Documentation

On

Project Management Methodologies, Version Control and Git.

Submitted to,

SayOne Technologies,

By,

Alisha Vineeth,

Software Trainee,

SayOne Technologies.

WTC – Infopark,

Kochi - 682042

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

This document covers the basics related to the Project Management Methodologies, Version Control and Git. Various popular project management methodologies like Waterfall, Agile, Hybrid, Scrum, Critical Path Method, Critical Chain Management, Integrated Project Management and Prism are discussed in detail. Version Control, its benefits and types are explained.

Working of Git along with the set of basic commands is explained in detail. Centralised workflow, which is a common workflow in Git is explained step by step along with necessary commands. This document serves as a guide even for a beginner in the current topics.

CHAPTER 2
PROJECT MANAGEMENT METHODOLOGIES

2.1 Introduction to Project Management

A **Project** is a temporary endeavour undertaken to create a unique product, service or result. The development of software for an improved business process, the construction of a building or bridge, the relief effort after a natural disaster, the expansion of sales into a new geographic market — all are projects.

Project management, then, is the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements. It has always been practiced informally, but began to emerge as a distinct profession in the mid-20th century.

A **methodology** is a model, which project managers employ for the design, planning, implementation and achievement of their project objectives. There are different project management methodologies that cater to the needs of different projects spanned across different business domains. It refers to applying different principles, themes, frameworks, processes and standards to help provide structure to the way we deliver projects. Some project management methodologies simply define **principles**, like agile. Others define a 'full-stack' methodology framework of **themes**, **principles**, and **processes**, such as Prince2. Some are an extensive list of **standards** with some process, like PMI's PMBOK, or XP and some are very light, and simply define process, like Scrum.

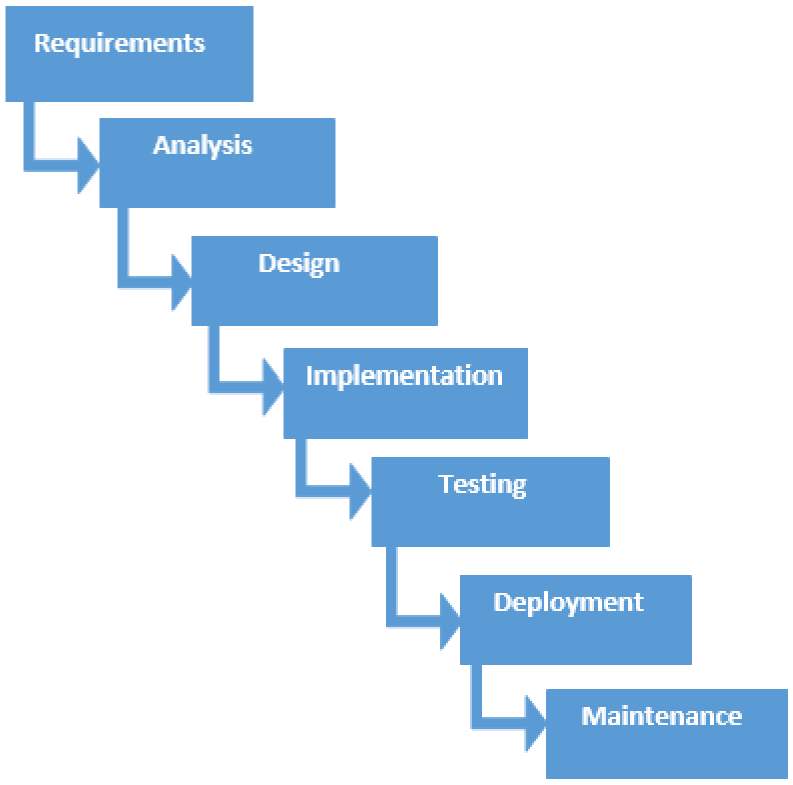
2.2 Popular Project Management Methodologies:

2.2.1 Waterfall:

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. You start by

- collecting and analysing requirements,
- designing the solution (and your approach)
- implementing the solution
- and fixing issues, if any.

Each stage in this process is self-contained; you wrap up one stage before moving onto another.



It is a sequential, linear process of project management. It consists of several discrete phases. No phase begins until the prior phase is complete, and each phase's completion is terminal—waterfall management does not allow you to return to a previous phase. The only way to revisit a phase is to start over at phase one.

It emphasizes that you're only able to move onto the next phase of development once the current phase has been completed. Waterfall is a project management methodology that stresses the importance of documentation. The idea is that if a worker was to leave during the development process, their replacement can start where they left off by familiarizing themselves with the information provided on the documents.

Waterfall project management can be problematic if the project requirements are not perfectly clear, which happens when a user has a general idea of what they want but can't nail down specifics. The waterfall system's linear nature is not suited to discovery, and the project will likely suffer without more specific requirements. If you are certain that the project requirements are static, then waterfall project management provides a straightforward way to push a project through a clearly defined process. It's simple to manage and easy to track.

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 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

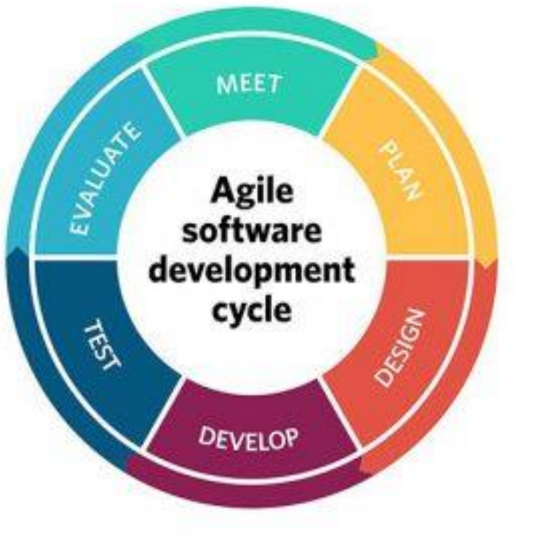
- **Higher risk:** If you find an error or need to change something, you have to essentially start the project from the beginning. This substantially increases the risk of project failure.
- **Front-heavy:** It heavily on your understanding and analysing requirements correctly. Should you fail to do 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

2.2.2 Agile

Agile Methodology is a people-focused, results-focused approach to software development. Agile is best suited for projects that are iterative and incremental. As the name implies, this method favours 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. This methodology rejects sequential phases and relies on simultaneous, incremental work across various departments. Various checkpoints throughout the project allow the team to change direction as needed. By continually taking the temperature of the project throughout the process, you can deliver a better final product.



Team members get feedback on their work and an understanding of how what they created had an impact, providing intrinsic benefit as individuals. Engineering organizations collectively become not only more efficient in their operations but also more effective in their delivery of value. Companies as a whole become more adaptive to changes in their markets and therefore more competitive—leveraging both the increased effectiveness and their newfound responsiveness.

Instead of planning and moving towards a delivery or launch date, the agile methodology breaks the developmental process into iterative steps, allowing for flexibility, testing, and change throughout the lifecycle of the project. It abandons the risk of spending months or years on a process that ultimately fails because of some small mistake in an early phase. It relies instead on trusting employees and teams to work directly with customers to understand the goals and provide solutions in a **fast and incremental way**.

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:

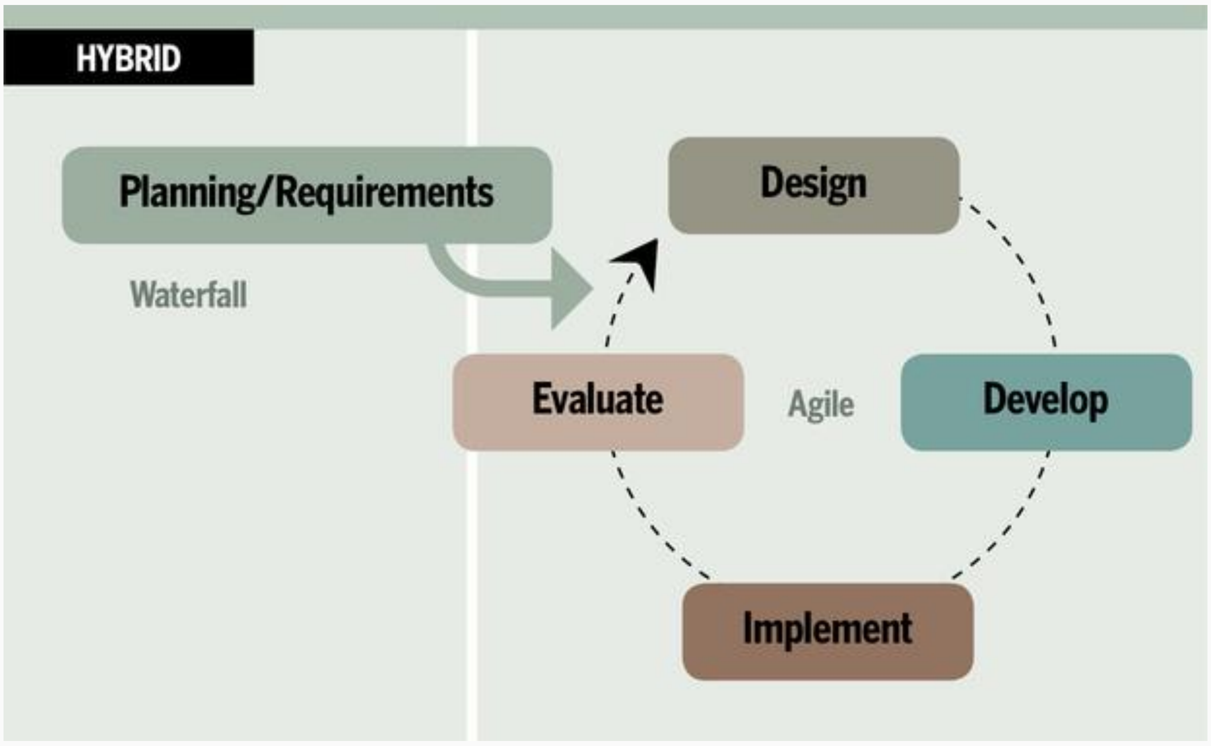
- When you don't have a fixed end in mind but have a general idea of a product.
- When the project needs to accommodate quick changes.
- If collaboration and communication are your key strengths (and planning isn't)

2.2.3 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 analysing requirements initially - a nod to the Waterfall method. From thereon, it takes the flexibility of Agile approach with an emphasis on rapid iterations.

In the real world of project management and development, many projects are not completely black or white. They actually benefit most from a hybrid approach that takes advantage of the strengths of both Agile and Waterfall methodologies without allowing them to get in each other’s way.

In hybrid method, the project is broken down into manageable components either by discipline (hardware, software, mechanical etc) or by functionality. This work breakdown is accomplished by using a process called Work Breakdown Structure or WBS. When multiple disciplines are used in a project, one discipline might use agile (software) and waterfall is used by others (hardware, mechanical, manufacturing etc).



When a project is broken down in terms of functionality, the waterfall is used to map out the path from requirements and specification to development, testing, and final release to the customer. Each component is then specified in more details and developed using agile project management method.

The beauty of the Hybrid project management approach is that all high-level tasks, their interrelationship (dependencies), and final product delivery are defined by Work Breakdown Structure method. Agile is used to speed up the development of each component and its sub-component in the plan. This hybrid approach makes for better quality products with less development time and faster reaction to market changes. After each component of the project is broken down into tasks which may take anywhere from one month to a few months, agile method come into the play. These components are broken down further into four to six weeks product releases called sprints. Here all methods used for agile project management method are applied. The outcome of each sprint is testes and sent either to market (if applicable) or used as the base for the next sprint.

Thus, 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".

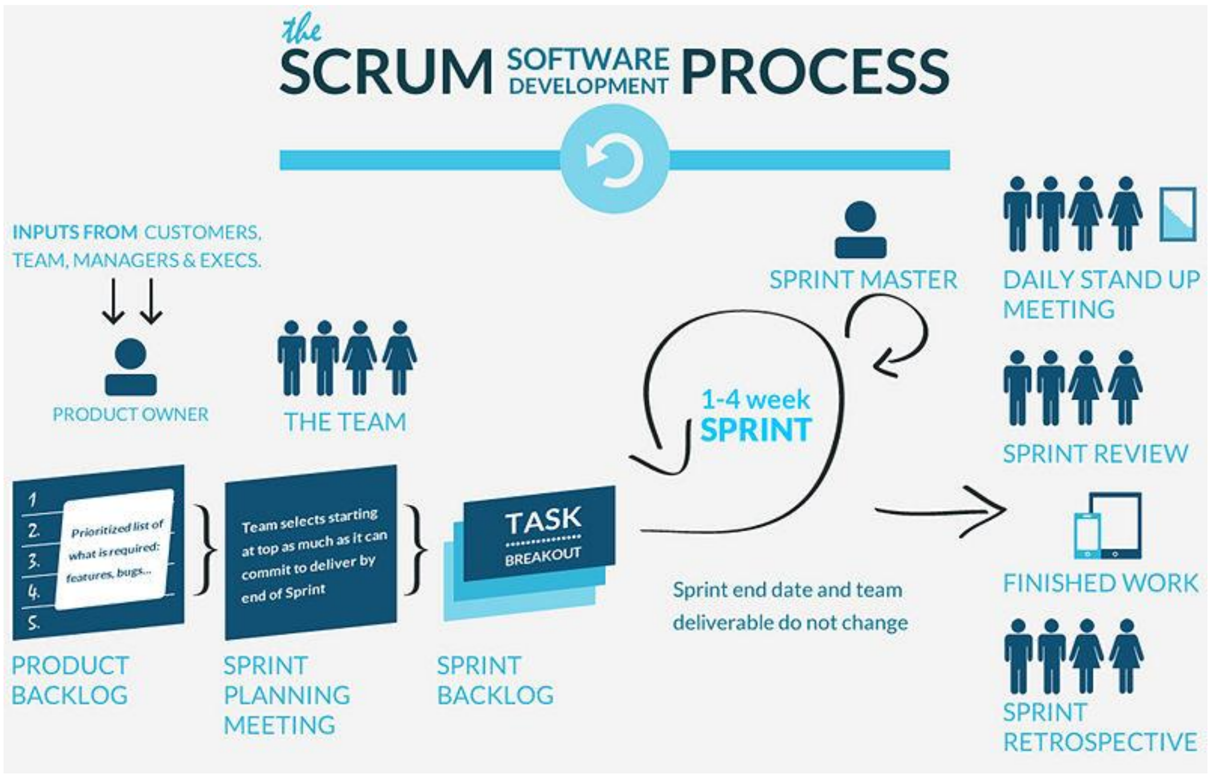
Disadvantages

- **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 surefootedness 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.

2.2.4 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 SOFTWARE DEVELOPMENT PROCESS
INPUTS FROM CUSTOMERS, TEAM, MANAGERS & EXECS.
PRODUCT OWNER
THE TEAM
SPRINT MASTER
DAILY STAND UP MEETING
1-4 week SPRINT
SPRINT REVIEW
Prioritized list of what is required: features, bugs...
Team selects starting at top as much as it can commit to deliver by end of Sprint
TASK BREAKOUT
Sprint end date and team deliverable do not change
FINISHED WORK
PRODUCT BACKLOG
SPRINT PLANNING MEETING
SPRINT BACKLOG
SPRINT RETROSPECTIVE

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.

It’s goal is to develop, deliver, and sustain complex products through collaboration, accountability, and iterative progress. What distinguishes Scrum from the other Agile project management methodologies is how it operates by using certain roles, events, and artifacts.

Scrum team roles

- **Product owner**: Product expert who represents the stakeholders, and is the voice of the customer.
- **Development team**: Group of professionals who deliver the product (developers, programmers, designers).
- **Scrum master**: Organized servant-leader who ensures the understanding and execution of Scrum is followed.

Scrum events

- **Sprint**: Iterative time boxes in which a goal is accomplished. Time frame does not exceed one calendar month and are consistent throughout the development process.
- **Sprint planning**: Where the entire Scrum team get together—at the beginning of every Sprint—to plan the upcoming sprint.
- **Daily Scrum**: 15 minute time boxed meeting held at the same time, every day of the Sprint, where the previous day's achievements are discussed, as well as the expectations for the following one.
- **Sprint review**: An informal meeting held at the end of every Sprint where the Scrum team present their Increment to the stakeholders, and discuss feedback.
- **Sprint retrospective**: A meeting where the Scrum team reflect on the proceedings of the previous Sprint and establish improvements for the next Sprint.

Scrum Artifacts

- **Product backlog**: Managed by the Product Owner, it's where all the requirements needed for a viable product are listed in order of priority. Includes features, functions, requirements, enhancements, and fixes that authorize any changes to be made to the product in future releases.
- **Sprint backlog**: A list of the tasks and requirements that need to be accomplished during the next Sprint. Sometimes accompanied by a Scrum task board, which is used to visualize the progress of the tasks in the current Sprint, and any changes that are made in a 'To Do, Doing, and Done' format.

Advantages

- **Scrum "sprints":** The Scrum approach is heavily focused on 30-day "sprints". This is where the project team breaks down a wishlist 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.

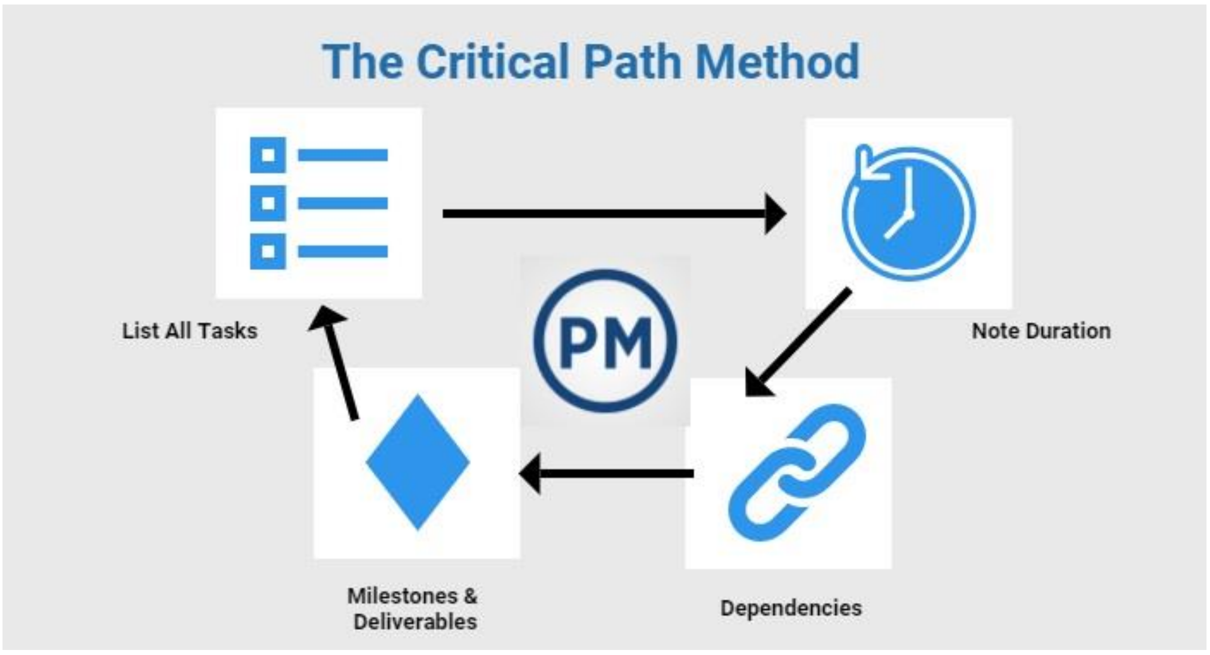
Disadvantages

- **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.

2.2.5 Critical Path Method (CPM)

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.



Advantages

- **Better scheduling:** The emphasis on mapping the duration of activities and their interdependencies help you schedule tasks better. If task X depends on task Y to be finished first, CPM will help you identify and schedule for it.
- **Prioritization:** The success of the CPM methodology depends on identifying and mapping critical and non-critical activities. Once you've mapped these activities, you can prioritize resources better.

Disadvantages

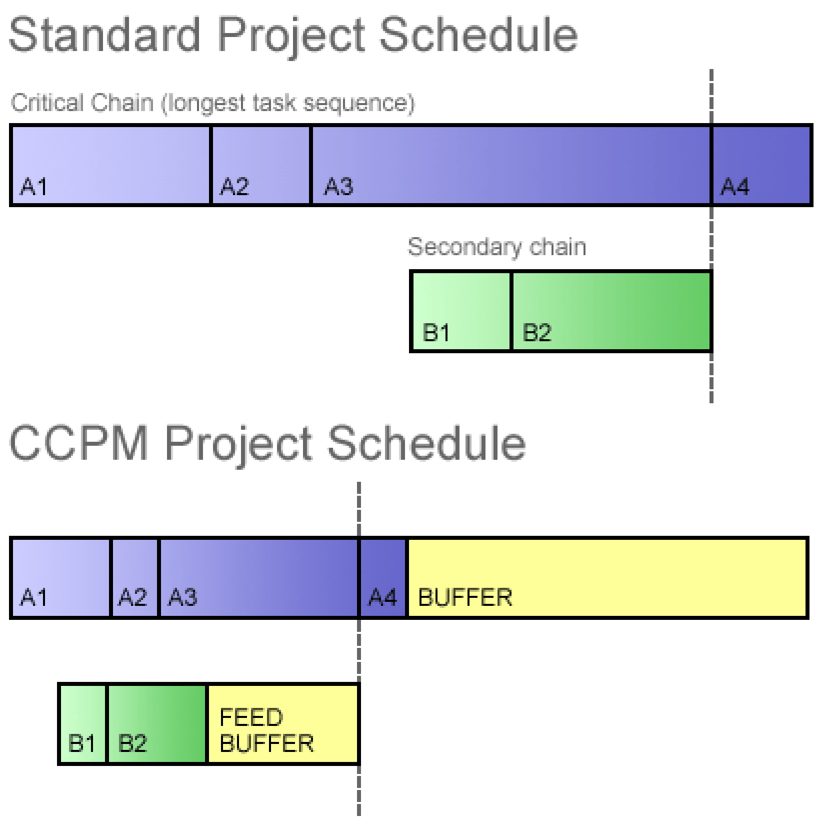
- **Scheduling requires experience:** As any experienced project manager will tell you, things always take more time than you expect. If you don't have real-world experience with scheduling, you are bound to miscalculate time for each activity.
- **No flexibility:** Like the Waterfall method, CPM is 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.

2.2.6 Critical Chain Project Management (CCPM)

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.

CCPM emphasizes resource utilization and minimizing 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.

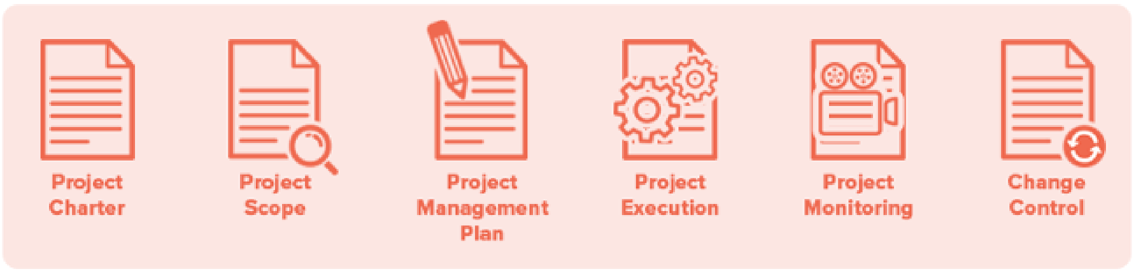
Disadvantages

- **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.

2.2.7 **Integrated Project Management**

Integrated Project_Management (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.
An integrated project has the following components:



By integrating processes across the organization, IPM gives project managers better insight into the project and access to the right resources.
This makes IPM particularly appropriate for creative agencies.

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.

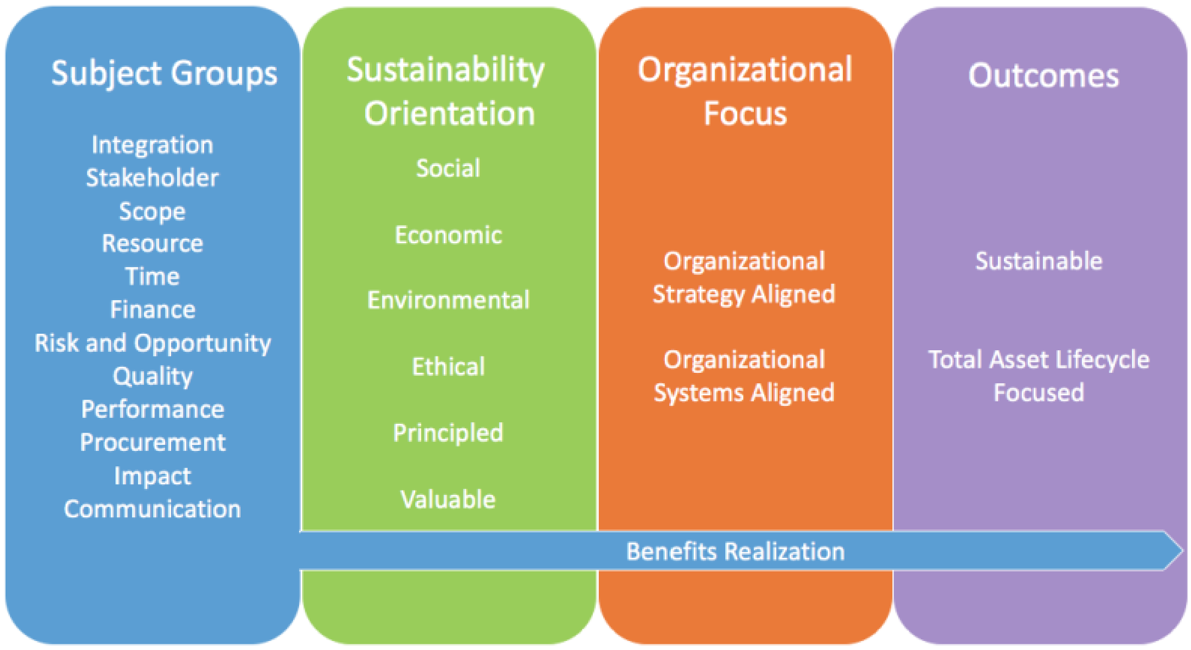
Disadvantages

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.

2.2.8 PRISM

PRiSM (Projects integration Sustainable Methods) is a project management methodology developed by Green Project Management (GPM) Global.This approach focuses on accounting for and minimizing 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.
Here's an overview of how activities are organized in PRiSM:



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

CHAPTER 2
VERSION CONTROL

2.1 What is Version Control?

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.

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.

2.2 Benefits of version control

The primary benefits you should expect from version control are as follows.

- **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. This history should also include the author, date and written notes on the purpose of each change. Having the complete history enables going back to previous versions to help in root cause analysis for bugs and it is crucial when needing to fix problems in older versions of software.

- **Branching and merging.**

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. Many software teams adopt a practice of branching for each feature or perhaps branching for each release, or both.

- **Traceability.**
  Being able to trace each change made to the software and connect it to project management and bug tracking software, 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. Having the annotated history of the code at your fingertips when you are reading the code, trying to understand what it is doing and why it is so designed can enable developers to make correct changes that are in accord with the intended long-term design of the system. This can be especially important for working effectively with legacy code and is crucial in enabling developers to estimate future work with any accuracy.

2.3 Types of version control systems.

There are two types of version control: centralized and distributed.

Centralized version control

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

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.

While you don't have to have a central repository for your files, you may want one "central" place to keep your code so that you can share and collaborate on your project with others. That's where Bitbucket comes in. Keep a copy of your code in a repository on Bitbucket so that you and your teammates can use Git or Mercurial locally and to push and pull code.

CHAPTER 3
GIT

3.1 What is Git?

Git is the most widely used modern version control system in the world today. Having a distributed architecture, Git is an example of a DVCS (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.

3.2 Git Workflow

A 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.

The basic workflow of Git.

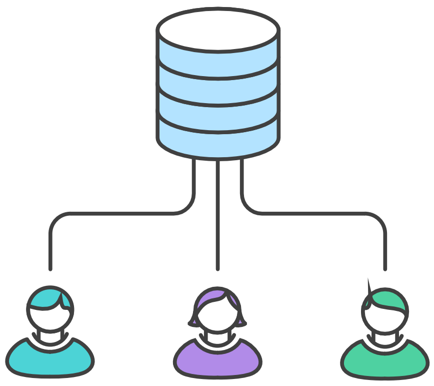
Step 1: You modify a file from the working directory.

Step 2: You add these files to the staging area.

Step 3: You perform commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git repository.

Centralized Workflow:

The Centralized Workflow is a great Git workflow for teams transitioning from SVN. Like Subversion, 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.



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. It 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.

- **Clone the central repository.**

  Next, each developer creates a local copy of the entire project. This is accomplished via the git clone command:

  git clone ssh://user@host/path/to/repo.git

  When you clone a repository, Git automatically adds a shortcut called origin that points back to the "parent" 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.

- **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

  This command will push the new committed changes to the central repository.

- **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.

3.3 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
	Create a working copy	git clone /path/to/repository

Check out a repository	of a local 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	`git checkout -- <filename>`

	last content in head: Changes already added to the index, as well as new files, will be kept.	

	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()"`

CHAPTER 5
CONCLUSION

I hereby conclude this documentation on Project Management Methodologies, Version Control and Git that covers the basics of the topics mentioned.

CHAPTER 6

REFERENCES

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

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

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

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

https://www.lucidchart.com/blog/waterfall-project-management-methodology