first.

3. Use Daily Stand-Up Meetings

Short, focused meetings help track progress and identify blockers early.

Benefits: Improved team communication and issue resolution.

4. Foster Continuous Collaboration

Encourage collaboration between developers, testers, business users, and other stakeholders.

Benefits: Shared understanding and fewer misunderstandings.

5. Embrace Continuous Integration and Continuous Delivery (CI/CD)

Automate builds, tests, and deployments to reduce manual errors and accelerate delivery.

Benefits: Faster feedback, higher quality, smoother releases.

6. Maintain Transparency of Work

Use visual tools like Kanban boards, sprint backlogs, and dashboards.

Benefits: Allows everyone to see progress and detect bottlenecks.

7. Conduct Regular Sprint Reviews

Demonstrate completed work to stakeholders at the end of each sprint.

Benefits: Ensures alignment and enables quick feedback.

8. Perform Sprint Retrospectives

Teams should reflect on what worked and what didn't and identify improvements.

Benefits: Continuous improvement and better team performance.

9. Limit Work in Progress (WIP)

Kanban WIP limits prevent overloading team members and help maintain focus.

Benefits: Faster flow, reduced context switching.

10. Encourage Test-Driven Development (TDD) & Automated Testing

Testing should be part of the development process, not an afterthought.

Benefits: Higher code quality and fewer defects.

11. Promote Self-Organizing Teams

Empower teams to make decisions on how work should be executed.

Benefits: Higher ownership, accountability, and motivation.

12. Adapt and Respond to Change

Agile encourages responding to changes even late in the development cycle.

Benefits: Builds flexible, customer-focused products.

Short Answer

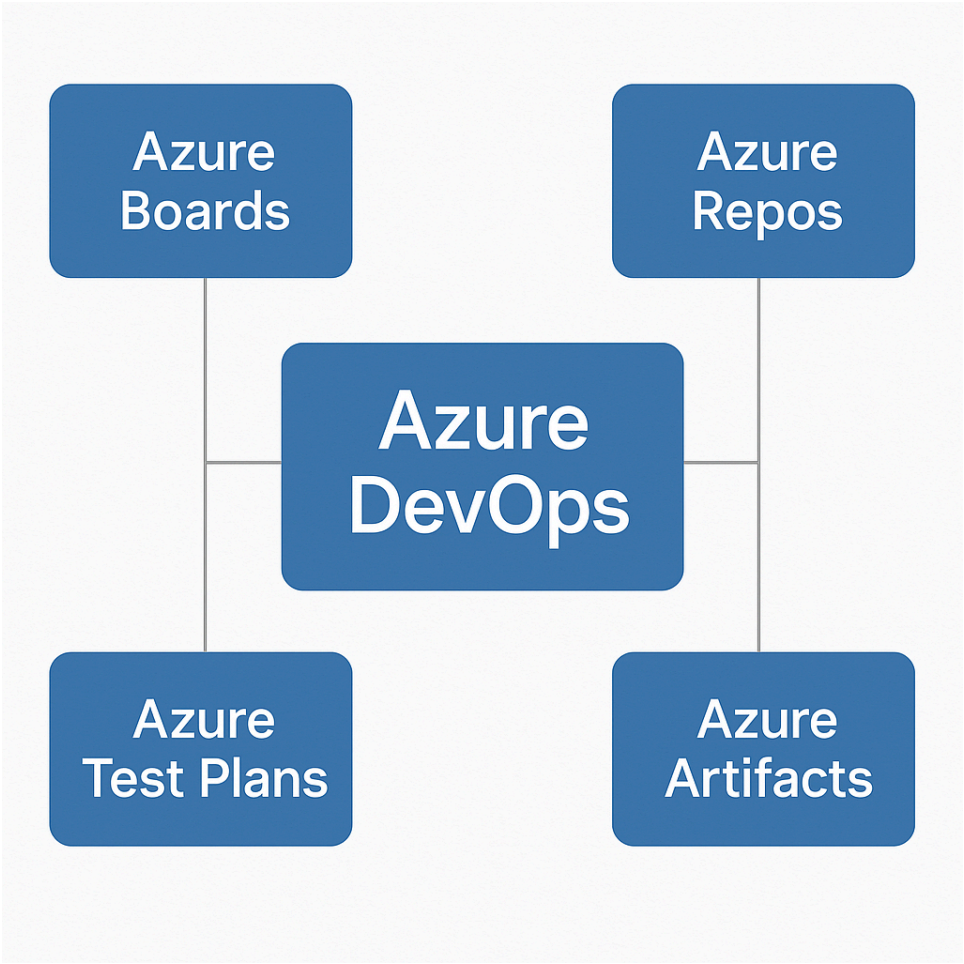
Best practices for Agile management include breaking work into small increments, prioritizing based on business value, maintaining transparency with visual boards, fostering collaboration, using CI/CD, limiting work in progress, holding daily stand-up meetings, and conducting sprint reviews and retrospectives to ensure continuous improvement.

Q7) b) Write short note on fundamental components of Azure DevOps.
Ans.

Fundamental Components of Azure DevOps

Azure DevOps is a complete end-to-end DevOps platform that helps teams plan, build, test, deliver, and monitor software efficiently.

It consists of **five core components**, each supporting a different stage of the software development lifecycle.



1. Azure Boards – Work Tracking & Agile Project Management

Azure Boards is used to plan, track, and manage work across the team.

Key Features:

- Work items (Epics, Features, User Stories, Tasks, Bugs)
- Product & Sprint Backlogs

- Kanban Boards
- Dashboards and Analytics
- Sprint planning and tracking

Purpose: Helps teams manage Agile projects and maintain visibility.

2. Azure Repos – Source Code Management

Azure Repos provides version control to store and manage source code.

Key Features:

- Git repositories
- Branching & merging
- Pull requests and code reviews
- Policies (build validation, reviewers, work item linking)
- TFVC support

Purpose: Ensures collaboration, code quality, and version history.

3. Azure Pipelines – CI/CD Automation

Azure Pipelines automates building, testing, and deploying applications.

Key Features:

- Continuous Integration (CI)
- Continuous Delivery (CD)
- Multi-platform support (Windows, Linux, Mac)
- YAML or Classic UI pipelines
- Integration with GitHub, Bitbucket, Docker, Kubernetes

Purpose: Enables fast, repeatable, and reliable software delivery.

4. Azure Test Plans – Testing & Quality Assurance

Azure Test Plans helps teams ensure quality through manual and automated testing.

Key Features:

- Manual and exploratory test management
- Test suites and test runs
- Bug tracking with detailed steps
- Traceability to work items and builds

Purpose: Improves product quality through structured testing.

5. Azure Artifacts – Package Management

Azure Artifacts is used to store and distribute packages internally.

Key Features:

- Supports Maven, npm, NuGet, Python packages
- Create and share private feeds

- Versioning and dependency management

  Purpose: Simplifies package sharing and dependency control.

Additional Supporting Features

◆ Boards → Repos → Pipelines → Test → Artifacts Integration

All components are highly integrated:

- Commits link to work items
- Pipelines build code from Repos
- Test Plans validate builds
- Artifacts used in deployments

◆ Extensions Marketplace

Supports 1,000+ add-ons such as Slack, SonarCloud, Docker, and ServiceNow.

◆ Azure DevOps Server

On-premises version for enterprise environments.

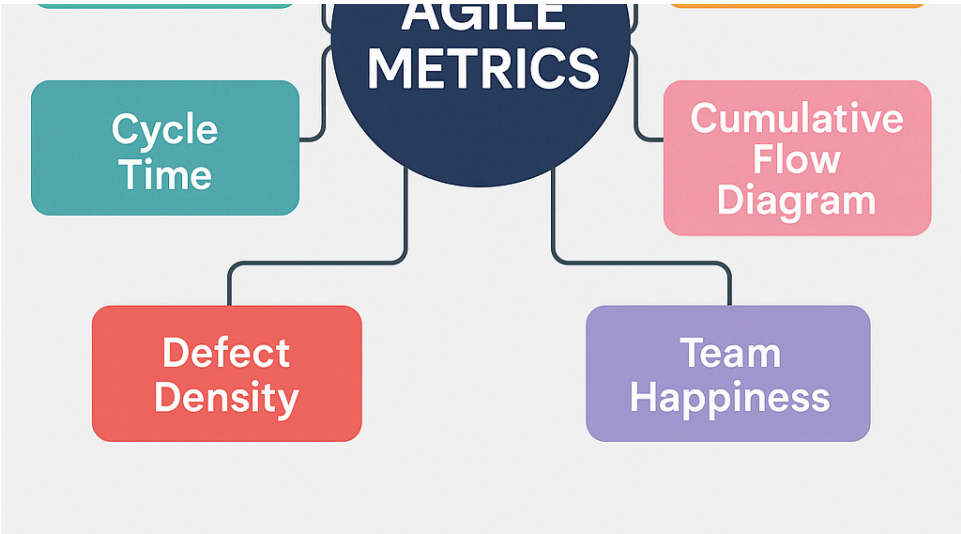
Summary Table – Fundamental Components

Component	Purpose
Azure Boards	Plan, track, and manage Agile work
Azure Repos	Source code management & version control
Azure Pipelines	CI/CD automation for build, test, deploy
Azure Test Plans	Manual & automated testing tools
Azure Artifacts	Package management & sharing

Q7) c) Explain Metrics in Agile Practice and Metrics for Project Management.
Ans.

Metrics in Agile Practice

Agile teams use metrics to measure progress, team performance, quality, and flow. These metrics support transparency and enable continuous improvement.



1. Velocity

- Measures the amount of work (in story points) completed in a sprint.
- Helps predict future sprint capacity and release timelines.

Example:

If a team completes 30 points in three consecutive sprints, their velocity is ~30.

2. Burndown Chart

- Shows remaining work over the sprint timeline.
- Helps track whether the team is on schedule.

Uses: Identifying scope creep, bottlenecks, or delays.

3. Burnup Chart

- Tracks completed work vs. total scope.
- Shows progress *and* scope changes more clearly than burndown charts.

4. Lead Time

- Time taken from when a work item is created to when it is completed.
- Indicates responsiveness and delivery speed.

5. Cycle Time

- Time taken from when work starts to when work is completed.
- Shorter cycle time = better flow efficiency.

6. Cumulative Flow Diagram (CFD)

- Shows work distribution across states (To Do, In Progress, Done).
- Helps detect bottlenecks (e.g., too much WIP in development or testing).

7. Defect Density / Quality Metrics

- Measures bugs relative to the size of delivered work.
- Used to assess code quality and testing effectiveness.

8. Team Happiness / Morale Metrics

- Agile emphasizes people over processes.
- Helps identify issues impacting productivity and collaboration.

Metrics for Project Management

Project management metrics track success in terms of schedule, budget, quality, scope, risk, and delivery effectiveness.

These metrics apply to **Agile, Waterfall, or Hybrid** projects.

🔹 1. Schedule Performance Metrics

Planned vs. Actual Progress

- Compares actual work completed against the project plan.

Schedule Variance (SV)

```
SV = Earned Value (EV) – Planned Value (PV)
```

Indicates if the project is behind or ahead of schedule.

🔹 2. Cost Performance Metrics

Budget Variance (BV)

Indicates whether spending is above or below the planned cost.

Cost Performance Index (CPI)

```
CPI = EV / Actual Cost (AC)
```

CPI < 1 means over budget.

🔹 3. Resource Utilization

Measures how effectively people, tools, or equipment are used.

Example: % of planned hours actually utilized.

🔹 4. Risk Metrics

- Number of open risks
- Severity & probability
- Risk exposure
- Mitigation effectiveness

Helps track and control uncertainties.

🔹 5. Scope Metrics

- Requirements completion rate
- Approved vs. implemented changes
- Scope creep percentage

Useful for change management.

🔹 6. Quality Metrics

- Defect leakage rate
- Test coverage percentage
- Rework hours

Indicates the overall quality of deliverables.

🔹 7. Stakeholder Satisfaction

Measured through surveys, reviews, and feedback loops.

Ensures that project outcomes meet business expectations.

🔹 8. Delivery Performance

- On-time delivery rate
- Number of releases per period
- Deployment success rate

Important for Agile and DevOps-driven environments.

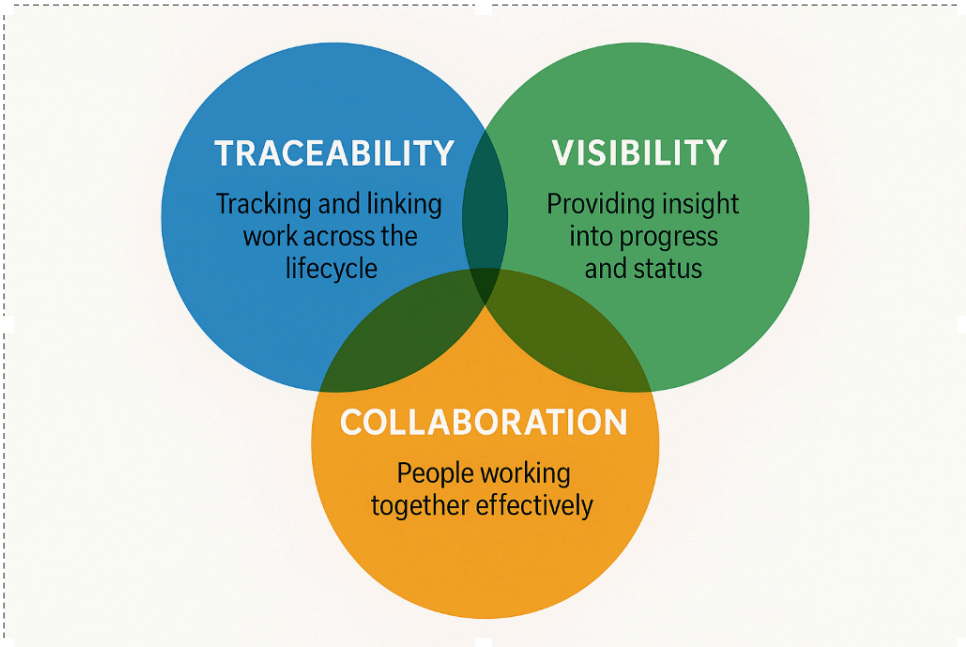
Summary Table

Agile Metrics	Purpose
Velocity	Predict sprint capacity
Burndown/Burnup	Track progress & scope
Lead Time	Time from request to delivery
Cycle Time	Efficiency of workflow
CFD	Identify bottlenecks
Defect Metrics	Measure quality
Team Morale	Improve collaboration

Project Management Metrics	Purpose
Schedule Metrics	Monitor timelines
Cost Metrics	Control budget

Resource Utilization	Optimize productivity
Risk Metrics	Minimize project risks
Scope Metrics	Manage change
Quality Metrics	Ensure deliverable quality
Stakeholder Satisfaction	Validate value delivery

Q8) b) Write a short note on: i) Traceability ii) Visibility iii) Collaboration
Ans.



i) Traceability

Definition:

Traceability is the ability to **track and link work across the entire lifecycle** of a project — from requirements → development → testing → deployment → maintenance.

Purpose:

- Ensures every requirement is implemented, tested, and delivered
- Helps identify the impact of changes
- Supports audits and compliance
- Connects work items to code, builds, and releases

Examples:

- A user story linked to its tasks, test cases, commits, and deployments
- Seeing which features or fixes are included in a specific build
- Tracking a defect back to the original requirement

Benefits:

✓ Improved quality

✓ Clear understanding of change impact
✓ Easier troubleshooting
✓ Full lifecycle accountability

ii) Visibility

Definition:

Visibility means providing **transparent access to real-time information** about progress, risks, quality, and workflow across the project.

Purpose:

- Allows teams and stakeholders to see what is happening
- Helps detect bottlenecks early
- Enables better decision-making

Examples:

- Dashboards showing burndown charts or sprint progress
- Kanban board showing tasks moving from To Do → In Progress → Done
- Build status (success/failure) displayed for everyone
- Release status and deployment history

Benefits:

✓ Informed decision-making

✓ Early detection of delays or issues
✓ Trust and transparency across teams

iii) Collaboration

Definition:

Collaboration is the process of **people working together** effectively to achieve shared goals using communication, coordination, and teamwork.

Purpose:

- Break down silos between development, testing, operations, and business teams
- Help teams share knowledge and resolve issues faster
- Improve communication during planning, execution, and review

Examples:

- Developers and testers working together on a user story
- Daily stand-up meetings
- Peer code reviews and pair programming
- Shared documentation and wikis
- Cross-functional Agile teams

Benefits:

✓ Faster problem resolution

✓ Higher-quality outputs
✓ Shared ownership and teamwork
✓ Improved innovation

⭐ Summary Table

Concept	Meaning	Purpose	Key Benefits
Traceability	Linking work across lifecycle (req → code → test → deploy)	Ensure completeness & accountability	Quality, impact analysis, audit readiness
Visibility	Making project status & data transparent	Detect issues early & enable decisions	Transparency, control, faster adjustments

Collaboration	Working together effectively	Break silos & increase coordination	Faster delivery, higher quality, teamwork