***UNIT 6 (SPM)***

1. ***Application Lifecycle Management (ALM)***

**Application Lifecycle Management (ALM):**

**Definition:** Application Lifecycle Management (ALM) is an integrated process that manages a software application throughout its entire lifecycle, from initial planning to development, deployment, maintenance, and eventual decommissioning. It involves people, tools, and processes to ensure the efficient management and quality of software applications.

**Key Components of ALM:**

1. **Initial Planning and Requirement Gathering:**
   * Defines what the application needs to achieve.
   * Sets goals, compliance requirements, and design considerations.
2. **Development:**
   * The process of coding, building, and creating the application.
   * Involves breaking down the application into phases or modules.
3. **Testing and Quality Assurance:**
   * Verifies if the application meets predefined requirements.
   * Includes unit testing, integration testing, and quality checks.
4. **Deployment:**
   * Releases the application to the intended environment (e.g., cloud, on-premises).
   * Ensures users can interact with it effectively.
5. **Maintenance and Updates:**
   * Provides ongoing support to resolve bugs.
   * Adds new features and ensures compatibility with changing technologies.
6. **Decommissioning:**
   * Ends the application's lifecycle when it is no longer needed or replaced by another.

**Importance of ALM:**

1. **Improves Collaboration:**
   * Provides a standardized platform for communication between development, operations, and other teams.
2. **Enhances Quality:**
   * Integrates frequent testing and feedback mechanisms for quality assurance.
3. **Optimizes Productivity:**
   * Automates processes to save time and resources.
4. **Compliance and Governance:**
   * Ensures adherence to industry standards and regulations.

**Summary:**

* ALM is a structured process to manage a software application throughout its lifecycle.
* It involves **planning**, **development**, **testing**, **deployment**, **maintenance**, and **decommissioning**.
* Ensures quality, productivity, compliance, and collaboration.

1. ***Overview of Azure DevOps***

**What is Azure DevOps?**

**Azure DevOps is a cloud-based Software-as-a-Service (SaaS) platform by Microsoft. It provides tools and services to support DevOps practices across the entire software lifecycle—from planning to development, testing, deployment, and maintenance. It allows teams to collaborate, automate workflows, and deliver high-quality applications faster.**

**Key Features of Azure DevOps:**

1. **Git Repositories:**
   * **Version control for code using Git or Team Foundation Version Control (TFVC).**
   * **Helps manage and track code changes.**
2. **Azure Pipelines:**
   * **Supports Continuous Integration (CI) and Continuous Delivery (CD).**
   * **Automates building, testing, and deploying code across various platforms.**
3. **Azure Boards:**
   * **Work tracking and project management tools using Agile, Scrum, and Kanban methods.**
   * **Provides dashboards, backlogs, and custom reporting.**
4. **Azure Test Plans:**
   * **Integrated testing solutions for manual, exploratory, and automated tests.**
   * **Ensures application quality and stability.**
5. **Azure Repos:**
   * **Cloud-hosted private Git repositories.**
   * **Enables team collaboration on code.**
6. **Azure Artifacts:**
   * **Package management for Maven, NuGet, npm, and Python.**
   * **Helps share libraries and packages across teams.**

**Why is Azure DevOps Important?**

1. **End-to-End Integration: It combines all tools needed for the software lifecycle in one platform.**
2. **Collaboration: Enables developers, testers, and operations teams to work together seamlessly.**
3. **Customization: Highly customizable workflows and dashboards to meet team-specific needs.**
4. **Scalability: Suitable for small startups to large enterprises.**
5. **Supports Multi-Platform Development: Works with various programming languages, frameworks, and cloud providers.**

**Example:**

**Imagine a company developing an e-commerce website:**

1. **Developers use Azure Repos to collaborate on the codebase.**
2. **They set up Azure Pipelines to build and test the code automatically after every update.**
3. **Project managers use Azure Boards to track progress, assign tasks, and manage backlogs.**
4. **Testers use Azure Test Plans to validate features and ensure the site is bug-free.**
5. **Teams use Azure Artifacts to share reusable packages like payment integration libraries.**

**Benefits of Azure DevOps:**

* **Accelerated Delivery: Automation speeds up releases.**
* **Improved Quality: Continuous testing ensures high standards.**
* **Simplified Collaboration: Teams can work effectively regardless of location.**
* **Cost-Effective: Reduces the need for multiple tools by providing an all-in-one solution.**

**Summary (Point-to-Point):**

1. **What is Azure DevOps?: A cloud-based platform for managing the software lifecycle.**
2. **Key Features:** 
   * **Git Repos for version control.**
   * **Pipelines for CI/CD automation.**
   * **Boards for project management.**
   * **Test Plans for quality assurance.**
   * **Artifacts for package management.**
3. **Importance: Integrates tools, fosters collaboration, and supports scalability.**
4. **Example: Managing an e-commerce project using Azure DevOps features.**
5. **Benefits: Faster delivery, better quality, effective collaboration, and cost efficiency.**
6. ***Traceability, Visibility, and Collaboration in Azure DevOps***

**Traceability, Visibility, and Collaboration in Azure DevOps**

**1. Traceability**

**Definition:  
Traceability in Azure DevOps refers to the ability to track and link all elements of the development process, such as work items, code changes, builds, and test results. It ensures that every part of the project can be traced back to its origin or forward to its impact.**

**Key Features of Traceability in Azure DevOps:**

* **Tracks work items through all phases of the lifecycle.**
* **Links code commits to specific requirements or tasks.**
* **Allows backward and forward tracking of issues and changes.**

**Example:  
A developer adds a new feature based on a work item. With traceability, the work item, code changes, and related test cases are all linked. If a bug arises, it can be traced back to the specific feature and code changes.**

**Advantages:**

* **Reduces ambiguity by linking requirements, tasks, and outcomes.**
* **Simplifies audits and compliance checks.**

**2. Visibility**

**Definition:  
Visibility in Azure DevOps refers to the ability to share and access information about the project’s progress, performance, and status across the team or organization.**

**Key Features of Visibility in Azure DevOps:**

* **Dashboards: Visual representation of project metrics, trends, and progress.**
* **Queries: Custom queries to track specific work items or bugs.**
* **Reports: Real-time insights into team performance, sprints, and releases.**

**Example:  
Project managers can view a dashboard showing sprint progress, active bugs, and resolved issues. They can use this data to identify bottlenecks and adjust priorities.**

**Advantages:**

* **Enhances decision-making with real-time data.**
* **Keeps stakeholders informed and aligned.**

**3. Collaboration**

**Definition:  
Collaboration in Azure DevOps refers to enabling teams to work together effectively by providing tools for communication, shared workflows, and task management.**

**Key Features of Collaboration in Azure DevOps:**

* **Work Item Discussion: Comments and discussions on specific tasks or bugs.**
* **Integrated Tools: Boards, pipelines, and repos for shared workflows.**
* **Wiki: Centralized knowledge-sharing platform for the team.**

**Example:  
Team members discuss a bug on a work item, update its status, and collaborate on a fix. Documentation for the fix is added to the project wiki for future reference.**

**Advantages:**

* **Breaks silos by connecting developers, testers, and project managers.**
* **Ensures smooth communication, even in distributed teams.**

**Example Scenario**

**A project manager assigns a task to a developer using Azure Boards. The developer commits code referencing the task ID, ensuring traceability. The task's progress is visible on the project dashboard. Later, the team collaborates on fixing a related bug by discussing it directly on the work item.**

**Summary (Point-to-Point):**

1. **Traceability:**
   * **Tracks work items, code, and tests throughout the lifecycle.**
   * **Example: Linking a bug to its code change and test case.**
2. **Visibility:**
   * **Provides dashboards, queries, and reports for project insights.**
   * **Example: Tracking sprint progress with real-time dashboards.**
3. **Collaboration:**
   * **Facilitates teamwork through discussions, wikis, and shared tools.**
   * **Example: Team members resolving a bug together via work item comments.**

**Let me know if you'd like to dive deeper into any of these! 😊**

1. ***Explain Azure Pipeline with diagram.***

**Azure Pipelines Explained**

**What is Azure Pipelines?**

**Azure Pipelines is a tool within Azure DevOps that automates the build, test, and deployment of your code. It supports Continuous Integration (CI) and Continuous Delivery (CD), ensuring your application is always ready to deploy to any target.**

**Key Concepts:**

1. **Continuous Integration (CI):**
   * **Automatically builds and tests your code whenever changes are pushed to the repository.**
   * **Ensures the new code integrates smoothly with existing code.**
   * **Example: Developers push new features to Git; Azure Pipelines builds and tests them immediately.**
2. **Continuous Delivery (CD):**
   * **Automates the deployment of the application after a successful build.**
   * **Deploys the code to the environment (e.g., staging, production).**
   * **Example: After passing all tests, the pipeline deploys a new app version to production.**

**How Does Azure Pipelines Work?**

1. **Source Code Repository: Azure Pipelines fetches code from GitHub, Azure Repos, or another version control system.**
2. **Build Phase: Compiles the code, runs unit tests, and packages the application.**
3. **Testing Phase: Runs integration and performance tests to verify quality.**
4. **Deployment Phase: Deploys the tested application to staging or production environments.**

**Advantages of Azure Pipelines:**

1. **Multi-Platform Support: Works with any language (e.g., Java, Python, .NET).**
2. **Cloud Integration: Deploys to any cloud provider, including Azure and AWS.**
3. **Scalability: Handles large-scale projects efficiently.**
4. **Open Source Friendly: Free CI/CD for open-source projects on GitHub.**

**Example:**

**A software team uses Azure Pipelines to manage a fitness app:**

1. **Developers push updates to GitHub.**
2. **The pipeline builds the app and runs unit tests.**
3. **Integration tests check compatibility with APIs.**
4. **The app is deployed to staging for user testing.**
5. **Once approved, it goes live on production servers.**

**Summary (Point-to-Point):**

1. **Definition: Automates build, test, and deployment processes.**
2. **Key Concepts: CI for integration, CD for deployment.**
3. **Workflow: Fetch code → Build → Test → Deploy.**
4. **Advantages: Multi-platform, scalable, cloud-friendly, and open-source support.**
5. **Example: Automating a fitness app’s CI/CD pipeline.**
6. ***Difference between TFS and Azure DevOps***

**Difference Between Microsoft TFS and Azure DevOps**

**Overview:**

**Microsoft Team Foundation Server (TFS) and Azure DevOps are both platforms for managing the software development lifecycle. However, Azure DevOps is the next evolution of TFS, providing enhanced features, flexibility, and cloud support.**

| **Feature** | **TFS (Team Foundation Server)** | **Azure DevOps** |
| --- | --- | --- |
| **Deployment** | **On-premises only. Requires setup and maintenance on local servers.** | **Cloud-based (SaaS) by default with on-premises option (Azure DevOps Server).** |
| **Ease of Use** | **More complex to install and manage.** | **Simplified setup with minimal maintenance for cloud users.** |
| **Updates** | **Requires manual updates and patching.** | **Automatic updates in the cloud; always uses the latest version.** |
| **Flexibility** | **Limited integration with modern tools and services.** | **Highly extensible with integrations for third-party tools and APIs.** |
| **Collaboration** | **Basic collaboration tools like work item tracking.** | **Advanced collaboration with boards, test plans, pipelines, and wikis.** |
| **Continuous Integration** | **Limited CI/CD capabilities.** | **Fully integrated CI/CD pipelines with robust automation.** |
| **Cost** | **Higher upfront cost for licenses and server maintenance.** | **Pay-as-you-go pricing for cloud services; no upfront cost.** |
| **Extensibility** | **Limited extensibility options.** | **Extensive marketplace with pre-built extensions and custom add-ons.** |
| **Version Control** | **Supports TFVC and Git but less focus on Git.** | **Full Git support with advanced features like pull requests and branching.** |
| **Global Accessibility** | **Limited to local network unless exposed manually via VPNs or the internet.** | **Accessible anywhere with an internet connection.** |
| **Dashboards and Reports** | **Basic dashboards with fewer customization options.** | **Highly customizable dashboards and reporting tools.** |

**Advantages of Azure DevOps Over TFS:**

1. **Cloud-First Approach: Azure DevOps is designed for modern cloud-centric development.**
2. **Better Collaboration: Supports advanced project management tools like boards, wikis, and test plans.**
3. **Integrated CI/CD: Seamless pipelines for building, testing, and deploying applications.**
4. **Frequent Updates: Cloud-based nature ensures automatic updates with new features.**
5. **Extensibility: Integrates with numerous tools (e.g., GitHub, Jenkins) via a robust marketplace.**

**Example Scenario:**

1. **With TFS:** 
   * **A team sets up TFS on a local server, manually configures version control, and performs builds. Updates require downtime and manual installation.**
2. **With Azure DevOps:** 
   * **The team uses Azure DevOps in the cloud, integrates GitHub for source control, and automates builds and deployments using Azure Pipelines. No server maintenance is needed, and new features roll out automatically.**

**Summary (Point-to-Point):**

1. **Deployment: TFS is on-premises, while Azure DevOps is cloud-based.**
2. **Ease of Use: Azure DevOps is easier to set up and maintain.**
3. **Updates: TFS requires manual updates; Azure DevOps updates automatically.**
4. **CI/CD: Limited in TFS, robust in Azure DevOps.**
5. **Collaboration: Azure DevOps offers advanced tools like boards and wikis.**
6. **Cost: Azure DevOps is more cost-effective for cloud users.**

**Let me know if you'd like more details on any point! 😊**

1. ***Metrics in Agile Practice***

**Metrics in Agile practice are measurable indicators used to track the progress, performance, and quality of a project. These metrics help teams understand how efficiently they are working, identify potential bottlenecks, and ensure they meet their goals.**

**1. Backlog Overview**

* **The backlog is a list of all tasks or features your team needs to work on.**
* **It helps the team decide what to do first and what can wait.**
* **Example: Think of it like a grocery shopping list. You decide what to buy now (urgent items) and what can wait until later.**

**2. Sprint Burndown**

* **This is a graph that shows how much work is left during a sprint (a short, fixed time to complete tasks).**
* **It helps the team check if they are working at the right speed to finish all tasks in time.**
* **Example: Imagine you have to clean 10 rooms in a week. Each day, you see how many are left to clean and adjust your pace if needed.**

**3. Velocity Report**

* **Velocity tells how much work the team completed in past sprints. This helps plan how much work to take in the next sprint.**
* **It’s like knowing how fast you run a kilometre. If you consistently run in 10 minutes, you can plan your next run accordingly.**
* **Example: If your team completes 20 tasks every sprint, you know you can commit to 20 tasks for the next one.**

**4. Release Burndown**

* **This is a graph showing progress towards the final goal or product release.**
* **It helps check if the project will be finished on time.**
* **Example: If you’re painting a house in 3 months, this chart shows how many rooms are painted versus how many are left.**

**5. Remaining Work**

* **It shows how many tasks are still left to complete during a sprint.**
* **This helps the team focus and understand if they’re on track.**
* **Example: If you planned to bake 10 cakes and 3 are left, you know you’re almost done!**

**6. Unplanned Work**

* **These are tasks added after the sprint starts. They are not planned but are important enough to include.**
* **Too much unplanned work can slow down the team.**
* **Example: While baking cakes, you notice a tray broke, and you need to fix it before continuing. Fixing the tray is unplanned work.**

**Summary (Point to Point):**

1. **Backlog Overview: A list of all tasks, like a to-do list, to prioritise work.**
2. **Sprint Burndown: A graph showing how much work is left in the current sprint.**
3. **Velocity Report: Measures how much work the team can handle based on past sprints.**
4. **Release Burndown: Tracks progress for the overall project deadline.**
5. **Remaining Work: Focuses on what’s still incomplete during the sprint.**
6. **Unplanned Work: Unexpected tasks added during the sprint.**

**Example for All: In a project to develop a website:**

* **Backlog: Tasks like "Create homepage," "Add contact page."**
* **Sprint Burndown: Tracks progress of these tasks during the sprint.**
* **Velocity Report: Shows how much the team completed in previous sprints.**
* **Release Burndown: Tracks overall progress for launching the website.**
* **Remaining Work: Highlights tasks still to be done in the current sprint.**
* **Unplanned Work: Fixing an urgent bug found while creating the homepage.**

1. ***Metrics for Project Management***

Good metrics help measure progress, identify problems, and ensure project goals are met. These metrics fall into four categories:

**1. Agile Metrics**

* Focused on tracking team performance and project progress.
* **Key Metrics**:
  1. **Backlog Overview**: A list of tasks or features to be completed.
  2. **Sprint Burndown**: A graph showing remaining work in a sprint.
  3. **Velocity Report**: Tracks how much work a team completes in each sprint.
  4. **Release Burndown**: Tracks progress towards a full product release.
  5. **Remaining Work**: Tasks that are still pending.
  6. **Unplanned Work**: Tracks additional tasks added during the sprint.

**2. Metrics for Architecture, Analysis, and Design**

* These metrics evaluate the system’s structure, complexity, and maintainability.
* **Key Metrics**:
  1. **Lines of Code**: A high count can mean the code is too complex and hard to maintain.
  2. **Class Dependency / Class Coupling**: Measures dependencies between classes; low coupling is better for flexibility and reuse.
  3. **Depth of Inheritance**: Tracks the hierarchy of class definitions; deeper hierarchies are harder to manage.
  4. **Code Path Complexity / Cyclomatic Complexity**: Measures the number of code paths; high complexity makes maintenance and testing harder.
  5. **Maintainability Index**: A score from 0 to 100 that shows how easy it is to maintain the code (above 60 is good).

**3. Metrics for Developer Practices**

* Focused on measuring the quality of coding practices.
* **Key Metrics**:
  1. **Code Coverage**: Tracks how much of the code is tested; shown as a percentage of the total codebase.
  2. **Code Metrics**: Includes lines of code, class coupling, inheritance depth, and maintainability index.
  3. **Compiler Warnings / Code Errors and Alerts**: Errors or warnings in the code should be minimized to maintain quality.
  4. **Code Analysis Warnings**: Tools highlight issues related to design, performance, security, and more.

**4. Metrics for Software Testing**

* These metrics measure how effective the testing process is.
* **Key Metrics**:
  1. **Number of Bugs per State**: Tracks how many bugs are active, resolved, or closed.
  2. **Code Coverage**: Measures the percentage of code tested by automated tests.
  3. **Test Run Results**: Tracks how many tests pass or fail.
  4. **Percentage of Requirements Covered by Test Cases**: Shows how many requirements are verified by test cases.

**Summary (Point to Point):**

1. **Agile Metrics**:
   * Backlog, Sprint/Release Burndown, Velocity, Remaining, and Unplanned Work.
2. **Metrics for Architecture, Analysis, and Design**:
   * Lines of Code, Class Coupling, Depth of Inheritance, Cyclomatic Complexity, Maintainability Index.
3. **Metrics for Developer Practices**:
   * Code Coverage, Code Metrics, Compiler Warnings, Code Analysis Warnings.
4. **Metrics for Software Testing**:
   * Bugs per State, Code Coverage, Test Run Results, Requirement Coverage.

**Example:**

In a project to develop a banking app:

* **Agile Metrics**: A sprint burndown chart shows 50% of the tasks are done midway through the sprint.
* **Architecture Metrics**: Cyclomatic complexity is high, indicating the need to simplify the code.
* **Developer Practices**: Code coverage is 85%, showing good testing practices.
* **Testing Metrics**: Bug reports reveal 10 active bugs, with 5 already resolved.

1. ***Which methodologies are supported by Azure DevOps***

Azure DevOps Server supports various methodologies to implement Agile project practices. It primarily supports:

1. **Scrum**:
   * Used for managing iterative and incremental projects.
   * It focuses on sprints, roles like Product Owner, Scrum Master, and development teams, and emphasizes a commitment to achieving sprint goals.
2. **Kanban**:
   * A visual system to manage workflows.
   * Helps track work items, reduces bottlenecks, and ensures continuous delivery.
3. **Agile Framework**:
   * Incorporates flexible planning, continuous improvement, and swift responses to change.
4. **Hybrid Models**:
   * Combines elements of Scrum and Kanban (Scrumban) or integrates Agile with traditional models for a custom approach.

**Features in Azure DevOps that support Agile methodologies:**

* **Azure Boards**: Offers tools like Kanban boards, backlogs, and sprint tracking to manage Agile workflows effectively.
* **Dashboards and Reporting**: Provide real-time insights into progress, productivity, and bottlenecks.
* **Work Item Tracking**: Tracks tasks, bugs, and user stories through different project stages.

**Summary:**

1. **Supported Methodologies**: Scrum, Kanban, Agile Framework, Hybrid Models.
2. **Tools Provided**:
   * **Azure Boards**: Manage sprints and workflows.
   * **Dashboards**: Real-time reporting.
   * **Work Item Tracking**: Follow task progress.
3. **Flexibility**: Combines methodologies for tailored project management.

**Example**: In an e-commerce app project, a team uses:

* Scrum for sprint-based development.
* Kanban to visualize and streamline the bug-fixing process.

1. ***List any four examples of reports for metrics in agile projects?***
2. **Bug Status Reports**:
   * Tracks the status of bugs (active, resolved, or closed).
   * Helps identify whether bugs are increasing, decreasing, or being resolved effectively.
   * **Example**: If active bugs are consistently decreasing while resolved bugs are increasing, it indicates effective testing and debugging.
3. **Reactivations Reports**:
   * Tracks the number of bugs that were reopened after being marked as resolved or closed.
   * A high number suggests issues with the quality of bug fixes or insufficient testing.
   * **Example**: If 20% of resolved bugs are reopened, it indicates the need to improve the debugging process.
4. **Bug Trend Reports**:
   * Monitors the rate at which bugs are reported, resolved, and closed over time.
   * Useful for tracking progress and identifying any recurring issues.
   * **Example**: A report showing a steady decrease in new bugs while resolved bugs increase signifies project stability.
5. **Test Run Results Reports**:
   * Tracks the results of test cases (passed, failed, or skipped).
   * Helps identify problem areas in the code and assess test coverage.
   * **Example**: If 90% of test cases pass and 10% fail, focus should be on the failing areas for improvement.

**Summary (Point to Point):**

1. **Bug Status Reports**: Tracks active, resolved, and closed bugs to monitor overall project health.
2. **Reactivations Reports**: Tracks bugs reopened due to incomplete or incorrect fixes.
3. **Bug Trend Reports**: Shows bug trends over time to assess stability and identify recurring issues.
4. **Test Run Results Reports**: Monitors test outcomes to identify problematic areas in the code.

**Example for All**:

* A project has 100 active bugs, 50 resolved bugs, and 20 reactivations. Bug trends show a decrease in new bugs, while test runs reveal 85% of cases passed. These reports indicate the team is progressing but needs to improve debugging quality【20:0†source】.