Unit 6: Applications of SPM in Industry (Q7 & Q8)

Rank	Question Topic	Occurrences	Max Marks	Paper References
1	**Application Lifecycle Management (ALM):** Define/Explain ALM tools; ALM phases/stages; Features of Azure DevOps supporting ALM.	6	7 22	, 17 , 23 , 24 , 25 , 26 , 27
2	**Azure DevOps Visibility & Boards:** What is visibility in DevOps? Ways to enable visibility; Explain Azure Boards.	5	6 22	, 23 , 24 , 25 , 27
3	**Metrics & Reports:** Explain metrics used for developer practices; Examples of reports for metrics in agile projects.	4	6 17	, 24 , 25 , 27
4	**Agile Project Management in Azure:** Explain Agile PM in Azure DevOps and TFS; Best practices; Methodologies supported.	4	7 22	, 20 , 27
5	**Azure DevOps Components:** Fundamental components; Branching and merging; Traceability and Collaboration.	2	10 20	, 27
6	**Azure Pipelines (CI/CD):** Explain how to use Azure Pipelines for CI/CD.	1	10 26	

Q1) Define Application Lifecycle Management (ALM) tools? What feature should be considered while choosing an ALM Tools List some examples of ALM tools?

1. Definition of Application Lifecycle Management (ALM) Tools 🛠️

Application Lifecycle Management (ALM) tools are integrated software systems that manage the entire lifespan of a software application, from its initial conception and requirements gathering through design, development, testing, deployment, and eventual maintenance and retirement.

ALM tools aim to **integrate and streamline** the traditionally separate functions of **governance, development, and operations** (DevOps) to improve collaboration, transparency, and product quality. They unify the three pillars of software development:

1. **Governance:** Requirements Management, Project Management, Change Control.
2. **Development:** Source Code Management, Build Management.

3. **Testing/Operations:** Quality Assurance, Release Management, Deployment, Monitoring.

2. Key Features to Consider When Choosing an ALM Tool

The effectiveness of an ALM tool depends on its ability to seamlessly connect all stages of the software lifecycle. Key featu res to evaluate include:

A. Core Integration and Traceability

End-to-End Traceability: The ability to link every item (code commit, test case, defect) back to its original **requirement**. This is crucial for audits and impact analysis.
Centralized Repository: A single source of truth for all project artifacts, including documents, code, and test results.

B. Requirements and Planning

Requirements Management: Tools for capturing, prioritizing, documenting, and versioning user needs (stories, use cases).
Project Management: Support for Agile frameworks (Scrum, Kanban) and traditional (Waterfall) methodologies, including task boards, sprint planning, and reporting.

C. Development and Quality Assurance (QA)

Source Code Management (SCM) Integration: Seamless linkage with external repositories (e.g., Git, Subversion) to associate code changes directly with tasks or defects.
Test Management: Functionality for creating, executing, and managing test cases (manual and automated), linking them to requirements, and logging defects.
Defect/Bug Tracking: Robust features for logging, prioritizing, assigning, and tracking the lifecycle of errors and issues.

D. Automation and Collaboration

Reporting and Dashboards: Customizable dashboards providing real-time visibility into project status, team velocity, quality metrics, and compliance.
Collaboration Features: Communication tools (wikis, comments, notifications) to facilitate continuous collaboration between developers, testers, and business analysts.
Automation/DevOps Integration: Capabilities for linking directly to Continuous Integration/Continuous Delivery (CI/CD) pipelines (e.g., Jenkins, GitLab CI) for automated building, testing, and deployment.

Q2) Explain application lifecycle management (ALM) process in detail.

The ALM process serves as the "thread" that ties together the improvement life cycle, covering requirements, development, testing, deployment, and ongoing maintenance.
Here is the ALM process explained in detail through its five fundamental phases:

1. Defining Requirements

This is the foundational stage where the project scope is established.

- **Stakeholder Agreement:** All stakeholders (business and technical) come together to agree on the application's design based on specific business needs.
- **Requirements Hierarchy:** Requirements are usually processed from the "top-down," meaning detailed needs are derived from higher-level business goals. This often results in a hierarchical tree where a general parent node breaks down into specific child nodes.
- **Result:** A clear list of requirements is defined, which serves as the blueprint for the development team.

2. Product Development

Once requirements are set, the actual creation of the software begins.

- **Execution:** The development team begins building the application based on the defined requirements. This phase involves coding, designing architecture, and implementing features.
- **Methodology:** This phase is often broken down into smaller pieces or iterations (especially in Agile environments) to manage complexity. It encompasses the traditional Software Development Life Cycle (SDLC) activities,.

3. Testing and Quality Assurance

This phase ensures the product works as intended and meets the initial requirements.

- **Overlap with Development:** Testing and Quality Assurance (QA) often overlap with the development phase. Testers start preparing their test cases and environments before the coding is fully complete.
- **Feedback Loop:** Testers provide feedback on the application during development. This includes integration and unit tests.
- **Verification:** The formal testing phase validates that the application meets the requirements defined in the first phase. If discrepancies or bugs are found, the software is sent back to development for fixing.

4. Deployment

This phase involves moving the application from the development/testing environment to the live production environment where users can access it.

- **Release:** The deployment phase involves releasing the product to users. This process varies depending on the application type (e.g., a web application might be deployed on a company's internal servers or cloud services).
- **Automation:** Modern ALM tools often automate this stage to ensure consistent and reliable releases.

5. Continuous Maintenance and Improvement

ALM does not end with deployment; it continues into the "Operations" phase.

- **Ongoing Support:** This involves monitoring the application, fixing bugs that arise after release, and ensuring the software remains functional.
- **Updates:** Providing regular updates and improvements based on user feedback and changing business requirements. This ensures the longevity and continued value of the application.

Key Aspects and Pillars of ALM

In addition to the sequential phases, the sources highlight that a robust ALM process relies on three distinct aspects and pillars:

Three Aspects of ALM Process:

1. **Governance:** Ensuring the project aligns with business needs and managing decision-making throughout the lifecycle.
2. **Development:** The actual lifecycle of creating the software (SDLC).
3. **Operations:** The work required to run and manage the application once it is deployed.

Three Pillars of ALM:

1. **Traceability:** The ability to track the relationship between requirements, code, tests, and releases. This ensures that every line of code can be traced back to a business requirement.
2. **Automation of High-Level Processes:** Automating workflows (like build and release) to reduce manual error and increase efficiency.
3. **Visibility:** Providing a clear view of project status (progress, quality, and direction) to all stakeholders, often through dashboards and reporting.

Q3) What are the key features of Azure DevOps that support ALM?

1. Azure Boards (Planning and Tracking)

Function: This service is used for Agile planning, work item tracking, visualization, and reporting.
Features: It allows teams to track work with Kanban boards, backlogs, team dashboards, and custom reporting. It supports Agile methodologies like Scrum and Kanban, enabling teams to plan, track, and discuss work across the entire development effort.

2. Azure Repos (Version Control)

Function: This component handles source control, which is essential for managing code changes during development.
Features: It provides cloud-hosted private Git repositories. It allows teams to collaborate on code development using pull requests and advanced file management.

3. Azure Pipelines (Build and Release)

Function: This feature supports the **Continuous Integration (CI)** and **Continuous Delivery (CD)** phases of ALM.
Features: It automatically builds and tests code projects to make them available to others. It works with any language, platform, and cloud. It automates development and deployment processes to ensure consistent software delivery.

4. Azure Test Plans (Quality Assurance)

Function: This service is dedicated to the testing phase of the application lifecycle.
Features: It provides an integrated testing solution that includes tools for manual and exploratory testing. It allows teams to test and ship with confidence.

5. Azure Artifacts (Package Management)

Function: It manages the dependencies and packages used in software development.
Features: It provides integrated package management with support for Maven, npm, Python, and NuGet packages, allowing teams to share code efficiently.

6. Additional Support Features

Dashboards: Azure DevOps provides highly customizable dashboards to share progress, status, and trends with stakeholders, ensuring visibility throughout the project.
Built-in Wiki: It includes a built-in wiki for sharing information, documentation, and knowledge within the team.
Traceability: A key aspect of ALM in Azure DevOps is traceability, which links requirements, code changes, and tests, ensuring that every piece of code can be traced back to a specific requirement or bug fix.

Q4) Explain Agile Project Management in Azure DevOps and TFS.

Agile Project Management in Azure DevOps and Team Foundation Server (TFS) involves using a suite of integrated tools to plan, track, and discuss work across software development teams.

Here is a detailed explanation of how Agile Project Management is structured and implemented within these platforms.

1. Relationship Between Azure DevOps and TFS

It is important to first clarify the relationship between the two systems:

- **Team Foundation Server (TFS):** This is the on-premise predecessor. It was a Microsoft product that provided source code management, reporting, requirements management, and project management.
- **Azure DevOps Server:** In 2019, TFS was rebranded as Azure DevOps Server. It allows the same code base and features as the cloud version but is hosted on-premise.
- **Azure DevOps:** This is the cloud-hosted, Software-as-a-Service (SaaS) platform. It provides the most current features for agile planning and collaboration.

2. Supported Methodologies

Azure DevOps supports several process templates to implement Agile practices. When creating a project, a manager can select from:

- **Agile:** Uses User Stories, Bugs, and Tasks.
- **Scrum:** Uses Product Backlog Items (PBIs), Bugs, and Tasks.
- **CMMI:** For more formal processes.
- **Basic:** A simplified model.

3. Key Components for Agile Management

The platform is divided into services that support the Agile lifecycle:

- **Azure Boards:** This is the primary tool for Agile project management. It offers Kanban boards, backlogs, team dashboards, and custom reporting to track work items.
- **Azure Repos:** Provides cloud-hosted Git repositories for version control, essential for collaborative coding in Agile teams.
- **Azure Pipelines:** Supports Continuous Integration (CI) and Continuous Delivery (CD) to automate building and testing, ensuring rapid feedback loops common in Agile.

4. Steps in Agile Project Management Process

The sources outline a specific workflow for managing an Agile project in Azure DevOps, often illustrated through a case study (e.g., an Online Shopping System):

A. Project Startup and Team Configuration

Creation: *The "Product Owner" or project manager initiates the project in the portal.*
Team Building: *A key step is "Creating new teams." Azure DevOps allows administrators to invite members, define roles, and manage permissions using security groups.*
Capacity Planning: *Managers can input the team's capacity (work hours per day) and days off to assist with sprint planning.*

B. Backlog Management

Work Items: *Work is broken down into "Work Items" (e.g., Epics, Features, User Stories). Each item is assigned a unique ID.*
Backlogs: *The Product Owner creates a backlog. For example, in a Scrum project, they list Product Backlog Items (PBIs) and prioritize them. Large items (Epics) are broken down into Features and then into PBIs.*

C. Sprint Planning

Velocity: *The team determines an "Initial Velocity" (how much work they can complete in a sprint) to guide how many items to pull from the backlog.*
Task Breakdown: *PBIs are broken down into specific tasks (e.g., "Design database," "Write code") estimated in hours.*
Assignment: *Tasks are assigned to team members, and the tool ensures the total work does not exceed the team's defined capacity.*

D. Execution and Tracking (Kanban & Scrum Boards)

Visualizing Work: *Teams use digital boards to move items through states such as "New," "Active/In Progress," "Resolved," and "Closed".*
Collaboration: *Azure DevOps provides wikis for documentation and discussion threads within work items to facilitate communication.*

5. Monitoring and Metrics

Azure DevOps provides real-time analytics to monitor Agile performance:

- **Sprint Burn-down Chart:** Tracks the remaining work in a sprint to predict if the goal will be met on time.
- **Velocity Chart:** Shows the amount of work completed in previous sprints to help forecast future performance.
- **Dashboards:** Customizable widgets provide visibility into project status, bugs, and code activity for all stakeholders.

Q5) Explain Metrics in Agile Practice and Metrics for Project Management.
The sources categorize these metrics into **Metrics in Agile Practice** (often focused on process and flow) and broader **Metrics for Project Management** (which can include financial and schedule performance). Additionally, there is a specific subset for **Developer Practices**.

1. Metrics in Agile Practice

Agile metrics are designed to provide visibility into the iterative process, helping teams understand their velocity and the remaining work. Common metrics include:

- **Sprint Burndown:** This metric tracks the remaining work in the sprint. It visualizes the relationship between the remaining work and the time left in the iteration, helping the team predict if they will finish the work on time.
- **Velocity:** This calculates the theoretical speed at which a team can work. It updates over time according to the team's actual velocity based on how much work they complete in each sprint. This helps in forecasting future sprints.
- **Release Burndown:** Similar to the sprint burndown, this tracks the progress toward a release goal over multiple sprints.
- **Backlog Overview:** This lists all user stories, filtered by tags and iterations, providing a snapshot of the work to be done.
- **Remaining Work:** A measure of the total hours or story points left to complete the planned tasks.

- **Unplanned Work:** This captures work added to an iteration after it has already started, highlighting disruptions or scope creep.

Reports for Agile Metrics: To visualize these metrics, Agile projects often use specific reports:

- **Bug Status Report:** Shows the number of bugs over time, categorized by their state (active, closed, resolved).
- **Reactivations Report:** Tracks bugs that were marked as fixed but were reopened.
- **Bug Trend Report:** Visualizes the rate of bug discovery versus bug resolution.

2. Metrics for Developer Practices

Within the scope of project management, specific technical metrics are used to ensure code quality and maintainability. These include:

- **Code Coverage:** Shows how much of the code base has been covered by automated unit tests. A value of 100% means every line of code is executed during testing, though typically a threshold (e.g., 80%) is set.
- **Code Metrics:** This includes measurements like **Cyclomatic Complexity** (indicating the complexity of the code logic) and **Maintainability Index**. These metrics warn teams if the code is becoming too difficult to maintain.
- **Compiler Warnings:** Tracks the number of warnings generated during the build process. Teams may set rules to fail a build if warnings exceed a certain limit.
- **Code Analysis:** Involves running static code analysis rules to identify potential issues or violations of coding standards.

3. Metrics for Project Management (Tracking and Control)

Broader project management metrics, often used in conjunction with Earned Value Management (EVM), focus on cost and schedule performance:

- **Earned Value (EV):** The value of the work that has been effectively completed so far.
- **Planned Value (PV):** The authorized budget assigned to the scheduled work.
- **Actual Cost (AC):** The total cost incurred for the work performed.
- **Schedule Variance (SV):** EV - PV. This indicates whether the project is ahead of or behind schedule.
- **Cost Variance (CV):** EV - AC. This indicates whether the project is under or over budget.
- **Schedule Performance Index (SPI):** EV / PV. An index less than 1 indicates the project is behind schedule.
- **Cost Performance Index (CPI):** EV / AC. An index less than 1 indicates the project is over budget.

Additionally, **Kanban boards** provide metrics related to workflow efficiency:

- **Lead Time:** The time that has passed between the request for an item and its delivery to the customer.
- **Cycle Time:** The time it takes to process a task from the moment work begins until it is completed.

Q6) Write a note on fundamental components of Azure DevOps.

Here is a note on the five fundamental components:

1. Azure Boards

• **Purpose:** This service is designed for planning, tracking, and discussing work across the entire development team.

• **Functionality:** It provides a rich set of capabilities including Kanban boards, backlogs, team dashboards, and custom reporting. It supports Agile methodologies (like Scrum and Kanban) to track User Stories, tasks, bugs, and features effectively.

• **Key Benefit:** It ensures visibility into the project's progress and helps manage work items throughout the development lifecycle.

2. Azure Repos

• **Purpose:** This component manages source control, providing a place to store and version software code.

• **Functionality:** It offers cloud-hosted private Git repositories. It supports standard Git features, allowing developers to collaborate on code through pull requests and advanced file management.

• **Key Benefit:** It is the place where teams determine the version control of their source codes, ensuring code safety and enabling collaborative development.

3. Azure Pipelines
• **Purpose:** This service handles **Continuous Integration (CI)** and **Continuous Delivery (CD)**.
• **Functionality:** It automates the building and testing of code projects to make them available to others. It works with any language, platform, or cloud.
• **Key Benefit:** It allows teams to build, test, and deploy code automatically to any target, facilitating rapid and reliable software delivery.

4. Azure Test Plans
• **Purpose:** This provides an integrated testing solution to ensure software quality.
• **Functionality:** It includes tools for manual and exploratory testing. It allows teams to plan and execute tests to identify defects before deployment.
• **Key Benefit:** It helps teams test and ship with confidence by providing a structured environment for managing test cases and tracking defects.

5. Azure Artifacts
• **Purpose:** This component is used for package management.
• **Functionality:** It allows teams to create, host, and share packages. It supports common package feed types such as Maven, npm, Python, and NuGet.
• **Key Benefit:** It enables the integrated sharing of code and dependencies across teams and pipelines, ensuring that the correct versions of libraries are used during builds.

---

q7) Explain best practices for Agile management.

Best practices for Agile management, particularly within the context of platforms like Azure DevOps, focus on leveraging integrated tools to maintain transparency, automate workflows, and ensure quality throughout the development lifecycle.

The following practices are recommended to effectively implement Agile management:

- **Utilize Azure Boards for Tracking:** Teams should use boards to track their backlog items systematically. This involves categorizing work based on a hierarchy such as Epics, Features, User Stories, and Tasks. This practice aids in preparing sprints and provides transparency regarding which items are scheduled for delivery in each sprint.
- **Implement Version Control with Azure Repos:** It is essential to use a centralized repository for source code management. This allows developers to control versions of their code, collaborate effectively, and merge their developments into shared branches.
- **Automate Delivery via Azure Pipelines:** A key best practice is setting up automated processes for building and deploying the product. Azure Pipelines facilitate **Continuous Integration (CI)** and **Continuous Delivery (CD)**, ensuring that code is automatically compiled, tested, and pushed to the server without manual intervention.
- **Ensure Quality with Azure Test Plans:** Teams should utilize rich testing tools to validate the application. This includes performing manual, exploratory, and continuous testing to ensure the software meets requirements before release.
- **Manage Dependencies using Azure Artifacts:** Efficiently managing software packages is critical. Azure Artifacts allows teams to easily package, share, and consume dependencies (such as Maven, NuGet, or Python packages), facilitating better code reuse and consistency across projects.

Monitoring Agile Metrics In addition to using the right tools, effective Agile management involves tracking specific metrics to assess team performance and project health:

- **Sprint Burndown:** Monitoring this chart helps the team visualize the remaining work in a sprint and predict if they will meet their goals on time.
- **Velocity:** Tracking velocity (the amount of work completed in a sprint) helps managers forecast how much work the team can handle in future iterations.
- **Cumulative Flow and Backlogs:** Keeping a close eye on the backlog overview and cumulative flow ensures that work is not piling up and that the team maintains a steady pace.

---

q8) Write a note on: i) Traceability ii) Visibility iii) Collaboration

i) Traceability

Traceability is one of the three vital pillars of traditional Application Lifecycle Management (ALM). It refers to the abilit y to track

and document the relationships between two or more phases of a development process, such as linking a business requirement to the specific lines of code written to implement it.

- **Importance:** Without traceability, organizations may struggle to maintain compliance with rules and regulations or fail to see how a bug fix might affect other parts of the system. It ensures that requirements can be followed all the way through architecture, design, tests, and deployment.
- **In Azure DevOps:** The platform supports traceability by linking **Work Items** (like user stories or bugs) to code changes (commits), builds, and tests. This allows teams to see exactly which requirement generated which code and which tests validate that code.
- In software project management, traceability ensures that every piece of the final software product—including lines of code, test cases, and logged defects—can be linked directly to a specific original **business requirement** or user story.
    - **Forward Traceability:** Links requirements to the resulting design, code, and test cases. It answers: "Which test cases were run to verify this requirement?"
    - **Backward (or Reverse) Traceability:** Links artifacts back to the original requirements. It answers: "Which requirement caused this piece of code or this defect to exist?"

ii) Visibility

Visibility is the ability for stakeholders and managers to see the status and progress of development efforts in real-time. It is considered the third pillar of ALM, addressing the common issue where managers have limited insight into project advancement until it is too late.

- **Mechanisms for Visibility:** Azure DevOps empowers visibility through three main features:
    - **Widgets and Dashboards:** Highly configurable visual displays that show the flexibility and status of information, tracking progress against the plan,.
    - **Queries:** These allow users to ask specific questions of the work item tracking service (e.g., "Show all active bugs") to monitor progress and patterns,.
    - **Power BI:** Integrating analytics to generate detailed reports and visual data representations for deeper insights.

  Benefits: It allows for **early detection of bottlenecks** and risks, enables data-driven decision-making, and builds trust by eliminating information silos between different groups (e.g., development, testing, and business).

iii) Collaboration

Collaboration is the process of individuals and teams working together effectively, sharing information, and coordinating their activities to achieve a common goal—the successful delivery of the software.

- In ALM, collaboration involves breaking down the traditional barriers between roles (e.g., Business Analyst, Developer, Tester, Operations) to create a single, unified team focused on product value.
    - **Mechanism:** ALM tools facilitate this by providing shared platforms for task assignment, commenting, status updates, and document sharing (like wikis or specifications). The integration of tasks and discussions within the same tool prevents key information from being lost in emails or separate systems.
    - **Goal:** To establish a **DevOps culture** where continuous communication and feedback loops accelerate the development and deployment cycle, resulting in faster time-to-market and better alignment with user feedback.

---

q9) Feature of Azure DevOps Project Management.

Integrated Project Management Features

These features focus on the administration, organization, and workflow management within the platform:

- **User Control:** Azure DevOps allows organizations to connect with **Azure Active Directory**. It enables administrators to add users from on-premise systems via Active Directory (AD) and invite guests. It facilitates restricting access for specific users or enabling restrictions based on directory policies.
- **Groups and Permissions:** The platform supports managing permissions through security groups. Administrators can add members to pertinent groups to organize team settings effectively.
- **Teams:** Projects can be split into specific teams to distribute work. This feature lets leaders set defaults for each team and manage team-specific settings.
- **Processes and Work Items:** Azure DevOps comes with standardized templates to manage projects. The four common work item templates offered are **Basic, Scrum, Agile, and CMMI**. These templates define how work items (like bugs, tasks, or features) are tracked.
- **Backlogs:** This is a core planning tool. It allows teams to plan, monitor, and organize their work items. There are generally three basic backlogs: the Portfolio backlog, Requirement backlog, and Iteration backlog.
- **Boards:** Boards offer an intuitive interface to manage and organize work. They visualize work items using cards and columns (Kanban style), making it easy to track progress.
- **Iteration Path:** This feature allows teams to assign work items to specific time intervals, commonly referred to as sprints, to manage the project schedule.
- **Areas:** This utility allows for breaking down work into a logical sequence, coordinating functional areas, and building websites for various business segments.
- **Notifications:** The system sends alerts to keep the team notified of every important task, ensuring everyone is aware of work assignments and changes.
- **Dashboards:** Users can construct custom dashboards to keep the team focused. These dashboards can display various widgets and metrics to monitor changes, build status, and work item trends.

Fundamental Service Components

In addition to the management features above, Azure DevOps is composed of five specific services that support the project lifecycle:

1. **Azure Boards:** Powerful work tracking with Kanban boards, backlogs, team dashboards, and custom reporting. It supports agile planning and visualization.
2. **Azure Repos:** Provides cloud-hosted private Git repositories for source control. It allows collaboration on code through pull requests and advanced file management.
3. **Azure Pipelines:** Supports **Continuous Integration (CI)** and **Continuous Delivery (CD)**. It works with any language, platform, and cloud to automate building, testing, and deploying code.
4. **Azure Test Plans:** An integrated testing solution providing tools for manual and exploratory testing to ensure software quality.
5. **Azure Artifacts:** Integrated package management that allows teams to create, host, and share packages (like Maven, npm, and NuGet) across teams and pipelines.

Additional Capabilities

- **Built-in Wiki:** The platform includes a built-in wiki for sharing information, documentation, and knowledge within the team.
- **Traceability:** It ensures traceability by linking requirements to code changes, builds, and tests, allowing for a clear audit trail throughout the development lifecycle.

---

q10) Which methodologies are supported by Azure DevOps server to implement Agile project practices?

The supported methodologies are:

1. **Basic:** This is likely a simplified model for teams that want to get started quickly with fewer strict rules.

2. **Agile:** This template is designed for teams using Agile planning methods, typically utilizing **User Stories**, **Bugs**, and **Tasks** to track work.

3. **Scrum:** This template supports the Scrum framework, focusing on **Product Backlog Items (PBIs)**, **Bugs**, and **Tasks** to manage work within sprints.

4. **CMMI (Capability Maturity Model Integration):** This template is used for teams that require a more formal process for improvement and rigorous change management.

These templates allow teams to choose the methodology that best fits their project management style, enabling them to plan, monitor, and organize work items such as portfolios, requirements, and iteration backlogs.

1. Scrum (Default and Most Popular) ⚙️

Scrum is the most widely used and often the default process model in Azure DevOps, aligning perfectly with its core features.

- **Focus:** Delivering working software in short, fixed-length **iterations** (sprints).
- **Work Items:** Uses **Product Backlog Items (PBIs)** and **Bugs** (for requirements) and **Tasks** (for execution).
- **Azure DevOps Support:** Provides built-in templates for **Sprint Planning**, **Velocity Charts**, **Sprint Burndown**, and **Capacity Planning** for teams. It natively supports the Scrum roles and artifacts.[1]

2. Agile (Microsoft's Flavor of Agile)

The "Agile" template in Azure DevOps implements a version of the Agile approach that is often seen as a hybrid, slightly less prescriptive than strict Scrum, but more adaptable to general business needs.

- **Focus:** Flexibility and continuous feedback, but with slightly more focus on tracking progress through the entire lifecycle, not just sprints.
- **Work Items:** Uses **User Stories** (for requirements), **Features** (for high-level requirements), and **Tasks** (for execution).[2]
- **Azure DevOps Support:** Excellent support for **Kanban boards** for continuous flow, **Cumulative Flow Diagrams (CFDs)**, and defining clear **State** transitions (e.g., New [3]$\rightarrow$ Active [4]$\rightarrow$ Resolved [5]$\rightarrow$ Closed) for work items.[6]

3. CMMI (Capability Maturity Model Integration) 📐

While CMMI is not strictly an Agile methodology, its implementation within Azure DevOps supports process control and change management, which are vital for **scaling Agile** or ensuring **governance** in large, regulated projects. Many organizations use this template alongside an Agile practice like Scrum.

- **Focus:** Formal process, rigorous documentation, and structured change management.
- **Work Items:** Uses **Requirements** (highly detailed specifications), **Change Requests**, and **Risks**.
- **Azure DevOps Support:** Provides structured forms and strict workflow rules ideal for traceability, audits, and formal sign-offs, which is crucial when implementing Agile in highly compliant environments.

Additional Methodology: Basic 🆕

The newer **Basic** process is a very lightweight option, best suited for smaller teams or proofs-of-concept that need minimal process overhead.[7] It primarily uses only three work item types: **Epics**, **Issues**, and **Tasks**. It supports simple Kanban boards and is the quickest way to start managing work.[8]

---

q11) What is visibility in devops ?Discuss three primary ways of enabling visibility in Azure DevOps.

What is Visibility in DevOps? 👀

In the context of DevOps, **visibility** is the practice of ensuring that all teams (Development, Operations, and business stakeholders) have **real-time, transparent access** to the metrics and status of the software delivery pipeline, from initial requirement to production operation.

Visibility is crucial because it breaks down **information silos**, enabling teams to quickly identify bottlenecks, anticipate risks, and make data-driven decisions. It allows teams to see the status of:

- **Flow:** How fast work (features, bugs) is moving through the system (e.g., lead time, cycle time).

- **Quality:** The health of the code and the product (e.g., test pass rates, defect density).
- **Health:** The stability and performance of the deployed application (e.g., uptime, latency).

Three Primary Ways of Enabling Visibility in Azure DevOps (ADO)

Azure DevOps offers multiple integrated tools to maximize visibility across the Application Lifecycle. The three primary methods you listed are:

1. Widgets and Dashboards 📊

Dashboards are personalized, central hubs that provide a high-level, real-time summary of project status using customizable **Widgets**.

- **Mechanism:** Users create dashboards and add various widgets that pull data from Boards, Pipelines, Repos, and Test Plans.
- **Visibility Enabled:** Provides a single-screen view of **project health** and **progress**. The focus is on instant, digestible metrics.
- **Examples of Widgets:**
    - **Sprint Burndown:** Shows progress toward the sprint goal and forecasts completion.
    - **Velocity:** Tracks the amount of work a team consistently completes per iteration.
    - **Query Tile/Chart:** Displays counts or charts based on custom work item queries (e.g., "Critical Bugs Opened Last Week").

2. Queries (Work Tracking) 🔎

Queries are fundamental to visibility as they allow users to retrieve specific, filtered subsets of data from the centralized work item database.

- **Mechanism:** Queries use **Work Item Query Language (WIQL)** to find and list work items (e.g., User Stories, Tasks, Bugs) based on fields, operators, and grouping clauses.
- **Visibility Enabled:** Provides **detailed, actionable insights** into the state of the backlog and team focus. Queries are the source data for many widgets and reports.
- **Examples:**
    - "Show all **User Stories** assigned to the current sprint where the **State** is 'New' and the **Priority** is 1."
    - "List all **Bugs** created by the QA team that have been inactive for more than 10 days."

3. Power BI Integration 📈

Power BI is Microsoft's business analytics tool that connects to Azure DevOps to provide sophisticated, advanced reporting and data visualization capabilities far beyond native ADO dashboards.

- **Mechanism:** ADO exposes its data via an **OData feed** (specifically the **Analytics Views** in ADO). Power BI connects to this feed to extract large datasets.
- **Visibility Enabled:** Allows for **cross-project analysis**, complex trend reporting, and the creation of visually rich, boardroom-ready reports. It is used for **strategic visibility** and forecasting.
- **Examples:**
    - Generating a multi-year report showing **Lead Time** trends across multiple projects or teams.
    - Creating a **forecast model** for when the current backlog will be completed based on historical team velocity and capacity.

---

q12) Explain the Customizing the Process Template in Azure DevOps

1. Standardized Base Templates

Before customizing, a project in Azure DevOps is initialized using one of the four standardized process templates. These templates define the default work item types and workflow rules:

- **Basic:** A simplified model for teams wanting to start quickly.
- **Agile:** Supports Agile planning methods (User Stories, Tasks, Bugs).
- **Scrum:** Designed for the Scrum framework (Product Backlog Items, Bugs, Tasks).
- **CMMI:** For teams requiring more formal processes and rigorous change management.

2. Customization Capabilities

While Azure DevOps provides these out-of-the-box templates, it supports **"Easy customization"** to ensure the tool aligns with the team's specific way of working. Unlike the on-premise Team Foundation Server (TFS) which allowed for extensive (and complex) XML-based customization, Azure DevOps offers a more streamlined, "Good" level of customization suitable for a SaaS environment.

3. Key Areas of Customization

Teams can modify the process template to suit their needs in several ways:

- **Work Items:** Teams can customize the **Work Items** (such as Bugs, Features, or User Stories). This includes adding custom fields to capture specific data points relevant to the organization.
- **Visualizing Data:** The system allows for the customization of how data is displayed. For instance, a team can configure reports or dashboards to show the status of all errors or specific metrics they care about, rather than just the defaults.
- **Workflow and Status:** Teams can adapt the workflow states (e.g., changing how a "Bug" moves from "Active" to "Closed") to match their internal approval or development processes.
- **Project Settings:** Customizations are typically managed through the **Project Settings** interface, where administrators can configure team permissions, notifications, and process details.

4. Purpose of Customization

The primary goal of customizing the process template is to ensure that the "system allows us to use one of the core components... to suit our needs". This flexibility ensures that the tool supports the team's unique development lifecycle rat her than forcing the team to adapt to the tool's rigid structure.

Q13) Explain any four metrics used for developer practices?

1. Code Coverage

• **Definition:** This metric shows how much of the code base has been covered by automated unit tests.

• **Purpose:** It helps determine the effectiveness of testing. The result is typically expressed as a percentage of code blocks or lines executed during testing.

• **Usage:** While achieving 100% coverage is difficult, teams often set a goal (e.g., 80% or higher) to ensure that a significant portion of the logic is tested. If the coverage is low, it serves as a warning that the team needs to discuss how to improve their testing strategy.

2. Code Metrics

• **Definition:** This involves measuring various quantitative aspects of the code, such as **lines of code**, **class coupling**, **inheritance depth**, **cyclomatic complexity**, and the **maintainability index**.

• **Purpose:** These metrics provide insight into the architecture, analysis, and design quality of the code.

• **Usage:** High complexity or deep inheritance trees can indicate that the code is becoming difficult to maintain or understand, prompting refactoring efforts.

3. Compiler Warnings

• **Definition:** This metric tracks the number of warnings generated by the compiler during the build process.

• **Purpose:** Errors and warnings should be avoided in the project. Allowing them to persist tends to lower the overall quality of the code base.

• **Usage:** Teams should monitor this metric to ensure the code remains clean. Ideally, the build policy should be enforced to fail the build if any errors or warnings exist, ensuring developers address them immediately.

4. Code Analysis

• **Definition:** This involves using development tools to perform static code analysis, which checks the code against a set of rules without executing it.

• **Purpose:** It helps developers identify potential issues related to design, globalization, interoperability, performance, and security.

• **Usage:** Code analysis tools provide warnings that indicate policy violations (e.g., naming conventions or security vulnerabilities). Many DevOps tools offer good support for code analysis functionality by writing sets of rules or suppressing rules that do not apply to the specific project.

---

q14) List any four examples of reports for metrics in agile projects?

1. **Burndown Chart** 📉
    a. **Metric Focus: Progress** toward a goal (Sprint or Release/Product).
    b. **What it shows:** Tracks the amount of **remaining work** (typically estimated in hours, story points, or task count) against the **time remaining** in an iteration or project period. The goal is for the line showing remaining work to "burn down" toward zero.
2. **Velocity Chart** 🚀
    a. **Metric Focus: Predictability** and **Capacity**.
    b. **What it shows:** The average amount of work (usually measured in **Story Points**) the team has successfully completed in the last few iterations. This metric is used to forecast how much work the team can reliably commit to in future sprints.
3. **Cumulative Flow Diagram (CFD)** 🌊
    a. **Metric Focus: Flow** and **Bottlenecks** (Kanban or continuous flow).
    b. **What it shows:** Plots the **cumulative number of work items** entering and leaving each workflow state (e.g., To Do, In Progress, Testing, Done) over time. Parallel lines indicate a stable process, while a widening gap between lines (like 'In Progress' and 'Done') signals a **bottleneck** or a buildup of work-in-progress (WIP).
4. **Defect/Bug Trend Report** 🐞
    a. **Metric Focus: Quality** and **Stability**.
    b. **What it shows:** Tracks the number of **new defects found** versus the number of **defects fixed/closed** over time (e.g., per sprint or week). A healthy trend shows the 'closed' line rising faster than the 'newly opened' line, indicating that the team is improving quality and paying down technical debt.

---

q15) What is Azure Borad? Explain with suitable example?

What is Azure Boards?

Azure Boards is a fundamental service within the Azure DevOps platform designed for **planning, tracking, and discussing work** across the entire development team. It serves as the primary interface for project management, allowing teams to manage their software projects through a rich set of capabilities including **Kanban boards, backlogs, team dashboards, and custom reporting**.

Key characteristics include:

• **Work Item Tracking:** It tracks distinct units of work called "Work Items" (e.g., User Stories, Bugs, Tasks, Features, or Epics). Every work item has a unique ID, allowing teams to follow its progress from creation to completion.

• **Agile Planning:** It supports Agile methodologies, such as Scrum and Kanban, enabling teams to prepare sprints, define deliverables, and monitor the flow of work.

• **Visual Management:** It provides highly interactive and visual tools (like cards on a board) to monitor the status of work items (e.g., Active, Resolved, Closed).

• **Traceability:** It ensures traceability by linking work items to code changes, builds, and tests, providing a clear history of how a requirement was implemented.

Suitable Example: Online Shopping System Case Study

The sources provide a case study of an **"Online Shopping System"** to illustrate how Azure Boards is used in a real-world scenario.

1. Project Setup and Backlog Creation: Imagine a team developing an Online Shopping System. The **Product Owner** uses Azure Boards to create a **Product Backlog**. They input specific requirements as **Product Backlog Items (PBIs)**.

• *Example Item:* A PBI might be "Create User Login Feature" or "Design Shopping Cart."

• In Azure Boards, the Product Owner organizes these items hierarchically. They might create an **Epic** (a large body of work) and break it down into smaller **Features** and **User Stories**.

2. Sprint Planning: The team creates a **Sprint Backlog** by selecting high-priority items from the Product Backlog to complete in the upcoming iteration (Sprint 1).

• Using the **Backlog tab** in Azure Boards, the team assigns specific PBIs to "Sprint 1."

• They can also use the **Capacity** tab to set the number of work hours available for each team member (e.g., Developer, Tester) to ensure they do not overcommit.

3. Execution and Tracking (The Board): Once the sprint begins, the team uses the **Kanban board** to track progress visually.

• A developer working on the "User Login" task moves the corresponding card from the **"To Do"** column to the **"In Progress"** column.

• Once the code is written and tested, the card is moved to **"Done"** or **"Closed"**.

• This visual flow ensures that everyone on the team knows the real-time status of the project and can identify bottlenecks immediately.

4. Reporting: Throughout the process, the team monitors metrics like the **Sprint Burndown Report** generated by Azure Boards. This chart shows the remaining work versus the time left in the sprint, helping the team predict if they will finish the "Online Shopping System" features on time.

---

q16) Explain the concept of branching and merging in Azure DevOps and how it is used to support Agile Project Management.

Concept of Branching and Merging

In Azure DevOps, **Azure Repos** provides cloud-hosted private Git repositories where the source code is stored and versioned. The concept operates as follows:

- **Branching:** Developers do not work directly on the main codebase (often called the `master` or `main` branch). Instead, they create **branches**, which are parallel versions of the code. This allows every team member to push their developments to their own isolated branch without affecting the stability of the primary project.
- **Merging:** Once a developer finishes their work on a branch (e.g., completing a specific feature or bug fix), the changes must be integrated back into the main codebase. This process is called **merging**. In Azure DevOps, this is typically managed through **Pull Requests**, where code is reviewed before it is merged (or rejected) into the master branch, which is kept ready to push to the server,.

Support for Agile Project Management

Branching and merging strategies in Azure DevOps directly support Agile practices in several ways:

1. **Parallel Development (Sprint Execution)** Agile teams often work on multiple **User Stories** or tasks simultaneously within a sprint. Branching allows different team members to work on different items (e.g., `feature/login`, `bugfix/cart-error`) at the same time without their code conflicting or breaking the application for others. This isolation ensures that incomplete work does not block the team's progress,.
2. **Traceability** Azure DevOps links code changes directly to **Work Items** (such as User Stories, Bugs, or Tasks defined in Azure Boards). When a developer creates a branch or completes a merge, these actions can be associated with specific requirements. This ensures **traceability**, allowing the team to see exactly which lines of code were written to implement a specific business requirement or fix a specific bug,.

3. **Collaboration and Quality Assurance** The merging process in Agile is often gated by **Pull Requests**. This fosters collaboration by requiring other team members to review the code before it is merged. This step ensures that the "Definition of Done" is met, coding standards are followed, and knowledge is shared across the team, effectively preventing bugs from reaching the production environment,.
4. **Continuous Integration (CI)** Branching and merging are the triggers for **Azure Pipelines**. When code is merged into the master branch, it can automatically trigger a build and test process (**Continuous Integration**). This automation ensures that the integrated work from the team remains in a releasable state, which is a core tenet of Agile delivery,.

---

q17) 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.

Azure Pipelines for CI/CD in Agile Projects

Azure Pipelines is a cloud service used to automatically build, test, and deploy code, enabling teams to implement **Continuous Integration (CI)** and **Continuous Delivery (CD)** efficiently. This automation facilitates working with any language or platform and connecting to various repositories like GitHub.

Implementing Continuous Integration (CI) The CI process in Azure Pipelines focuses on automating the merging and testing of code.

- **Build Automation:** CI is used to test and build automation for the project. It automatically compiles the code and ensures that broken code is not sent to the shared repository.
- **Testing:** The system runs tests continuously to catch bugs or issues early in the development cycle.
- **Artifact Creation:** Successful builds produce "artifacts" (deployable units), which are then ready for the next stage of the lifecycle.

Implementing Continuous Delivery (CD) The CD process follows CI to automate the deployment of the software.

- **Automated Deployment:** CD involves automatically deploying and testing code in several stages to increase quality. It pushes the application to various infrastructure environments or directly to production.
- **Release Pipelines:** The system takes the "tested code" generated from the CI process and pushes it to the server. The goal is to ensure the software is always in a deployable state, often utilizing "release pipelines" to manage the flow of artifacts to end-users.

Managing Work Items in Azure Boards

Azure Boards serves as the hub for planning, tracking, and discussing work across the entire Agile team. It manages **Work Items**, which are distinct units of work (such as features, bugs, or user stories) that possess unique IDs.

There are several distinct ways to manage these work items:

1. Backlogs and Hierarchy Teams manage work items using a hierarchical structure to organize the scope of the project.

- **Portfolio and Requirement Backlogs:** The backlog allows teams to plan and prioritize work. Items are often structured hierarchically, starting with **Epics**, breaking down into **Features**, and further into **Product Backlog Items (PBIs)** or **User Stories**.
- **Prioritization:** The Product Owner uses the backlog to list stories and prioritize them based on business value or urgency before moving them into active sprints.

2. Kanban Boards Azure Boards visualizes work items using **Kanban boards**, which provide a graphical representation of the workflow.

- **Visual Signals:** Work items appear as cards on the board.

- **Columns and Status:** Cards move through various columns representing the workflow state, such as "To Do," "In Progress," and "Complete".
- **WIP Limits:** Teams can set Work In Progress (WIP) limits on columns to prevent bottlenecks and ensure focus.

3. Sprint Planning (Iterations) For teams using Scrum, work items are managed through **Iterations** or **Sprint Backlogs**.

- **Sprint Assignment:** Work items are assigned to specific time intervals (sprints).
- **Capacity Planning:** Teams can manage work items based on team capacity. The "Work By" tab allows leads to see how much work is assigned to each member relative to their available capacity.

4. Traceability and Linking A critical management feature is **traceability**. Work items are not isolated; they are linked to other artifacts in the development lifecycle.

- **Linking:** Azure DevOps allows users to link work items to code changes, specific builds, and test results. This ensures that for every piece of code or bug fix, there is a corresponding work item tracking the requirement.