Documentation
on
Project Management Methodologies,Version Control,Git

Submitted to
SayOne Technologies

By **Anisha Vineeth**
Software Trainee
SayOne Technologies

Table of contents

1.INTRODUCTION

This document contains a brief description on the topics, Project Management Methodologies, Version Control, GIT.The commonly used different types of project management methods are included along with the advantages and disadvantages of each of them.The introduction to Version control , its benefits and different types of version control are discussed.

Finally GIT is described along with basic GIT commands, and workflow . Working with Git is one of the essential things in every project since it allows collaboration of each team members contribution to the project.

2.Project Management Methodologies

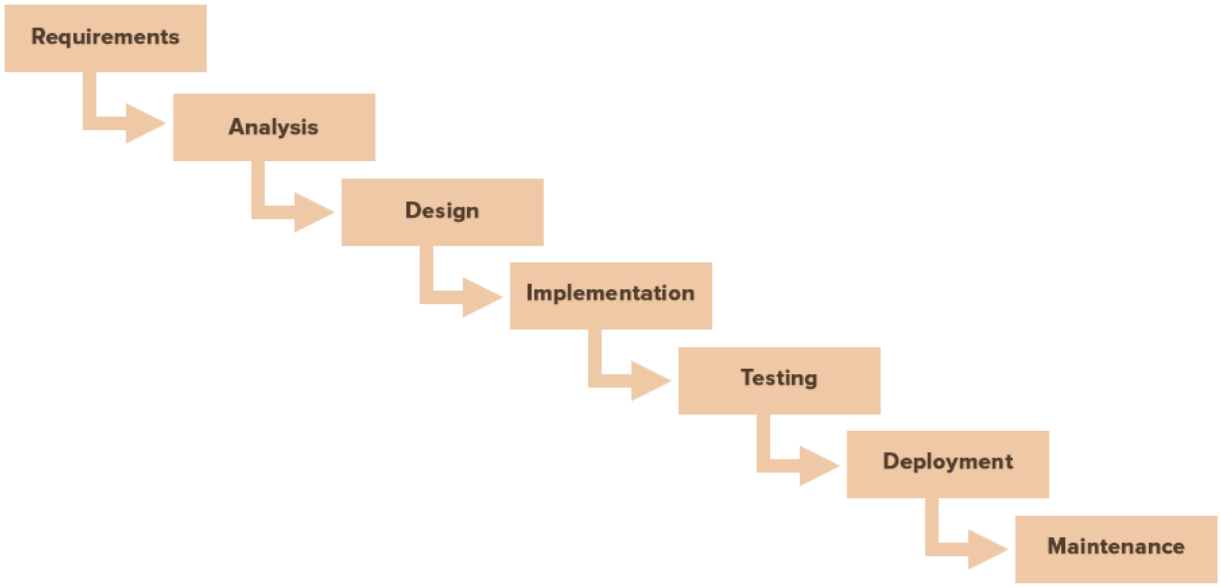
A project management methodology is essentially a set of guiding principles and processes for managing a project. Your choice of methodology defines how you work and communicate.

2.1 TYPES OF PROJECT MANAGEMENT METHODOLOGY:

Most project management tools are specialised to use a handful of methodologies. This will be a factor in what methodology you eventually choose to use.The different project management methodologies are discussed below,

2.1.1 WATERFALL

The Waterfall methodology is the oldest methodology on this list. It was first outlined by Dr. Winston Royce in 1970 as a response to managing the increasingly complex nature of software development. Since then, it has become widely adopted, most prominently in the software industry.The Waterfall methodology is sequential. It is also heavily requirements-focused. You need to have a crystal clear idea of what the project demands before proceeding further. There is no scope for correction once the project is underway. The Waterfall method is divided into discrete stages.Each stage in this process is self-contained; you wrap up one stage before moving onto another.



Advantages:

- ***Ease of use***: This model is easy to understand and use. The division between stages is intuitive and easy to grasp regardless of prior experience.
- ***Structure:*** The rigidity of the Waterfall method is a liability, but can also be a strength. The clear demarcation between stages helps organize and divide work. Since you can't go back, you have to be "perfect" in each stage, which often produces better results.
- ***Documentation:*** The sharp focus on gathering and understanding requirements makes the Waterfall model heavily reliant on documentation. This makes it easy for new resources to move in and work on the project when needed.

Disadvantages:

- ***Higher risk:*** The rigidity of this methodology means that if you find an error or need to change something, you have to essentially start the project from the beginning. This substantially increases the risk of project failure.
- ***Front-heavy:*** The entire Waterfall approach depends heavily on your understanding and analyzing requirements correctly. Should you fail to do that - or should the requirements change - you have to start over. This lack of flexibility makes it a poor choice for long and complex projects.

The Waterfall methodology is most commonly used in software development. It works best for the following project types:

- Short, simple projects
- Projects with clear and fixed requirements
- Projects with changing resources that depend on in-depth documentation

2.1.2. AGILE

Agile, another software development-focused PM methodology, emerged as a response to the failure of Waterfall method for managing complex projects. Although Agile PM ideas had been in use in the software industry for quite a while, it formally came into

being in 2001 when several IT representatives released the "Agile Manifesto".In approach and ideology, Agile is the opposite of the Waterfall method. As the name implies, this method favors a fast and flexible approach. There is no top-heavy requirements-gathering. Rather, it is iterative with small incremental changes that respond to changing requirements.

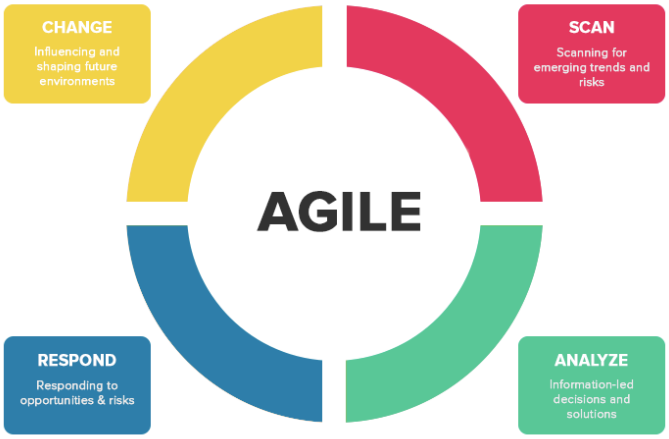


Fig: Graphical Representation of Agile model

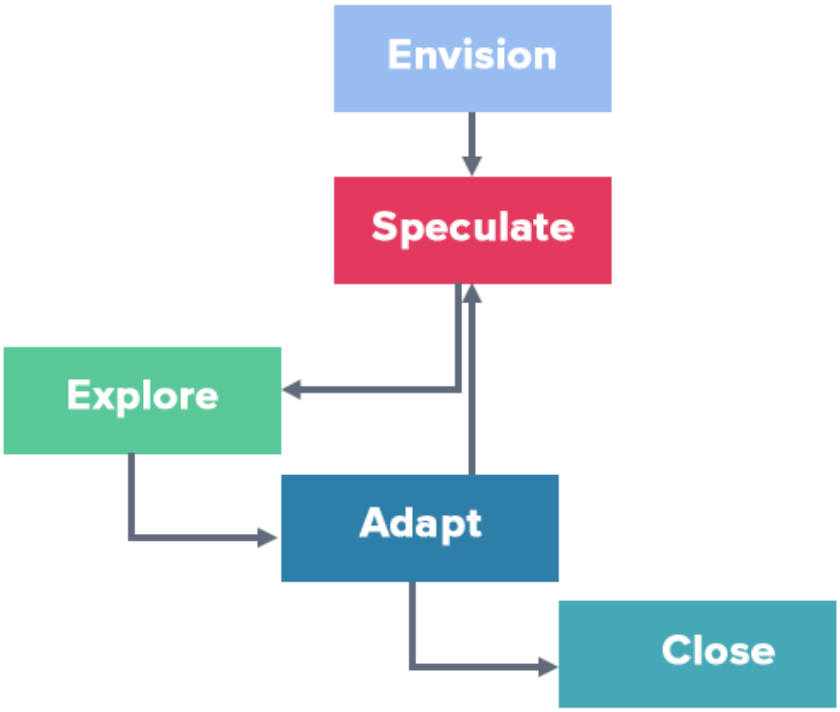
Advantages:

- **Flexibility and freedom:** Since there are no fixed stages or focus on requirements, it gives your resources much more freedom to experiment and make incremental changes. This makes it particularly well-suited for creative projects.
- **Lower risk:** With Agile management, you get regular feedback from stakeholders and make changes accordingly. This drastically reduces the risk of project failure since the stakeholders are involved at every step.

Disadvantages:

- **No fixed plan:** The Agile approach emphasizes responding to changes as they occur. This lack of any fixed plan makes resource management and scheduling harder. You will constantly have to juggle resources, bringing them on/off on an ad-hoc basis.

- **Collaboration-heavy:** The lack of a fixed plan means all involved departments - including stakeholders and sponsors - will have to work closely to deliver results. The feedback-focused approach also means that stakeholders have to be willing (and available) to offer feedback quickly.



The flexibility of the Agile approach means that you can adapt it to different types of projects.

That said, this methodology works best for:

- When you don't have a fixed end in mind but have a general idea of a product.
- When the project needs to accommodate quick changes.
- If collaboration and communication are your key strengths (and planning isn't).

2.1.3. HYBRID

The Hybrid approach, as the name implies, is a combination of the Waterfall and Agile methodologies. It takes the best parts of both Waterfall and Agile and combines them in a flexible yet structured approach that can be used across different projects. The Hybrid methodology focuses on gathering and analyzing requirements initially - a nod to the Waterfall method. From thereon, it takes the flexibility of Agile approach with an emphasis on rapid iterations.By combining attributes of Waterfall and Agile, the Hybrid method (sometimes called "Structured Agile") gives you the best of both worlds.

Advantages

- **Increased flexibility:** Past the planning stage, the Hybrid method affords you significantly increased flexibility when compared to the Waterfall method. As long as the requirements don't change substantially, you can make changes as they're requested.
- **More structured:** By borrowing the initial planning phase from Waterfall, the Hybrid method addresses one of the biggest complaints about the Agile approach - lack of structure and planning. Hence, you get the "best of both worlds".

Disadvantages

- **Requires compromise:** Since you're essentially reconciling two polar opposite approaches, both sides will need to compromise on requirements and flexibility.
- **"Best of both worlds"** approach robs you of the flexibility of Agile and the sure footedness of Waterfall. Any iterations you make will have to comply with the budgeting and scheduling constraints set up front.

- The Hybrid approach is best-suited for projects that have middling requirements when compared to Agile and Waterfall, i.e. they require structure as well as flexibility.
- Mostly, this would be medium-sized projects with moderately high complexity but fixed budgets. You would likely have an idea of the end product but you are also open to experimentation. You will need close collaboration, especially past the planning stage.

2.1.4.SCRUM

Scrum isn't a fully-featured project management methodology. Rather, it describes an approach to Agile management with a focus on project teams, *short "sprints"* and *daily stand-up meetings*. While it borrows the principles and processes from Agile, Scrum has its own specific methods and tactics for dealing with project management.

The Scrum approach places the project team front and center of the project. Often, there is no project manager. Instead, the team is expected to be self-organizing and self-managing. This makes it ideal for highly focused and skilled teams, but not so much for others.Scrum is best suited when the cost of delay is high and deadlines should meet a minimal delay. Scrum is often used when the end product is unclear or the requirements have no proper feedback from the clients.

Unlike Waterfall, the Scrum model adopts more flexible disciplines which are open to last-minute changes. Teamwork, inspection, and transparency are key factors in the Scrum method.

The structure:

- product backlog (a set of top-priority tasks allowing to build MVP as soon as possible)
- sprint backlog (contains high priority features that developers are going to deal with following 2–4 weeks)
- sprint itself

Advantages

- **Scrum "sprints":** The Scrum approach is heavily focused on 30-day "sprints". This is where the project team breaks down a wish list of end-goals into small chunks, then works on them in 30-day sessions with daily stand-up meetings. This makes it easy to manage large and complex projects.
- **Fast paced:** The "sprint" approach with its 30-day limit and daily stand-up meetings promotes rapid iteration and development.

- **Team-focused:** Since the project team is expected to manage itself, Scrum teams have clear visibility into the project. It also means that project leaders can set their own priorities as per their own knowledge of their capabilities.

Besides these, it has all the benefits of Agile - rapid iteration and regular stakeholder feedback.

Disadvantages

- **Scope creep:** Since there is no fixed end-date, nor a project manager for scheduling and budgeting, Scrum can easily lead to scope creep.
- **Higher risk:** Since the project team is self-managing, there is a higher risk of failure unless the team is highly disciplined and motivated. If the team doesn't have enough experience, Scrum has a very high chance of failure.
- **Lack of flexibility:** The project-team focus means that any resource leaving the team in-between will hugely impact the net results. This approach is also not flexible enough for large teams.

- The Scrum approach is best for highly experienced, disciplined and motivated project teams who can set their own priorities and understand project requirements clearly.
- It has all the flaws of Agile along with all its benefits. It works for large projects, but fails if the project team itself is very large.
- In short: use Scrum if you're developing complex software and have an experienced team at your disposal.

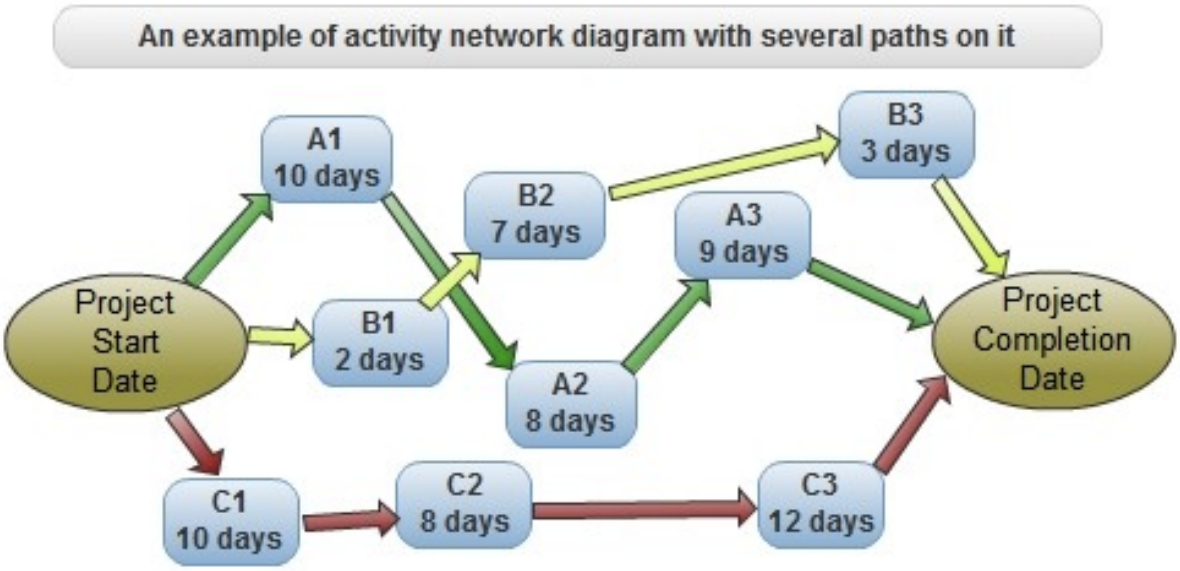
2.1.5. CRITICAL PATH METHOD

A critical path in project management is certain tasks that need to be performed in a clear order and for a certain period.If part of one task can be slowed down or postponed for a term without leaving work on others, then such a task is not critical. While tasks with

a critical value cannot be delayed during the implementation of the project and are limited in time.

Critical Path Method (CPM) is an algorithm for planning, managing and analyzing the timing of a project. The step-by-step CPM system helps to identify critical and non-critical tasks from projects' start to completion and prevents temporary risks.In the Critical Path Method, you categorize all activities needed to complete the project within a work breakdown structure. Then you map the projected duration of each activity and the dependencies between them.This helps you map out activities that can be completed simultaneously, and what activities should be completed before others can start.

Critical tasks have a zero run-time reserve. If the duration of these tasks changes, the terms of the entire project will be "shifted". That is why critical tasks in project management require special control and timely detection of risks.



Advantages:

Critical path analysis is required in order to predict the timing of project's completion.

- This method visualises projects in a clear graphical form. It defines the most important tasks.
- Saves time and helps in the management of deadlines.

- Helps to compare the planned with the real status.
- Identifies all critical activities that need attention.
- Makes dependencies clear and transparent.

Disadvantages

- **Scheduling requires experience:** As any experienced project manager will tell you, things always take more time than you expect. If you don't have real-world experience with scheduling, you are bound to miscalculate time for each activity.
- **No flexibility:** Like the Waterfall method, CPM is front-heavy. You need to plan everything out at the very start. If there are any changes, it makes the entire schedule irrelevant. This makes this method unsuitable for projects with changing requirements.

- The Critical Path Method is best-suited for projects with interdependent parts. If you require tasks to be completed simultaneously, or for one task to end before another can begin, you'll want to use this methodology.
- CPM finds a lot of application in complex, but repetitive activities such as industrial projects. It is less suited for a dynamic area such as creative project management.

Key Stages of Critical Path Method:

CPM involves the following consecutive steps:

1. **Identify activities /tasks**

Knowing the scope of the project, yo12u can divide the work structure into the list of activities, giving them names or codes. All activities in the project must have a duration and a specific date.

2. **Identify the sequences.**

This is the most important step because it gives a clear idea of the links between activities and helps establish dependencies because some actions will depend on the completion of others.

3. **Create a network of your activities.**

Once you have determined which actions depend on each other, you can create a network diagram or a path analysis chart. Using the arrows, you can easily connect activities based on their dependencies.

4. **Determine the time intervals for completing each activity.**

Estimating how much time will be spent for each action, you will be able to determine the time needed to complete the entire project (small projects can be assessed in a few days; more complex ones require a long evaluation).

5. **Find a critical path.**

The activity network will help you create the longest sequence on the path or the critical path using the following parameters:

- Early Start – the time when all previous tasks are completed.
- Early Finish – the nearest start time and the time required to complete the task.
- Late Finish – all activities are completed without postponing the deadlines.
- Late Start – the last end time minus the time it takes to complete the task.

2.1.6. CRITICAL CHAIN PROJECT MANAGEMENT METHOD

Critical Chain PM is one of the newer project management methodologies out there. It was developed as an alternative to the Critical Path method with a focus on resource management.
With CCPM, you work backward from the end goal. You recognize the deliverables, then use past experience to map out the tasks required to complete the project. You also map out the interdependencies between resources and allocate them accordingly to each task.

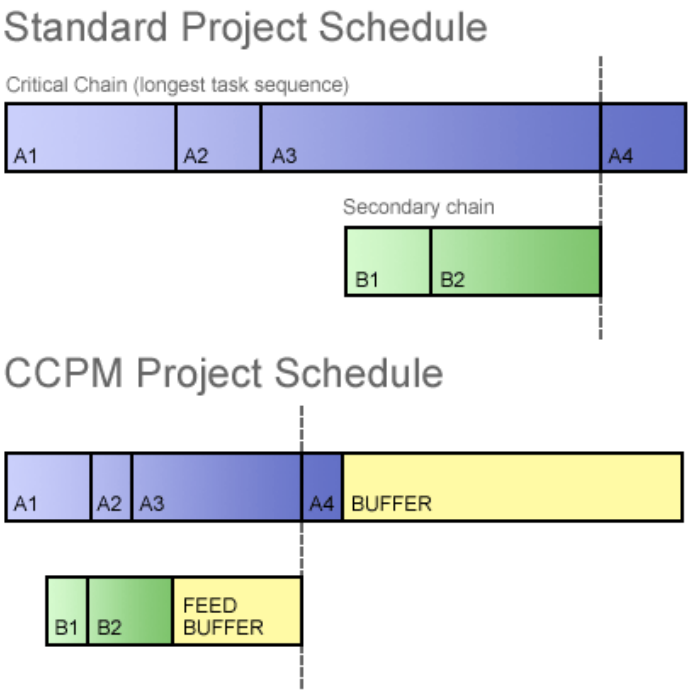


Fig shows difference between traditional and a CCPM project schedule

CCPM emphasizes resource utilisation and minimising lost productivity. It is heavily reliant on "monotasking", i.e. focusing on the task at hand and avoiding multitasking.
For resource-strapped project teams, CCPM can be a powerful methodology.

Advantages

- **Resource-efficient:** The entire focus on proper resource management makes CCPM one of the most resource-efficient project management methodologies around. The emphasis on monotasking is also well-aligned with our modern understanding of the detrimental effects of multitasking.

- **Focused on end goal:** CCPM doesn't obsess over the "optimum" solution to a problem. Instead, it prioritizes "good enough" solutions that can help meet the end-goal. Since you also work backward from the end-goal, CCPM usually yields better results for complex projects.

Disadvantages

- **Not appropriate for multi-project environments:** CCPM's resource-focused approach can only work in single-project environments. In multi-project environments, projects might share resources. CCPM can't plan for resource distribution in such a scenario.

- **Delays common:** CCPM allots a gap or padding between tasks to derive a task time length. In theory, this is supposed to make up for resources overestimating their own efficiency. In reality, resources, following Parkinson's Law, fill up the padding with inordinate delays.

- CCPM works best in environments where resources are devoted to a single project.
- If you have a dedicated team for a project, it works great.
- If your team is spread across several projects, you'll struggle with resource planning.
- The resource-focused approach of CCPM is also ideal for resource-strapped project teams. If you find yourself constantly overworked or missing deadlines, the CCPM methodology might be for you.

2.1.7. INTEGRATED PROJECT MANAGEMENT METHOD

Integrated project management is the collection of processes that ensure various elements of projects are properly coordinated. IPM - sometimes also called "Integrated Project Delivery" - is a common project management methodology in creative industries. This methodology emphasizes sharing and standardization of processes across the organization.

The IPM approach came about as a response to the increasingly integrated nature of creative campaigns. You don't just produce a single ad; you integrate the ad with microsites, digital content, etc. Most creative projects are a piece of a larger campaign.

Key Components of Integrated Project Management

By integrating processes across the organization, IPM gives project managers better insight into the project and access to the right resources.

Advantages

- **Transparency:** Integrating processes across the organization improves transparency within the organization. The IPM approach focuses on team members documenting and meeting regularly, which helps keep everyone in the loop.

- **Accountability:** The integrated nature of the IPM approach makes the entire project team responsible for the project. Since no team member can operate in a silo, IPM improves accountability.

Disadvantages

Requires extensive planning: With the IPM approach, you will have to plan extensively upfront and ensure that all processes are well-integrated. This increases your burden significantly and can lead to delays.

Large agencies with diverse teams and processes benefit the most from Integrated Project Management. It works best for complex creative projects where you need resources from multiple teams and departments to interface with each other.

2.1.8. PRISM

PRiSM (Projects integration Sustainable Methods) is a project management methodology developed by Green Project Management (GPM) Global.As hinted by the creator's name, the PRiSM approach focuses on accounting for and minimising adverse environmental impacts of the project. It is different from traditional methodologies in that it extends beyond the end of the project. Instead, it factors in the entire lifecycle of the project post-delivery to maximize sustainability.

PRiSM is mostly suited for large and complex real estate and industrial projects where sustainability is a key concern.

Advantages

The PRiSM approach is very pertinent for modern projects where environmental costs and sustainability are key success criteria. For large projects where reducing energy consumption, managing waste and minimising environmental impact is critical, PRiSM offers a viable project management ideology.

Disadvantages

PRiSM is unsuitable for projects where environmental impact is not a concern (such as software or creative projects).

Success with the PRiSM approach also requires every part of the project team - including outside contractors and stakeholders - to be onboard with the sustainability principle - a hard ask in most organizations.

PRiSM principles:

3.Version Control

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.Version control systems are a category of software tools that help a software team manage changes to source code over time.Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.

Software developers working in teams are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.

Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting. Changes made in one part of the software can be incompatible with those made by another developer working at the same time. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team.Further, in all software development, any change can introduce new bugs on its own and new software can't be trusted until it's tested. So testing and development proceed together until a new version is ready.

3.1 BENEFITS OF VERSION CONTROL SYSTEMS

Developing software without using version control is risky, like not having backups. Version control can also enable developers to move faster and it allows software teams to preserve efficiency and agility as the team scales to include more developers.

Version Control Systems (VCS) are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System).One of the most popular VCS tools in use today is called Git.

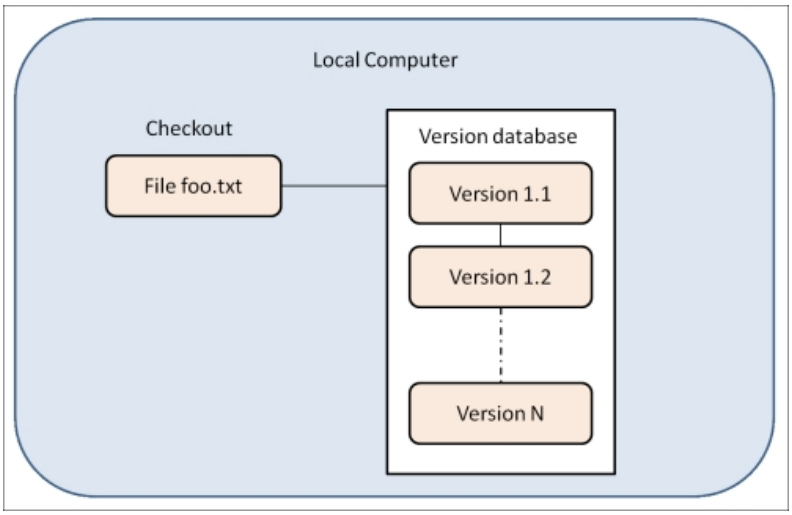
1. A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents.
2. Branching and merging. Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of changes. Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict.
3. Traceability. Being able to trace each change made to the software and connect it to project management and bug tracking software such as Jira, and being able to annotate each change with a message describing the purpose and intent of the change can help not only with root cause analysis and other forensics.

3.2 TYPES OF VERSION CONTROL

- Local version control system
- Centralized version control system
- Distributed Version control system

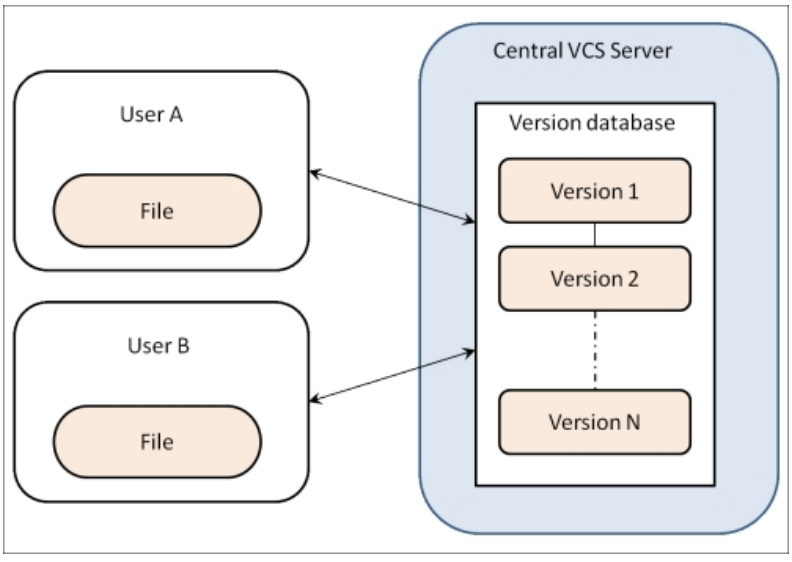
- *Local Version control system:*

After understanding that maintaining multiple versions of files by just following a file naming convention is highly error prone, a local version control system was the first successful attempt to solve this issue.This tool basically works by keeping patch sets (that is, the difference between the file's content at progressive stages) using a special format in the version tracker that is stored in your local hard disk.It can then recreate the file's contents exactly at any given point in time by adding up all the relevant patches in order and "checking it out" (reproducing the content to the user's workplace).



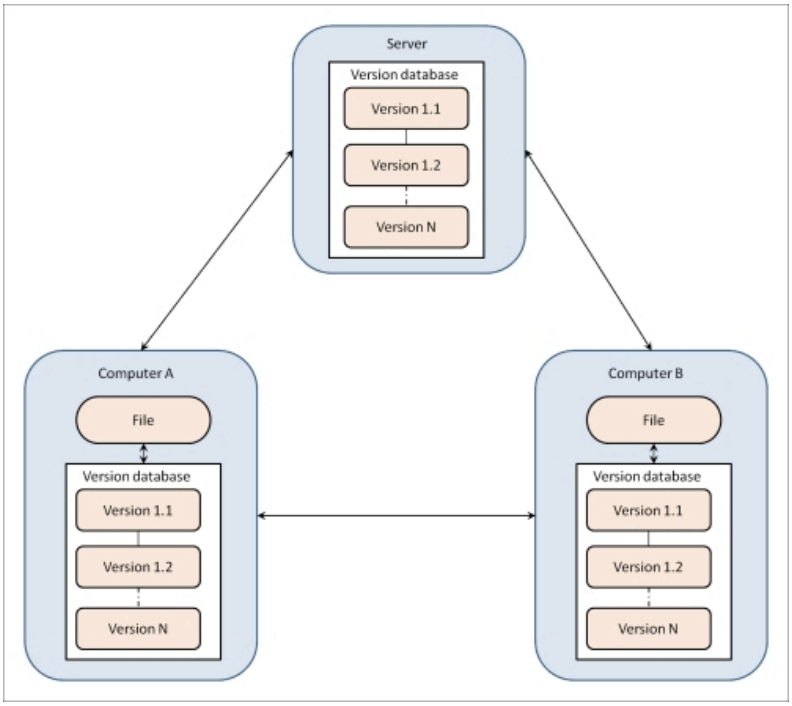
- *Centralized version control system:*

With centralized version control systems, you have a single “central” copy of your project on a server and commit your changes to this central copy. You pull the files that you need, but you never have a full copy of your project locally. Some of the most common version control systems are centralized, including Subversion (SVN) and Perforce.



- *Distributed Version control system:*

With distributed version control systems (DVCS), you don't rely on a central server to store all the versions of a project's files. Instead, you clone a copy of a repository locally so that you have the full history of the project. Two common distributed version control systems are Git and Mercurial.



4.GIT

The most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel. A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments).

Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control System). Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.In addition to being distributed, Git has been designed with performance, security and flexibility in mind.

Performance:

The raw performance characteristics of Git are very strong when compared to many alternatives. Committing new changes, branching, merging and comparing past versions are all optimized for performance. The algorithms implemented inside Git take advantage of deep knowledge about common attributes of real source code file trees, how they are usually modified over time and what the access patterns are.

Unlike some version control software, Git is not fooled by the names of the files when determining what the storage and version history of the file tree should be, instead, Git focuses on the file content itself. After all, source code files are frequently renamed, split, and rearranged. The object format of Git's repository files uses a combination of delta encoding (storing content differences), compression and explicitly stores directory contents and version metadata objects.

Being distributed enables significant performance benefits as well.

Security:

Git has been designed with the integrity of managed source code as a top priority. The content of the files as well as the true relationships between files and directories, versions, tags and commits, all of these objects in the Git repository are secured with a cryptographically secure hashing algorithm called SHA1. This protects the code and the

change history against both accidental and malicious change and ensures that the history is fully traceable.With Git, you can be sure you have an authentic content history of your source code.

Some other version control systems have no protections against secret alteration at a later date. This can be a serious information security vulnerability for any organization that relies on software development.

Flexibility:

One of Git's key design objectives is flexibility. Git is flexible in several respects: in support for various kinds of nonlinear development workflows, in its efficiency in both small and large projects and in its compatibility with many existing systems and protocols.

Git has been designed to support branching and tagging as first-class citizens (unlike SVN) and operations that affect branches and tags (such as merging or reverting) are also stored as part of the change history. Not all version control systems feature this level of tracking.

4.1 BASIC GIT COMMANDS:

Git task	Notes	Git commands
Tell Git who you are	Configure the author name and email address to be used with your commits. Note that Git strips some characters (for example trailing periods) from `user.name`.	`git config --global user.name "Sam Smith"` `git config --global user.email sam@example.com`
Create a new local repository		`git init`

Check out a repository	Create a working copy of a local repository:	`git clone /path/to/repository`
	For a remote server, use:	`git clone username@host:/path/to/repository`

Add files	Add one or more files to staging (index):	`git add <filename>`
		
		`git add *`

Commit	Commit changes to head (but not yet to the remote repository):	`git commit -m "Commit message"`
	Commit any files you've added with `git add`, and also commit any files you've changed since then:	`git commit -a`
Push	Send changes to the master branch of your remote repository:	`git push origin master`
Status	List the files you've changed and those you still need to add or commit:	`git status`
Connect to a remote repository	If you haven't connected your local repository to a remote server, add the server to be able to push to it:	`git remote add origin <server>`

	List all currently configured remote repositories:	`git remote -v`
Branches	Create a new branch and switch to it:	`git checkout -b <branchname>`
	Switch from one branch to another:	`git checkout <branchname>`
	List all the branches in your repo, and also tell you what branch you're currently in:	`git branch`
	Delete the feature branch:	`git branch -d <branchname>`
	Push the branch to your remote repository, so others can use it:	`git push origin <branchname>`
	Push all branches to your remote repository:	`git push --all origin`
	Delete a branch on your remote repository:	`git push origin :<branchname>`
Update from the remote repository	Fetch and merge changes on the remote server to your working directory:	`git pull`
	To merge a different branch into your active branch:	`git merge <branchname>`

	View all the merge conflicts:
	View the conflicts against the base file:
	Preview changes, before merging:
		`git diff`
		`git diff --base <filename>`
		`git diff <sourcebranch> <targetbranch>`

	After you have manually resolved any conflicts, you mark the changed file:	`git add <filename>`

Git task	Notes	Git commands
Tell Git who you are	Configure the author name and email address to be used with your commits. Note that Git strips some characters (for example trailing periods) from `user.name`.	`git config --global user.name "Sam Smith"` `git config --global user.email sam@example.com`
Create a new local repository		`git init`
Check out a repository	Create a working copy of a local repository:	`git clone /path/to/repository`
	For a remote server, use:	`git clone username@host:/path/to/repository`

Add files	Add one or more files to staging (index):	`git add <filename>`
		
		`git add *`

Commit	Commit changes to head (but not yet to the remote repository):	`git commit -m "Commit message"`
	Commit any files you've added with `git add`, and also commit any files you've changed since then:	`git commit -a`
Push	Send changes to the master branch of your remote repository:	`git push origin master`
Status	List the files you've changed and those you still need to add or commit:	`git status`
Connect to a remote repository	If you haven't connected your local repository to a remote server, add the server to be able to push to it:	`git remote add origin <server>`
	List all currently configured remote repositories:	`git remote -v`
Branches	Create a new branch and switch to it:	`git checkout -b <branchname>`
	Switch from one branch to another:	`git checkout <branchname>`

	List all the branches in your repo, and also tell you what branch you're currently in:	`git branch`
	Delete the feature branch:	`git branch -d <branchname>`
	Push the branch to your remote repository, so others can use it:	`git push origin <branchname>`
	Push all branches to your remote repository:	`git push --all origin`
	Delete a branch on your remote repository:	`git push origin :<branchname>`
Update from the remote repository	Fetch and merge changes on the remote server to your working directory:	`git pull`
	To merge a different branch into your active branch:	`git merge <branchname>`

	View all the merge conflicts:
	View the conflicts against the base file:
	Preview changes, before merging:
		`git diff`
		`git diff --base <filename>`
		`git diff <sourcebranch> <targetbranch>`

	After you have manually resolved any conflicts, you mark the changed file:	`git add <filename>`

Tags	You can use tagging to mark a significant changeset, such as a release:	`git tag 1.0.0 <commitID>`
	CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using:	`git log`
	Push all tags to remote repository:	`git push --tags origin`
Undo local changes	If you mess up, you can replace the changes in your working tree with the last content in head: Changes already added to the index, as well as new files, will be kept.	`git checkout -- <filename>`

	Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this:	`git fetch origin`
		
		`git reset --hard origin/master`

Search	Search the working directory for `foo()`:	`git grep "foo()"`

4.2 GIT WORKFLOW:

Git Workflow is a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner. Git workflows encourage users to leverage Git effectively and consistently. Git offers a lot of flexibility in how users manage changes. Given Git's focus on flexibility, there is no standardized process on how to interact with Git. When working with a team on a Git managed project, it's important to make sure the team is all in agreement on how the flow of changes will be applied. To ensure the team is on the same page, an agreed upon Git workflow should be developed or selected. There are several publicized Git workflows that may be a good fit for your team.

What is a successful Git workflow?

When evaluating a workflow for a team, it's most important that team's culture is considered. The workflow should be enhanced for the effectiveness of a team and not be a burden that limits productivity. Some things to consider when evaluating a Git workflow are:

- Does this workflow scale with team size?
- Is it easy to undo mistakes and errors with this workflow?
- Does this workflow impose any new unnecessary cognitive overhead to the team?

Centralized Workflow:

The Centralized Workflow uses a central repository to serve as the single point-of-entry for all changes to the project. Instead of trunk, the default development branch is called master and all changes are committed into this branch. This workflow doesn't require any other branches besides master.

The Centralized Workflow is similar to other workflows in its utilization of a remote server-side hosted repository that developers push and pull form. Compared to other workflows, the Centralized Workflow has no defined pull request or forking patterns. A Centralized Workflow is generally better suited for teams migrating from SVN to Git and smaller size teams.

How it works?

Developers start by cloning the central repository. In their own local copies of the project, they edit files and commit changes as they would with SVN; however, these new commits are stored locally - they're completely isolated from the central repository. This lets developers defer synchronizing upstream until they're at a convenient break point.

To publish changes to the official project, developers "push" their local master branch to the central repository. This is the equivalent of svn commit, except that it adds all of the local commits that aren't already in the central master branch.

- ***Initialize the central repository***

First someone needs to create the central repository on a server. If it's a new project, you can initialize an empty repository. Otherwise, you'll need to import an existing Git or SVN repository.

Central repositories should always be bare repositories (they shouldn't have a working directory), which can be created as follows:
ssh user@host git init --bare /path/to/repo.git

Be sure to use a valid SSH username for user, the domain or IP address of your server for host, and the location where you'd like to store your repo for /path/to/repo.git. Note that the .gitextension is conventionally appended to the repository name to indicate that it's a bare repository.

- ***Clone the central repository***

Next Central repositories should always be bare repositories (they shouldn't have a working directory), which can be created as follows:
ssh user@host git init --bare /path/to/repo.git

Be sure to use a valid SSH username for user, the domain or IP address of your server for host, and the location where you'd like to store your repo for /path/to/repo.git. Note that the .gitextension is conventionally appended to the repository name to indicate that it's a bare repository.

- ***Make changes and commit***

Once the repository is cloned locally, a developer can make changes using the standard Git commit process: edit, stage, and commit. If you're not familiar with the staging area, it's a way to prepare a commit without having to include every change in the working directory. This lets you create highly focused commits, even if you've made a lot of local changes.

```
git status
git add<some-file>
git commit
```

- ***Push new commits to central repository***

Once the local repository has new changes committed. These change will need to be pushed to share with other developers on the project.

```
git push origin master
```

- ***Managing conflicts***

The central repository represents the official project, so its commit history should be treated as sacred and immutable. If a developer's local commits diverge from the central repository, Git will refuse to push their changes because this would overwrite official commits.

Before the developer can publish their feature, they need to fetch the updated central commits and rebase their changes on top of them. This is like saying, "I want to add my changes to what everyone else has already done." The result is a perfectly linear history, just like in traditional SVN workflows.

If local changes directly conflict with upstream commits, Git will pause the rebasing process and give you a chance to manually resolve the conflicts. The nice thing about Git is that it uses the same git status and git add commands for both generating commits and resolving merge conflicts. This makes it easy for new developers to manage their own merges.

5.Conclusion

I hereby conclude the documentation of Project Management Methodologies, Version Control, and Git. An overall overview of the above mentioned topics are discussed in this documentation.

References

https://www.workamajig.com/blog/project-management-methodologies

https://www.workamajig.com/blog/project-management-methodologies

https://hu.atlassian.com/git/tutorials/what-is-version-control

https://subscription.packtpub.com/book/application_development/9781849517522/1/ch01lvl1sec12/types-of-version-control-systems

https://hu.atlassian.com/git/tutorials/what-is-git

https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

https://www.atlassian.com/git/tutorials/comparing-workflows