UNIT I – INTRODUCTION TO SOFTWARE ENGINEERING [9 hours]

Definition of Software Engineering, Software Development Life Cycle (SDLC) – Phases,Traditional vs Agile Models (Waterfall, Agile, DevOps),Scrum Basics – Roles, Sprint, Backlog,Version Control using Git and GitHub,Introduction to Project Tools (GitHub Projects, Jira, Trello)

SCRUM

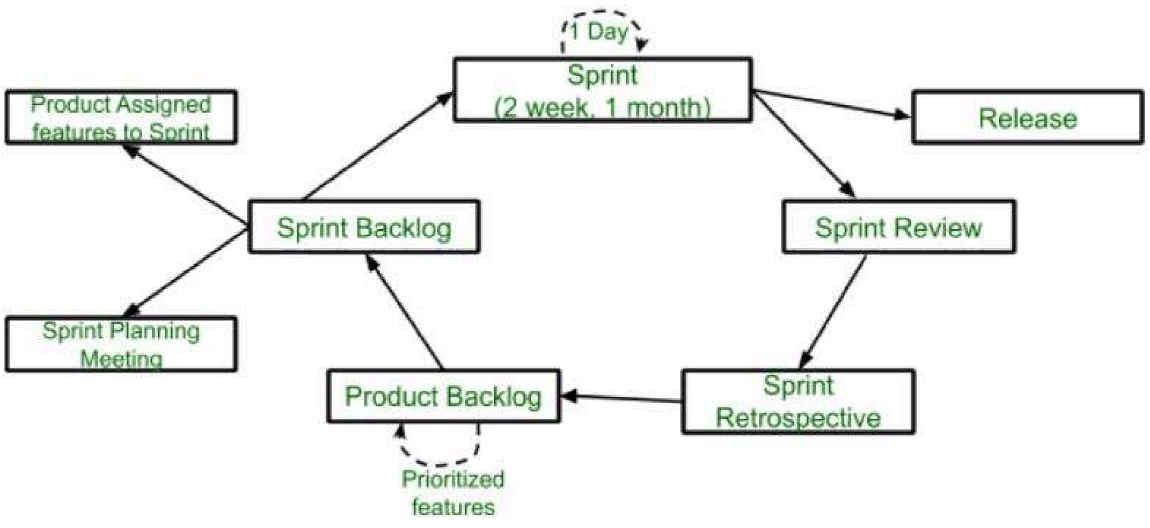
Scrum is a popular Agile framework used in software development (though it can be applied to other types of projects as well) to help teams work collaboratively, adapt to changes, and deliver high-quality results. Scrum is a management framework that teams use to self-organize tasks and work towards a common goal. It is a framework within which people can address complex adaptive problems while the productivity and creativity of delivering products are at the highest possible value. Scrum is a management framework that teams use to self-organize and work towards a common goal.

- Scrum allows us to develop products of the highest value while making sure that we maintain creativity and productivity.
- The iterative and incremental approach used in scrum allows the teams to adapt to the changing requirements.

Features of Scrum

- Scrum is a light-weighted framework
- Scrum emphasizes self-organization
- Scrum is simple to understand
- Scrum framework helps the team to work together

Lifecycle of Scrum



SCRUM ROLES

In Scrum, there are **three primary roles**. Each role plays a critical part in ensuring the team is working efficiently, collaborating well, and delivering value continuously.

1. Product Owner (PO)

The **Product Owner** represents the voice of the customer or business and has the responsibility of maximizing the value of the product being developed.

Primary Responsibilities:

- **Managing the Product Backlog**: The Product Owner creates, refines, and prioritizes the Product Backlog. The backlog items (also called **User Stories**) are essentially features, bug fixes, enhancements, or other work that needs to be done to complete the product. The Product Owner ensures that the most valuable items are worked on first.
- **Defining Product Vision and Goals**: The PO defines the overarching vision for the product, which guides the entire Scrum team. They are responsible for ensuring that the product is developed in a way that aligns with business goals, customer needs, and stakeholders' expectations.
- **Engagement with Stakeholders**: The PO acts as the main point of contact for stakeholders (such as customers, marketing, or sales teams) and ensures their feedback is incorporated into the Product Backlog.

- **Clarification of Requirements**: The Product Owner provides clarification for any requirements or questions that the team may have regarding backlog items.

Key Skills:

- Strong communication and collaboration skills to engage with stakeholders and the Scrum Team.
- A deep understanding of the business, customer needs, and market trends.

2. Scrum Master (SM)

The **Scrum Master** serves as a facilitator, coach, and protector for the Scrum team. The Scrum Master ensures that Scrum is understood and practiced correctly by the team.

Primary Responsibilities:

- Ensuring Scrum Framework Is Followed: The Scrum Master ensures that the Scrum process is being followed correctly. They help the team adhere to Scrum practices, ceremonies, and roles.
- Removing Impediments: The Scrum Master helps identify and remove obstacles that may prevent the team from completing their work. This can include anything from technical issues to organizational barriers.
- Facilitating Scrum Ceremonies: The Scrum Master facilitates key Scrum meetings (e.g., Sprint Planning, Daily Standups, Sprint Review, and Sprint Retrospective).
- Team Coaching and Development: The Scrum Master helps the team continuously improve their processes by facilitating retrospectives and encouraging self-organization. They help the team learn from failures and successes.
- Promoting Collaboration and Communication: They encourage collaboration between team members and stakeholders to ensure smooth communication and efficient workflow.

Key Skills:

- A deep understanding of Scrum and Agile principles.
- Conflict resolution and problem-solving abilities.
- Strong interpersonal and communication skills.

3. Development Team

The **Development Team** is a cross-functional group responsible for delivering the product increment. They have the skills needed to design, build, test, and deliver product features. Importantly, they are **self-organizing**.

Primary Responsibilities:

- **Delivering Increments**: The team works on items selected from the Sprint Backlog and strives to deliver a potentially shippable product increment at the end of each Sprint.
- **Self-organization**: The Development Team is responsible for organizing themselves to complete the work. They decide how to accomplish tasks and who will do what.
- **Collaboration**: They collaborate closely with the Product Owner for clarifying requirements and ensuring that the backlog items are well-defined. They also work together to resolve any technical issues during the Sprint.
- **Continuous Improvement**: The Development Team regularly evaluates their processes and identifies areas for improvement, particularly during Sprint Retrospectives.

Key Skills:

- Technical skills (e.g., coding, testing, design).
- Ability to collaborate and work as part of a cross-functional team.
- Self-motivation and accountability.

SPRINTS

A **Sprint** is the heart of Scrum—it is the time-boxed period during which the Development Team works to complete a predefined set of work from the **Sprint Backlog**.

Sprint Characteristics:

- **Time-boxed**: A Sprint has a fixed duration, usually 1 to 4 weeks. The goal is to deliver a potentially shippable product increment at the end of each Sprint.
- **Consistent Duration**: Sprints have the same length throughout the project, providing a predictable rhythm for the team.
- **No Changes During the Sprint**: Once the Sprint starts, no changes are allowed to the scope of work. This ensures that the team focuses on delivering the committed work.

Sprint Activities:

1. Sprint Planning:

- **Objective**: The team decides what work will be completed during the Sprint, based on the highest-priority items from the Product Backlog.
- **Participants**: Product Owner, Scrum Master, and Development Team.
- **Outcome**: A Sprint Backlog is created, consisting of the user stories or tasks to be worked on during the Sprint, along with the Sprint Goal—a clear, concise objective for the Sprint.

2. Daily Scrum (Standup):

- **Objective**: The team meets daily for 15 minutes to discuss progress and Challenges. Each team member answers three questions:
    - What did I do yesterday to help the team achieve the Sprint Goal?
    - What will I do today to help the team achieve the Sprint Goal?
    - Are there any impediments or blockers that I need help with?
- **Participants**: Development Team, Scrum Master (to facilitate), and Product Owner (optional).

3. Sprint Review:

- **Objective**: At the end of the Sprint, the team demonstrates the work they've completed. The goal is to gather feedback from stakeholders and review progress toward the product's vision.
- **Outcome**: The team may adjust the Product Backlog based on stakeholder feedback.

4. Sprint Retrospective:

- **Objective**: The team reflects on how the Sprint went—what went well, what could be improved, and what actions can be taken to improve in the next Sprint.
- **Outcome**: Actionable items for improving team performance and the Scrum process are identified.

BACKLOGS

There are two main types of backlogs in Scrum: Product Backlog and Sprint Backlog. The backlog is essentially a prioritized list of work that needs to be done.

Product Backlog

The Product Backlog is the single source of truth for all the work to be done to complete the product. It contains all the features, enhancements, bug fixes, and technical work required for the product.

Primary Characteristics:

- **Dynamic**: The Product Backlog is constantly evolving based on new information, customer feedback, or changes in the market.
- **Prioritized**: The Product Owner is responsible for prioritizing the items in the backlog based on their value to the business.
- **Granularity**: The higher-priority items are usually described in broad terms (e.g., features or themes), and as they get closer to being worked on, they are broken down into smaller, more specific tasks (user stories).

Sprint Backlog

The Sprint Backlog is the list of tasks that the team has committed to completing during a given Sprint. It is a subset of the Product Backlog that has been selected during Sprint Planning.

Primary Characteristics:

- **Commitment**: Once the Sprint begins, the team is committed to completing the items in the Sprint Backlog by the end of the Sprint.
- **Detailed and Actionable**: The items in the Sprint Backlog are broken down into smaller, actionable tasks so that the team knows exactly what they need to do.
- **Owned by the Team**: The Development Team owns the Sprint Backlog and is responsible for tracking progress and adjusting work as needed throughout the Sprint.

VERSION CONTROL SYSTEM

Version Control Systems are software tools used in collaborative projects, particularly in software development, to manage and track changes to files over time. It keeps a record of every single change made to the code. It also allows us to turn back to the previous version of the code if any mistake is made in the current version. Without a VCS in place, it would not be possible to monitor the development of the project.

Functionalities of VCS

- **Track changes:** Every modification made to a file is recorded, along with who made it, when, and why.
- **Revert to previous versions:** Easily undo changes or revert a file or the entire project to an earlier state if needed, saving time and effort spent manually undoing mistakes.
- **Compare changes:** Review the differences between versions to identify exactly what was changed.
- **Collaborate effectively:** Facilitate teamwork by allowing multiple developers to work on the same codebase simultaneously without overwriting each other's work.
- **Branch and merge:** Allow developers to create independent lines of development ("branches") to work on new features or bug fixes in isolation, then merge those changes back into the main project.
- **Provide backup and recovery:** Act as a backup system, ensuring that project history can be restored even if a local copy is lost.

Why are VCS important?

VCS are essential for:

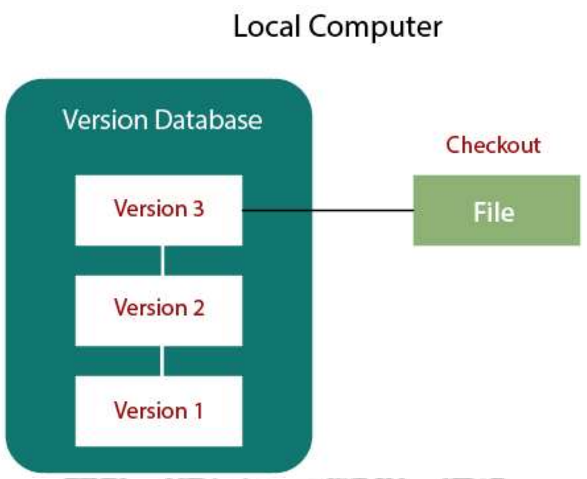
- **Ensuring code integrity and quality:** By tracking every change and facilitating reviews, VCS help maintain a consistent and functional codebase.
- **Speeding up development:** Automating tasks like testing and deployment, and enabling faster iteration and delivery of new features.
- **Promoting collaboration and communication:** Giving developers a single source of truth and enabling effective communication around changes.
- **Reducing risks:** Preventing loss of work, mitigating errors, and simplifying debugging by allowing developers to easily revert to previous versions.

Types of VCS

There are two main types of version control systems:

1. **Local Version Control System**

The Local Version Control System is located in your local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost. If anything happens to a single version, all the versions made after that will be lost. Also, with the Local Version Control System, it is not possible to collaborate with other collaborators.



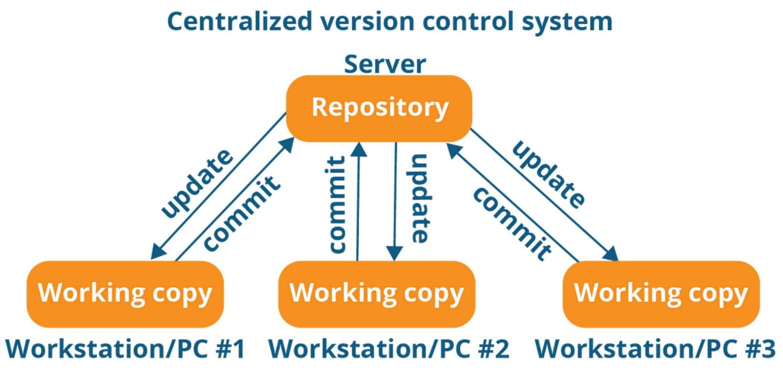
2. Centralized Version Control Systems (CVCS):

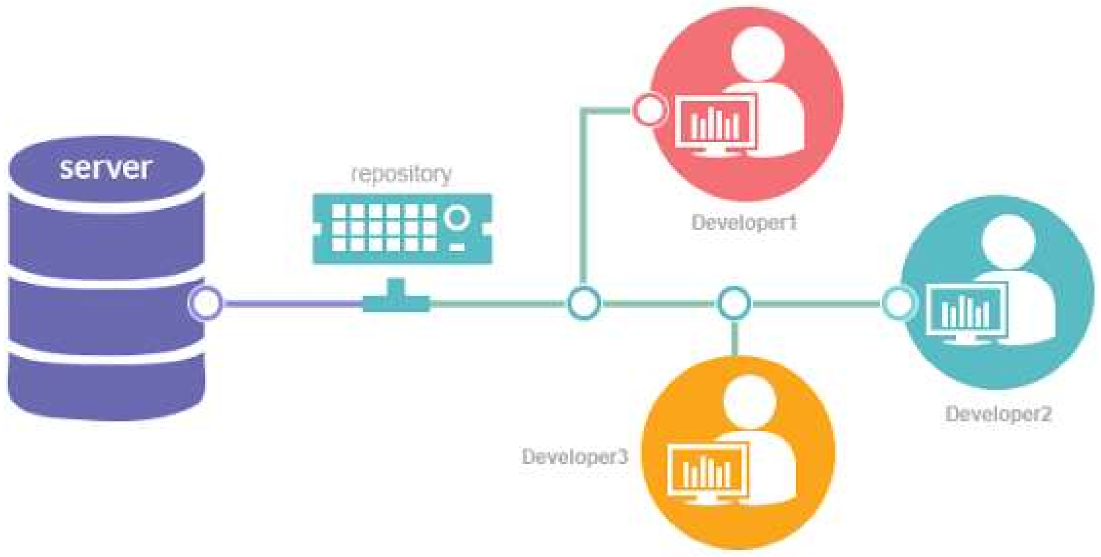
All file versions are stored on a single central server. It contains all the files related to the project, and many collaborators checkout files from this single server (only have a working copy). Developers check out files from the server, make changes, and then check them back in. Administrators manage access and merge the work of collaborators.

Advantages: Simple to set up and manage, suitable for smaller teams or projects with limited collaboration needs.

Disadvantages: Single point of failure (if the server goes down, work halts), slower performance due to constant communication with the server, less flexible in customizing workflows.

Examples: Subversion (SVN)





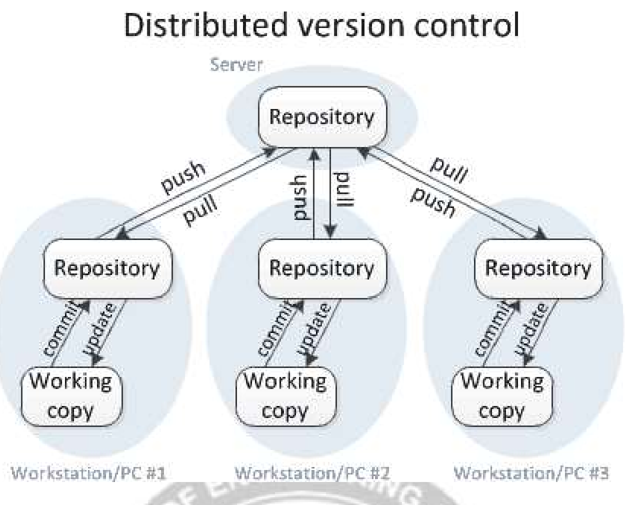
3. Distributed Version Control Systems (DVCS):

Each collaborator will have an exact copy (mirroring) of the main repository(including its entire history) on their local machines. Developers can work offline, make commits locally, and then push/pull changes to/from a central repository or other users. There will be one or more servers and many collaborators similar to the centralized system.

Advantages: Allows developers to work offline, better handling of branches and merges, faster performance due to local operations, more resilient as each clone acts as a backup. Hence even if the server crashes we would not lose everything as several copies are residing in several other computers.

Disadvantages: More complex to set up and understand initially, requires more storage space as each developer has a full copy, potentially higher bandwidth usage when syncing with remote servers.

Examples: Git, Mercurial



Popular VCS tools

- **Git:** The most widely used distributed version control system, known for its speed, flexibility, and robust features. It is the backbone of services like GitHub(Microsoft), GitLab (GitLab Inc), and Bitbucket (Atlassians).
- **Subversion (SVN)**: **Apache Subversion (SVN)** is a **centralized version control system (VCS)** used to manage changes to source code and documents over time. It is simpler to learn than Git and favored by some for its centralized control.
- **Mercurial:** Another distributed version control system, similar to Git but often considered to have a simpler interface. Bitbucket used to support Mercurial, but dropped support in 2020, fully switching to Git.

The choice between a centralized and distributed system depends on the project's specific needs, team size, workflow preferences, and the scale of the project. While DVCSs like Git are widely considered the standard for modern software development due to their flexibility and scalability, CVCSs like SVN remain relevant in some scenarios, particularly for smaller teams or specific enterprise requirements.