UNIT-5: PROJECT MANAGEMENT

Software Project Management- Software Configuration Management - Project Scheduling- DevOps: Motivation-Cloud as a platform-Operations- Deployment Pipeline: Overall Architecture Building and Testing-Deployment- Tools- Case Study

Software Project Management

The project life cycle

- ✓ The project management life cycle provides a structured plan for project managers to guide their projects to successful completion.
- ✓ It includes all the stages needed in a project – from the inception of an idea to the final implementation.
- ✓ These phases break the project down into simpler steps, making it easier for project managers to anticipate what's coming next.
- ✓ Using the right tools and methods also contributes to effective team management throughout the project life cycle.

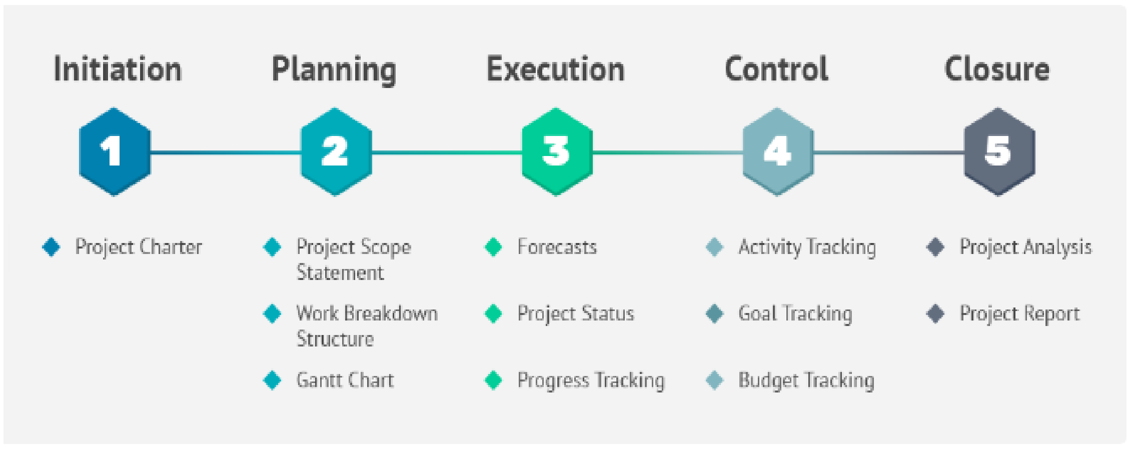
The primary constraints of a project are

- ***Time*** **-** the schedule for the project to reach completion
- ***Cost*** **-** the budget allocated for project to meet its objectives and complete it on time
- ***Scope*** **-** the specific deliverables of the project
- ***Quality*** **-** the standard of the outcome of the project

The 5 project life cycle phases

The project life cycle outlines the different stages a project goes through from start to finish. It encompasses several key phases, each addressing different needs as the project progresses.

- ➢ Project Initiation
- ➢ Project Planning
- ➢ Project Execution
- ➢ Project Monitoring and Control
- ➢ Project Closure



Initiation phase

- ✓ The initiation phase marks the beginning of a project, with the project manager defining the scope and objectives. During this phase, it's vital to align stakeholders on common goals and lay the foundation for a successful project.
- ✓ Next, the project manager creates a project charter, outlining the purpose, goals, and scope of the project.

This charter includes the following key information:

- ➢ Project purpose and justification
- ➢ Main objectives and deliverables
- ➢ Key stakeholders and team members
- ➢ Initial schedule and budget estimates

- ✓ The project manager also conducts a feasibility assessment to determine if the project is realistic and worthwhile.

Planning phase

- ✓ During the planning phase, the project manager develops a detailed project plan and roadmap.
- ✓ This involves determining key scheduling details, resource allocation, and risks that could impact the project.
- ✓ The goal is to create a comprehensive map of how the team will execute the work.

Execution phase

- ✓ During the execution phase, the team puts the project plan into action.
- ✓ The project manager plays a key role in coordinating resources, including people, tools, and materials, while also ensuring the team is well-informed about their individual tasks and timelines.

Monitoring and controlling phase

- ✓ The monitoring and controlling phase involve regularly checking project progress and team performance to ensure everything adheres to the project plan.

- ✓ During this phase, the project manager identifies any deviations from the plan and budget, determining the cause to take corrective action.
- ✓ Tools such as status reports, time tracking, budget reports, risk management plans, and stakeholder reviews make it easy to see the most important metrics and milestones.
- ✓ To make changes to the plan, team members should submit a change request for approval.

Closing phase

- ✓ The closing phase marks the formal end of a project.
- ✓ During this phase, the focus is on getting final approvals and sign-offs, conducting a post-project review, identifying what went well, determining areas for improvement, and documenting lessons learned.
- ✓ These activities foster a culture of continuous learning and promote accountability and transparency.

Importance of Project Phases

All projects go through each of the five phases regardless of their size. Effective project life cycle management streamlines processes in several ways:

- ➢ ***Improved project visibility*** **-** Teams can proactively remove obstacles to ensure timely, high-quality results. This enables more effective decision-making.
- ➢ ***Better risk management*** **-** Teams can spot risks early and find solutions. Regular risk checks ensure projects stay on time and avoid costly delays or failure.
- ➢ ***Enhanced stakeholder communication*** **-** With regular updates, progress reports, and meetings, participants stay more informed and involved throughout the project life cycle.
- ➢ The decision to divide a project into phases is an excellent way to manage the team’s focus, allocate resources, and align the entire project life cycle with clients and stakeholders.
- ➢ Defined activities, outputs, and responsibilities create a clear and common roadmap for the project team to follow.
- ➢ Defined phases and defined roles show a visible framework easily understood by all team members and stakeholders.

COCOMO Model

- ✓ COCOMO (Constructive Cost Model) is a regression model based on LOC, i.e., the **number of Lines of Code**.
- ✓ Software Project Estimation shapes resource allocation, timelines, and costs. COCOMO Model (Constructive Cost Model) is a cornerstone in Software Project Estimation.

- ✓ The COCOMO Model is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality.

The key parameters that define the quality of any software products, which are also an outcome of the COCOMO are primarily Effort and schedule:

1. **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
2. **Schedule:** This simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

Software Project Type

The COCOMO model divides software projects into 3 types

- ✓ Organic
- ✓ Semi-detached
- ✓ Embedded

Organic - A software development project comes under organic type if the development team is small, the problem is not complex, and has been solved before.

- ✓ Also, a nominal experience of team members is considered.
- ✓ Some of the projects that can be organic are small data processing systems, basic inventory management systems, business systems.

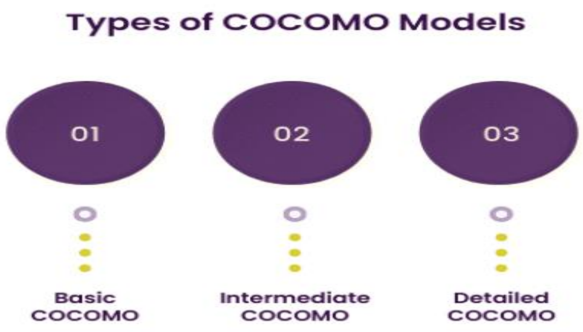
Semidetached - A software project that is in-between the organic and embedded system is a semi-detached system.

- ✓ Its characteristics include a middle-sized development team, with a mix of both-experienced and inexperienced members, and the project complexity is neither too high nor too low.

Embedded - A project requiring a large team with experienced members, and having the highest level of complexity is an embedded system.

- ✓ The members need to be creative enough to be able to develop complex models. Such projects include- air traffic models, ATMs, complex banking systems, etc.

Types of COCOMO Models



Basic COCOMO

- ✓ This Basic COCOMO Model is ideal for early-stage estimates. It estimates effort based solely on the size of the code to be developed. The central parameter here is the Source Lines of Code (SLOC).
- ✓ Basic COCOMO assumes a linear relationship between project size and effort. It's a simplistic yet useful approach for quick ballpark estimates in the initial project stages.

Intermediate COCOMO

- ✓ As projects become more intricate, Intermediate COCOMO introduces influencing factors that impact the estimation process.
- ✓ These factors include product characteristics, platform attributes, personnel skills, project complexity, and development environment.
- ✓ Intermediate COCOMO employs cost drivers to adjust the estimation based on these factors, making it more accurate than the Basic Model.

Detailed COCOMO

- ✓ Designed for complex and large-scale projects, Detailed COCOMO delves even deeper into estimation parameters. It encompasses a comprehensive set of factors, including those from Intermediate COCOMO, and additional considerations such as risk assessment, team cohesion, and Project Management capabilities.
- ✓ Detailed COCOMO employs extensive cost drivers and scales to provide a highly refined and nuanced estimation, making it suitable for projects with intricate requirements and diverse variables.

COCOMO Model Components

Source Lines of Code (SLOC)

- ✓ This fundamental component serves as the foundation for COCOMO's size-based estimation. SLOC quantifies the size of the Software Project by counting the number of lines of code to be written.
- ✓ It directly influences the effort required for development, forming the basis for early-stage estimates in Basic COCOMO.

Effort estimation

- ✓ Estimating effort involves predicting the human resources required to develop the software.

- ✓ It considers factors such as team size, expertise, and skill levels. Effort estimation constitutes the core purpose of COCOMO, ensuring adequate staffing for the project and efficient distribution of tasks.

Duration estimation

- ✓ Duration estimation focuses on predicting the time required to complete the project.
- ✓ It considers the project's complexity, team productivity, and resource allocation.
- ✓ Accurate duration estimation aids in effective project scheduling and timely delivery.

Cost estimation

- ✓ Cost estimation consists of the Software Project's monetary and resource costs.
- ✓ It includes personnel costs, hardware and software costs, overhead expenses, and potential risks.
- ✓ Proper cost estimation aids in budgeting and planning, preventing unexpected financial constraints

Advantages

- ✓ Systematic cost estimation
- ✓ Helps to estimate cost and effort of a software project at different stages of the development process.
- ✓ Helps in high-impact factors
- ✓ Helps to evaluate the feasibility of a project of a software project by estimating the cost and effort required to complete it.

Disadvantages

- ✓ Assumes project size as the main factor of a software project, which may not always be the case.
- ✓ Does not count development team-specific characteristics of the development team, which can have a significant impact on the cost and effort of a software project.
- ✓ Not enough precise cost and effort estimate - This does not provide a precise estimate of the cost and effort of a software project, as it is based on assumptions and averages.

COCOMO 2

- ✓ COCOMO 2 is also a cost estimation model for computing software development cost. This model was developed to overcome the limitations of COCOMO 1 model. The main objective of the COCOMO 2 model is to provide quantitative analytic structure, techniques, and tools.
- ✓ The COCOMO 2 model is based upon the non-linear reuse formula.

- ✓ This cost estimation model helps in providing estimates that represent one standard deviation near to the most likely estimate.
- ✓ COCOMO 2 model is useful in non-sequential, rapid development, reengineering and reuse models of software development cycle.

COCOMO II can be used for the following major decision situations

- ✓ Making investment or other financial decisions involving a software development effort Setting project budgets and schedules as a basis for planning and control
- ✓ Deciding on or negotiating tradeoffs among software cost, schedule, functionality, performance or quality factors
- ✓ Making software cost and schedule risk management decisions
- ✓ Deciding which parts of a software system to develop, reuse, lease, or purchase Making legacy software inventory decisions: what parts to modify, phase out, outsource, etc
- ✓ Setting mixed investment strategies to improve organization's software capability, via reuse, tools, process maturity, outsourcing, etc

COCOMO II Models

- ✓ ***The Application Composition Model*** used for prototyping.
- ✓ ***The Early Design Model*** used when requirements are available but design has not yet started.
- ✓ ***The Reuse Model*** used to compute the effort of integrating reusable components.
- ✓ ***The Post-Architecture Model*** used once the system architecture has been designed.

COCOMO II models require sizing information.

- ✓ Three different sizing options are available as part of the model hierarchy i.e. ***object points, function points, and lines of source codes.***
- ✓ The application composition model uses object points which is an indirect software measure that is computed using counts of the number of screens, reports and components likely to be required to build the application.

Components of COCOMO II

Cost Drivers - COCOMO II considers various cost drivers that influence software development effort and cost. These cost drivers are categorized into different categories such as Product, Platform, Personnel, Project, and Process.

- ✓ Examples of cost drivers include software reliability, complexity, required reusability, analyst capability, team cohesion, and development schedule.

Scale Factors - Scale factors are used to assess the impact of different project characteristics on development effort.

- ✓ These factors include aspects such as product complexity, team experience, development flexibility, and process maturity.

Effort Equation - The effort estimation in COCOMO II is typically calculated using an equation that combines the influence of scale factors, cost drivers, and other project attributes.

- ✓ The effort equation is based on the principles of COCOMO 1 but is more sophisticated and customizable to accommodate a broader range of project scenarios.

Estimation Process - COCOMO II provides guidance on how to collect and analyze project data, select appropriate cost drivers and scale factors, and apply the estimation equations to generate effort, cost, and schedule estimates.

- ✓ The estimation process can be tailored to suit the specific needs and characteristics of individual projects.

Estimation Outputs - The outputs of COCOMO II estimation include estimates of effort (in person-months), duration (in calendar months), staffing levels (number of personnel required), and cost (in currency units).

- ✓ These estimates help project managers and stakeholders make informed decisions regarding project planning, resource allocation, and budgeting.

Advantages

- ✓ **Flexibility -** allows project managers to select the appropriate level of detail based on project requirements and available information.
- ✓ **Comprehensive Estimation -** leads to more accurate and reliable cost and effort estimates compared to simpler estimation models.
- ✓ **Tailored to Modern Practices -** ensures that the model remains relevant and applicable to contemporary software projects.
- ✓ **Calibration and Validation -** COCOMO II has been calibrated using data from a broad range of software projects, making it applicable to different types of projects and industries. Additionally, the model has undergone validation to ensure its accuracy and reliability in estimating software project costs and effort.
- ✓ **Risk Management -** on managing project risks and uncertainties, allowing project managers to identify and mitigate potential risks early in the project lifecycle.

Disadvantage

- ✓ **Complexity -** COCOMO II is a relatively complex model compared to simpler estimation techniques. It requires a good understanding of software development processes and project management concepts to effectively apply. This complexity can sometimes make it challenging for inexperienced project managers to use the model accurately.
- ✓ **Resource Intensive -** Detailed COCOMO, the most comprehensive level of the model, requires a significant amount of data input and computation. Gathering all

the necessary information and performing the calculations can be time-consuming and resource-intensive, especially for large and complex projects.

- ✓ **Data Requirements -** COCOMO II relies heavily on historical data and project-specific information to generate accurate estimates. Obtaining reliable data for all the required parameters can be difficult, particularly for organizations with limited historical project data or for projects with unique characteristics.

Difference between COCOMO I and COCOMO II Model

COCOMO I	**COCOMO II**
It is used in waterfall model of software development cycle.	It is used in non-sequential, rapid development of models.
COCOMO 1 provides a simplistic estimation approach, primarily focusing on LOC as the main input parameter.	COCOMO II is more comprehensive and considers a broader range of factors that can influence software development effort and cost.
It is based on the linear reuse formula.	It is based on the non-linear formula of reuse.
It is based on the assumption of reasonable stable requirements.	It is based on the reuse model that focuses on effort required to understand and estimate.
The development begins after the requirements are assigned to software.	It uses spiral development method.
The size of the software is measured in terms of lines of code.	The size of software is stated in terms of object points, function points and lines of code.
It doesn't consider as many project-specific factors as COCOMO II.	It includes factors related to the software product, hardware attributes, personnel capabilities, project environment, and more.
COCOMO 1 offers less flexibility in terms of adapting to different project environments and software development methodologies.	COCOMO II is more flexible and customizable.
It is primarily suited for projects with straightforward requirements and development processes.	It allows users to tailor the estimation process by selecting appropriate sub-models and adjusting input parameters based on project-specific characteristics and constraints.
COCOMO 1 provides reasonably accurate estimates for simple projects or	COCOMO II offers improved accuracy by considering a broader set of

during the early stages of project planning.	influencing factors and providing more detailed estimation capabilities.
COCOMO 1 is suitable for quick and rough estimates during the early stages of project planning when detailed project information is limited.	COCOMO II is recommended for more detailed and accurate estimation efforts, especially for larger projects or when a more thorough analysis of project attributes is required.

Activities Planning

Software Project Management consists of many activities, that includes planning of the project, deciding the scope of product, estimation of cost in different terms, scheduling of tasks, etc.

The list of activities are as follows:

- Project planning and Tracking
- Project Resource Management
- Scope Management
- Estimation Management
- Project Risk Management
- Scheduling Management
- Project Communication Management
- Configuration Management

The list of these activities -

1. Planning - It is a set of multiple processes, or we can say that it a task that performed before the construction of the product starts.

2. Scope Management - It describes the scope of the project. Scope management is important because it clearly defines what would do and what would not. Scope Management create the project to contain restricted and quantitative tasks, which may merely be documented and successively avoids price and time overrun.

3. Estimation management - This is not only about cost estimation because whenever we start to develop software, but also figure out their size(line of code), efforts, time as well as cost.

- ✓ About the size, then Line of code depends upon user or software requirement.
- ✓ About effort, we should know about the size of the software, because based on the size we can quickly estimate how big team required to produce the software.
- ✓ About time, when size and efforts are estimated, the time required to develop the software can easily determine.
- ✓ About cost, it includes all the elements such as:

- Size of software
- Quality
- Hardware
- Communication
- Training
- Additional Software and tools
- Skilled manpower

4. Scheduling Management - Scheduling Management in software refers to all the activities to complete in the specified order and within time slotted to each activity. Project managers define multiple tasks and arrange them keeping various factors in mind.

For scheduling, it is compulsory -

- Find out multiple tasks and correlate them.
- Divide time into units.
- Assign the respective number of work-units for every job.
- Calculate the total time from start to finish.
- Break down the project into modules.

5. Project Resource Management - In software Development, all the elements are referred to as resources for the project. It can be a human resource, productive tools, and libraries.

Resource management includes:

- Create a project team and assign responsibilities to every team member
- Developing a resource plan is derived from the project plan.
- Adjustment of resources.

6. Project Risk Management - Risk management consists of all the activities like identification, analyzing and preparing the plan for predictable and unpredictable risk in the project.

Several points show the risks in the project:

- The Experienced team leaves the project, and the new team joins it.
- Changes in requirement.
- Change in technologies and the environment.
- Market competition.

7. Project Communication Management - Communication is an essential factor in the success of the project. It is a bridge between client, organization, team members and as well as other stakeholders of the project such as hardware suppliers.

From the planning to closure, communication plays a vital role. In all the phases, communication must be clear and understood. Miscommunication can create a big blunder in the project.

8. Project Configuration Management - Configuration management is about to control the changes in software like requirements, design, and development of the product.
The Primary goal is to increase productivity with fewer errors.
Reasons show the need for configuration management:

- ➢ Several people work on software that is continually update.
- ➢ Help to build coordination among suppliers.
- ➢ Changes in requirement, budget, schedule need to accommodate.
- ➢ Software should run on multiple systems.

Tasks perform in Configuration management:

- ➢ Identification
- ➢ Baseline
- ➢ Change Control
- ➢ Configuration Status Accounting
- ➢ Configuration Audits and Reviews

Software Configuration Management

- ✓ In Software Engineering, **Software Configuration Management (SCM)** is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.
- ✓ The primary goal is to increase productivity with minimal mistakes. SCM is part of cross-disciplinary field of configuration management and it can accurately determine who made which revision.

Need of Software Configuration management

- ✓ Any change in the software configuration Items will affect the final product. Therefore, changes to configuration items need to be controlled and managed.
- ✓ There are multiple people working on software which is continually updating
- ✓ It may be a case where multiple versions, branches, authors are involved in a software config project, and the team is geographically distributed and works concurrently
- ✓ Changes in user requirement, policy, budget, schedule need to be accommodated.
- ✓ Software should able to run on various machines and Operating Systems
- ✓ Helps to develop coordination among stakeholders
- ✓ SCM process is also beneficial to control the costs involved in making changes to a system.

Tasks in SCM process

1. Planning and Identification

2. Version Control and Baseline
3. Change Control
4. Configuration Status Accounting
5. Audits and Reviews

1. Planning and Identification

- ✓ The first step in the process is planning and identification. In this step, the goal is to plan for the development of the software project and identify the items within the scope.
- ✓ This is accomplished by having meetings and brainstorming sessions with the team to figure out the basic criteria for the rest of the project.
- ✓ Part of this process involves figuring out how the project will proceed and identifying the exit criteria. This way, the team will know how to recognize when all of the goals of the project have been met.

Specific activities during this step include:

- ➢ Identifying items like test cases, specification requirements, and code modules
- ➢ Identifying each computer software configuration item in the process
- ➢ Group basic details of why, when, and what changes will be made and who will be in charge of making them
- ➢ Create a list of necessary resources, like tools, files, documents, etc.

2. Version Control and Baseline

- ✓ The version control and baseline step ensure the continuous integrity of the product by identifying an accepted version of the software.
- ✓ This baseline is designated at a specific time in the SCM process and can only be altered through a formal procedure.
- ✓ The point of this step is to control the changes being made to the product. As the project develops, new baselines are established, resulting in several versions of the software.

This step involves the following activities

- ➢ Identifying and classifying the components that are covered by the project
- ➢ Developing a way to track the hierarchy of different versions of the software
- ➢ Identifying the essential relationships between various components
- ➢ Establishing various baselines for the product, including developmental, functional, and product baselines
- ➢ Developing a standardized label scheme for all products, revisions, and files so that everyone is on the same page.

  - ➢ Baselining a project attribute forces formal configuration change control processes to be enacted in the event that these attributes are changed.

3. Change Control

- ✓ Change control is the method used to ensure that any changes that are made are consistent with the rest of the project. Having these controls in place helps with quality assurance, and the approval and release of new baseline(s).
- ✓ Change control is essential to the successful completion of the project.
- ✓ In this step, requests to change configurations are submitted to the team and approved or denied by the software configuration manager. The most common types of requests are to add or edit various configuration items or change user permissions.

This procedure includes

  - ➢ Controlling ad-hoc changes requested by the client
  - ➢ Checking the merit of the change request by examining the overall impact they will have on the project
  - ➢ Making approved changes or explaining why change requests were denied.

4. Configuration Status Accounting

- ✓ The next step is to ensure the project is developing according to the plan by testing and verifying according to the predetermined baselines.
- ✓ It involves looking at release notes and related documents to ensure the software meets all functional requirements.
- ✓ Configuration status accounting tracks each version released during the process, assessing what is new in each version and why the changes were necessary.

Some of the activities in this step include

  - ➢ Recording and evaluating changes made from one baseline to the next
  - ➢ Monitoring the status and resolution of all change requests
  - ➢ Maintaining documentation of each change made as a result of change requests and to reach another baseline
  - ➢ Checking previous versions for analysis and testing.

5. Audits and Reviews

- ✓ The final step is a technical review of every stage in the software development life cycle. Audits and reviews look at the process, configurations, workflow, change requests, and everything that has gone into developing each baseline throughout the project’s development.

- ✓ The team performs multiple reviews of the application to verify its integrity and also put together essential accompanying documentation such as release notes, user manuals, and installation guides.

Activities in this step include

- ➢ Making sure that the goals laid out in the planning and identification step are met
- ➢ Ensuring that the software complies with identified configuration control standards
- ➢ Making sure changes from baselines match the reports
- ➢ Validating that the project is consistent and complete according to the goals of the project.

Advantages of SCM

- ✓ Configuration management allows all aspects of an existing system to adapt to new update specifications in a coordinated manner that doesn't interrupt program functions and upgrades it to the latest version.
- ✓ SCM tools work as a commutation tool and can provide alerts and reports if there are any deviations from the agreed-upon baseline.
- ✓ SCM provides proper monitoring and auditing of the software development product and helps to enhance the software development life-cycle process.

Disadvantages of SCM

- ✓ Not feasible for small-scale industries and projects to adapt software configuration management tools and the process to an extensive level.
- ✓ The SCM process requires fast and highly configured hardware for the development stages, which may come in as a shortcoming for some business setups.
- ✓ The team working on the SCM process needs to have full knowledge of the software configuration management tools to avoid delays and mistakes.

Cost-Benefit Analysis

- ✓ A cost-benefit analysis before investing organizational time and resources into a new project or business proposal can make the difference between eventual success and failure.
- ✓ A cost-benefit analysis (CBA), sometimes referred to as benefit-cost analysis (BCA), makes it clear what projects or investments are most viable, possible, and beneficial for an organization at any given time.

Cost-Benefit Analysis in Project Management

- ✓ A cost-benefit analysis in project management is a tool to evaluate the costs vs. benefits of an important project or business proposal. It is a practical, data-driven approach for guiding organizations and managers in making solid investment decisions. It helps determine if a project or investment is financially feasible and beneficial for the organization.
- ✓ A formal CBA identifies and quantifies all project costs and benefits, then calculates the expected return on investment (ROI), internal rate of return (IRR), net present value (NPV), and payback period. The difference between the costs and the benefits of moving forward with the project is then calculated.

In a CBA, costs may include the following

- ➢ **Direct costs -** These are costs that are directly related to the proposed project or investment, e.g., materials, labor, and equipment.
- ➢ **Indirect costs -** These are related fixed costs that contribute to bringing the project or investment to life, e.g., overhead, administrative, or training expenses.
- ➢ **Opportunity costs** - These are the benefits or opportunities foregone when a business chooses one project or opportunity over others. To quantify opportunity costs, it must weigh the potential benefits of the available alternatives.
- ➢ **Future costs -** These are costs that may come up later in the project. These costs depend on certain factors happenings, e.g., costs of mitigating potential risks.

Cost-benefit analysis facilitates a structured cost management process, helping project managers and company executives prioritize projects and allocate resources effectively to achieve the organization's main goals.

Benefits may include

- ✓ ***Tangible benefits* -** These are measurable outcomes that can be easily quantified in monetary terms, e.g., increased revenue or reduced costs.
- ✓ ***Intangible benefits* -** These benefits are difficult to measure in monetary terms. They are indirect or qualitative outcomes, such as improved customer satisfaction or increased employee morale.

Importance of cost-benefit analysis

- ✓ **Identify project costs and benefits -** A project cost-benefit analysis ensures all costs and benefits associated with a project are identified and quantified.
- ✓ **Provide a framework for analysis -** This helps ensure all factors are considered and the most optimal decision is made from a business perspective.
- ✓ **Make better-informed decisions -** comparing the costs and benefits of the project, decision-makers make better-informed decisions and allocate organizational resources effectively.

- ✓ **Promote transparency -** This can help ensure that decision-making is objective and all stakeholders have a clear understanding of the project's potential impacts.
- ✓ **Facilitate communication:** A cost-benefit analysis in project management facilitates trust and a foundation for communication between various stakeholders by providing a common language and framework for analyzing a potential project.

Project Scheduling

Critical Path in Project Management

- ✓ In project management, the critical path is the longest sequence of tasks that must be completed to execute a project.
- ✓ The tasks on the critical path are called critical activities because if they're delayed, the whole project completion will be delayed. To find the critical path, project managers use the critical path method (CPM).

The Critical Path Method (CPM)

- ✓ The critical path method (CPM) is a project management technique that's used by project managers to create an accurate project schedule.
- ✓ The CPM method, also known as critical path analysis (CPA), consists in using the CPM formula and a network diagram to visually represent the task sequences of a project.
- ✓ Once these task sequences or paths are defined, their duration is calculated to identify the critical path.

The Importance of CPM in Project Management

- ✓ Identify task dependencies, resource constraints and project risks
- ✓ Accurately estimate the duration of each task
- ✓ Prioritize tasks based on their float or slack time, which helps with project scheduling and resource allocation
- ✓ Identify critical tasks that have no slack and ensure those are completed on time
- ✓ Monitor the project progress and measure schedule variance
- ✓ Use schedule compression techniques like crash duration or fast tracking

Critical Path Method (CPM) Formula

The concept involved in CPM,

- ✓ ***Earliest start time (ES)*** **-** This is simply the earliest time that a task can be started in the project. It cannot determine this without first knowing if there are any task dependencies
- ✓ ***Latest start time (LS)*** **-** This is the very last minute in which, it can start a task before it threatens to delay the project timeline

- ✓ ***Earliest finish time (EF)* -** The earliest an activity can be completed, based on its duration and its earliest start time
- ✓ ***Latest finish time (LF)* -** The latest an activity can be completed, based on its duration and its latest start time
- ✓ ***Float* -** Also known as slack, float is a term that describes how long it can delay a task before it impacts its task sequence and the project schedule. The tasks on the critical path have zero float because they can't be delayed

Parts of CPM

The critical path method formula has two parts; *a forward pass and a backward pass.*

- ➢ **Forward Pass in CPM**
  - ✓ Use the CPM diagram and the estimated duration of each activity to determine their *earliest start (ES) and earliest finish (EF).*
  - ✓ The ES of an activity is equal to the EF of its predecessor, and its EF is determined by the CPM formula,

    EF = ES + t (t is the activity duration)
  - ✓ The EF of the last activity identifies the expected time required to complete the entire project.
- ➢ **Backward Pass in CPM**
  - ✓ Begins by assigning the last activity's earliest finish as its latest finish.
  - ✓ Then the CPM formula to find the LS is

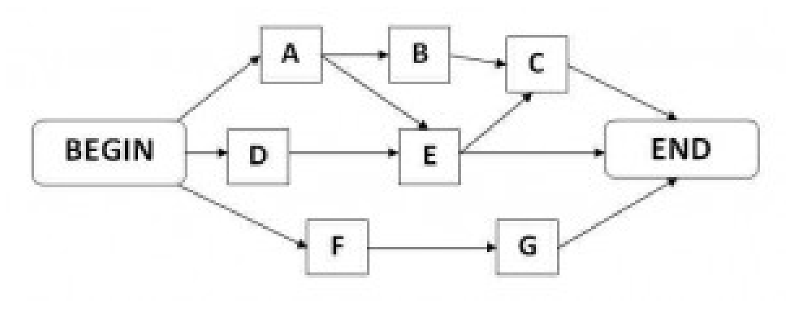
    LS = LF – t (t is the activity duration)
  - ✓ For the previous activities, the LF is the smallest of the start times for the activity that immediately follows.

Activity-on-Node

- ✓ Activity-on-node is a project management term that refers to ***a precedence diagramming method*** which uses boxes to denote schedule activities.
- ✓ These various boxes or "nodes" are connected from beginning to end with arrows to depict a logical progression of the dependencies between the schedule activities.
- ✓ Each node is coded with a letter or number that correlates to an activity on the project schedule.

Typically, an activity-on-node diagram will be designed to show which activities must be completed in order for other activities to commence***. This is referred to as "finish-to-start" precedence – meaning one activity must be finished before the next one can start.***

In the diagram below, activities A and D must be done so that activity E can begin. It is also possible to create other variations of this type of diagram. For example, a "start-to-start” diagram is one in which a ***predecessor activity*** must simply be started rather than fully completed in order for the ***successor activity*** to be initiated.



An activity-on-node diagram can be used to provide a visual representation of *the network logic* of an entire project schedule. Or, it can be used for any smaller section of the schedule that lends itself to being represented as having a defined beginning and end.

To keep the logic in the diagram simple, it may be most effective to include ***only critical path*** schedule activities. The planned start date of each node may also be listed in the diagram legend in accordance with the project management timeline.

DevOps

- ✓ DevOps is a software engineering practice that is suited to cloud computing. In a DevOps environment, developers collaborate with IT operations and other teams.
- ✓ DevOps goes beyond continuous integration and continuous delivery (CI/CD) to enable near-instantaneous deployment of products and services in the cloud.
- ✓ The agility and flexibility of the DevOps framework enables organizations to adapt quickly to marketplace demands and maintain a competitive edge.

What Is DevOps

DevOps is a cross-disciplinary practice where application development (dev) works together with IT operations (ops) to improve product quality and accelerate time to market.

How Does It Work

- ✓ In a DevOps framework, developers and IT managers collaborate with experts in quality control, security, and support.
- ✓ The goal of this cooperative effort it to deliver code rapidly, in a seamless loop of *continuous integration and continuous delivery (CI/CD).*

- ✓ DevOps builds on the frequent releases and CI/CD aspects of agile development methods but adds infrastructure management into the development process to make product delivery more flexible and dynamic.
- ✓ That process integration, in turn, depends on the ongoing collaboration among teams, including business unit management, developers, quality assurance, and IT operations. A DevOps approach replaces silos, rigid job descriptions, and bottlenecks with a flexible, cross-disciplinary model that enables innovation and continuous improvement.

Important Guidelines in DevOps

- ✓ **Clarify roles and responsibilities -** As developers work more closely with operations teams, “it’s time to understand who does what, plus identify any skill gaps,”
- ✓ **Define working agreements –** It work styles and methods be defined up front. Examples might include a choice of project tracking tools or a pledge to limit after-hours communication.
- ✓ **Anticipate risks -** Analyze gaps and develop a contingency plan before beginning a new project so DevOps team members are prepared for issues that might arise.
- ✓ **Conduct regular reviews -** Engage the team in a retrospective analysis to encourage continuous improvement.

These strategies should be repeated with each new project and whenever there is a change in personnel that affects the DevOps team.

Cloud DevOps

DevOps is a software development approach that combines cultural principles, tools, and practices to increase the speed and efficiency of an organization’s application delivery pipeline. It allows development and operations (DevOps) teams to deliver software and services quickly, enabling frequent updates and supporting the rapid evolution of products.

Cloud computing is a powerful technology that helps organizations implement DevOps strategies, enabling the crucial cultural and technological transformation needed to compete in the modern software marketplace.

There are three important ways DevOps and cloud work together,

1. **DevOps leverages the cloud -** DevOps organizations manage and automate infrastructure using cloud computing technology, enabling agile work processes.

2. **CloudSecOps** (cloud security operations) - an organizational pattern that moves processes to the cloud while tightly integrating security into the entire development lifecycle. It's like DevSecOps, only in the cloud.
3. **DevOps as a Service -** delivering integrated continuous integration and continuous delivery (CI/CD) pipelines via the cloud, in a software as a service (SaaS) model, to make DevOps easier to adopt and manage in an organization.

DevOps and Cloud work together

1. Cloud is leveraged by DevOps

- ✓ Organizations utilizing DevOps manage and automate their infrastructure using Cloud Computing technologies. This enables them to follow an agile approach to different work processes.
- ✓ DevOps processes are said to be very agile when implemented efficiently, but they can face limitations of an on-premise environment every now and then.

Cloud DevOps solutions are often said to be more cost-effective than on-prem automation solutions. They facilitate governance by unifying the different environments and reducing the security burden on teams. This high-level cloud-based DevOps automation helps minimize human error and streamline repeatable processes and tasks.

2. CloudSecOps (Cloud Security Operations)

- ✓ SecOps or Security Operations is a collaboration between IT security and IT operations. It integrates processes, tools and technology to keep an enterprise secure along with reducing any associated risks.
- ✓ **Cloud security operations** or **CloudSecOps** is an evolution of SecOps that aims to identify, respond, and recover systems from attacks targeting an organization's cloud assets.

CloudSecOps teams have several roles. Some of them are:

- ➢ **Incident management -** Identifies security incidents, responds to them, and coordinates the response with legal and communication teams.
- ➢ **Event Prioritization -** Prioritizing events based on risk score calculation for cloud systems, accounts, and devices, and identifying cloud data sensitivity.
- ➢ **Threat hunting -** Discovering advanced security threats with the help of tools that filter out the noise from security monitoring solutions enabling advanced investigation of data.

3. DevOps as a Service

- ✓ **DevOps as a Service** is a set of cloud-based tools that enables collaboration between the development and operations teams of an organization.
- ✓ This toolset covers all the relevant DevOps processes and provides them as a unified platform where the teams select the tools they want for each purpose.

- ✓ **DevOps as a Service platform** enables organizations to set up continuous integration / continuous delivery (CI/CD) pipelines via the cloud to increase development velocity and provide continuous feedback to the developers.
- ✓ This way the team can access any relevant technology without having to find, adopt, and learn multiple tools.

Approaches in Cloud DevOps

- ✓ **Continuous Integration/Continuous Delivery is a MUST -** CI/CD pipeline is the heart of the DevOps philosophy where CI means building and regularly validating the project by updating code changes at regular intervals and CD means automatically deploying code in the production environment.
- ✓ **Test for Performance -** Taking advantage of automated performance testing will allow to test application's performance at regular intervals. This will increase the chances of issues getting noticed earlier and improving the performance as well.
- ✓ **Integrating Containers -** Containers allow to compartmentalize applications. This means one can work on certain areas without worrying about the impact on others. This makes the applications easy to deploy, and easy to update as well.
- ✓ **Communication and Collaboration -** Every team needs to be on board with the project and stay informed. Additionally, there needs to be a forum to share useful feedback.
- ✓ **Continuous Monitoring and Logging -** All system activities must be monitored and recorded to ensure that the events get triggered at the right time. Continuous Montoring correct anything that isn't performing well.
- ✓ **Invest in Infrastructure -** Having a good infrastructure is an essential part for streamlining DevOps processes. This will save a lot of time and money in the long run.

Cloud Operations (CloudOps)

- ✓ **Cloud Operations** (**CloudOps**) is the practice of managing delivery, tuning, optimization, and performance of workloads and IT services that run in a cloud environment including multi, hybrid, in the data centre and at the edge.
- ✓ CloudOps codifies procedures and best practices for cloud-based operational processes, much as DevOps codifies the same for application development and delivery processes.

Cloud Operations relies heavily on analytics to enhance visibility of elements of the cloud environment, providing the insight needed to control resources and run services efficiently.

Benefits of cloud operations

- ✓ **Accelerated automation -** CloudOps tools can simplify automated tasks including storage, application testing, monitoring and reporting, application builds, and security.
- ✓ **Enhanced security –** Cloud providers utilize the industry's best physical security and constantly monitor their infrastructure to help prevent against cybercrime or data exfiltration.
- ✓ **Improved RPO/RTO -** Backup and replication strategies can lower the recovery point objective (RPO) and recovery time objective (RTO) to near-zero, helping to ensure availability around the clock for an increasingly global user and customer base.

  On-demand scalability - Seasonal or growth demands can be met with instant capacity increases, usually via self-service portal.
- ✓ **Anywhere access -** Organizations can access cloud applications and resources any time, using nearly any device, from anywhere with an internet connection.

Difference between DevOps and CloudOps

- ✓ DevOps practices deliver continues improvements in processes that enhance collaboration which leads to enhanced visibility throughout the software delivery lifecycle (SDLC) and helps reduce incidents that can disrupt IT operations or impact development schedules.
- ✓ DevOps improvements can bubble throughout the organization, helping to bring more reliable software applications to fruition faster, which leads to improved performance for the organization as a whole. Ultimately, DevOps helps improve the user experience for employee and customers alike.
- ✓ CloudOps encompasses cloud platform engineering principles, combining elements of cloud architecture, IT operations, application development, security, and regulatory compliance to enable organizations to manage cloud-based applications and services.

This enables organizations to

- ✓ Ensure the cloud platform – including hybrid and edge components – operates as a single platform
- ✓ Optimize application performance for a dispersed workforce regardless of access device
- ✓ Ensure reliability and that SLAs are met
- ✓ Maintain backups for disaster recovery and business continuity
- ✓ Automate repetitive services and configuration management
- ✓ Ensure data and applications are secure end-to-end

Approaches For Cloud Operations

- ✓ **Establish a migration strategy -** Each workload will have its own requirements, and the adoption of containerized applications and micro-services can put additional constraints on the way particular solutions are architected.
- ✓ **Include all stakeholders -** Cloud migration is change, and many organizations and departments are change-averse. Every stakeholder from users to top executives should be involved in migration planning to help ensure that business-critical processes do not fall through the cracks during migration.
- ✓ **Emphasize security -** Start by adopting a zero-trust approach to security, end-to-end encryption, and automating security monitoring and remediation to help ensure that little problems never have the opportunity to become expensive data breaches.
- ✓ **Automate to accelerate -** Adopt agile cloud workflows and non-disruptive automation tools including as many self-service capabilities including provisioning and password resets.
- ✓ **Include training in the plan.** Cloud management can require a vastly different skill set from on-premises data centers. The need for physical equipment maintenance vanishes to be replaced with new troubleshooting, provisioning, and deployment skills.
- ✓ **Start small.** Find an application to migrate that can provide proof-of-concept for operations and user teams alike and can demonstrate to all stakeholders the viability of a wide-scale cloud migration.

Importance Of Cloud Operation

DevOps and CloudOps teams can coexist and share best practices, since they both promote:

- ✓ Improved efficiency and utilization of cloud resources
- ✓ Growth of agile work environment for cloud workloads
- ✓ Automation of security and availability processes to support 24/7 operations
- ✓ Improved customers user experience
- ✓ Lowered overall costs of delivery cloud services
- ✓ Enhanced productivity of teams working with migrated applications

---

Deployment in DevOps

- ✓ Deployment in DevOps is a process that enables to retrieve important codes from version control so that they can be made readily available to the public and they can use the application in a ready-to-use and automated fashion.
- ✓ Deployment tools DevOps comes into play when the developers of a particular application are working on certain features that they need to build and implement in the application. It is a very effective, reliable, and efficient means of testing and deploying organizational work.

- ✓ Continuous deployment tools in DevOps simply mean updating the required codes on a particular server. There can be multiple servers and need the required number of tools to continuously update the codes and refresh the website.

The functionality of the DevOps **continuous deployment** tools can be explained as follows:

1. In the first phase of testing, the DevOps codes are merged for internal testing.
2. The next phase is staging where the client's test takes place as per their requirements.
3. Last but not least the production phase makes sure that any other feature does not get impacted because of the updating these codes on the server.

DevOps Deployment Tools

DevOps tools make it convenient and easier for companies to reduce the probability of errors and maintain continuous integration in operations. It addresses the key aspects of a company. DevOps tools automate the whole process and automatically build, test, and deploy the features.

DevOps tools make the whole deployment process with the following aspects

- Increased development.
- Improvement in operational efficiency.
- Faster release.
- Non-stop delivery.
- Quicker rate of innovation.
- Improvement in collaboration.
- Seamless flow in the process chain.

Examples of Deployment DevOps Tools

1. ***AWS CodeDeploy* -** It is a tool that is automated and works for deployment that is exclusively offered by Amazon. It is used for releasing new features without any Hassle.
2. ***Jenkins* -** It is a world-famous open-source automation server that is put down in Java. It is a server-based system that contains multiple dashboards.
3. ***DeployBot* -** It is used for building connections with any Git Repository so that automatic or manual deployments can take place. It can also be deployed through Slack.

DevOps Deployment Tools Key Features

- ✓ Continuous integration for improved software quality.
- ✓ Continuous delivery for software development.

- ✓ Use of infrastructure as code for various software development techniques.
- ✓ Establishes a proper chain of communication so that the information is communicated to the required people.
- ✓ It is an open and broad integration that includes various testing frameworks, communication platforms, and project management systems.
- ✓ It provides flexibility for adding any new features to particular software or applications according to the needs of the users.
- ✓ It has greater scalability and provides easy-to-implement techniques.

Use of DevOps Deployment Tools

- ✓ **Speed -** Tools for continuous deployment including DevOps offer high-velocity speed so that it can get a Grasp of the customers and provide them faster services.
- ✓ **Faster delivery -** It helps to fix bugs quickly and add new features to improve customer experience.
- ✓ **Continuous integration -** Continuous integration and faster delivery allow the developers to automate the process of releasing the software. It will start from the build and end at deployment.
- ✓ **Less scope of errors -** DevOps deployment tools are generally based on automated services and manual systems are less. It decreases the scope of errors, resulting in fewer failures.
- ✓ **Increased reliability -** With continuous improvement, the quality of the applications and the changes in the structure of the same area for better customer experience.
- ✓ **Improved collaboration -** DevOps build and deployment tools constitute an effective team for software and application development. It builds a cultural model that prioritizes accountability and ownership.

DevOps Pipeline

- ✓ *A DevOps pipeline is a set of automated processes and tools that allows developers and operations professionals to collaborate on building and deploying code to a production environment.*
- ✓ It typically includes build automation/continuous integration, automation testing, validation, and reporting.
- ✓ It may also include one or more manual gates that require human intervention before code is allowed to proceed.
- ✓ Continuous is a differentiated characteristic of a DevOps pipeline. *This includes continuous integration, continuous delivery/deployment (CI/CD), continuous feedback, and continuous operations.*

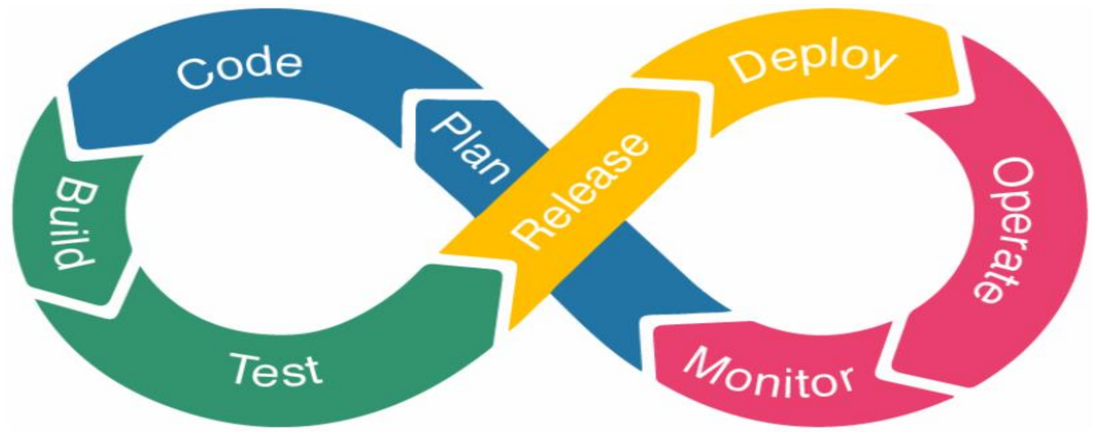
DevOps Pipeline Phases

A DevOps pipeline is a set of practices, tools, and automated processes used by development (Dev) and operations (Ops) teams to build and deploy code.

- ***The development phase*** consists of writing project code, testing, fixing bugs, building new features, conducting updates, and patching code. This phase has four steps: Plan, Code, Build, and Test. Here, developers use many tools to simplify the complexities of the development phase.
- ***The Operations phase*** is typically four steps: Release, Deploy, Operate, and Monitor. These steps can be just as complex as the development phase, with a lot going into seamlessly moving a software development project and any updates to production.

DevOps Pipeline Stages

Dev and Ops - to help illuminate how they work together as one process. It's also important to note that this entire process is iterative and can expect to repeat these stages many times over.



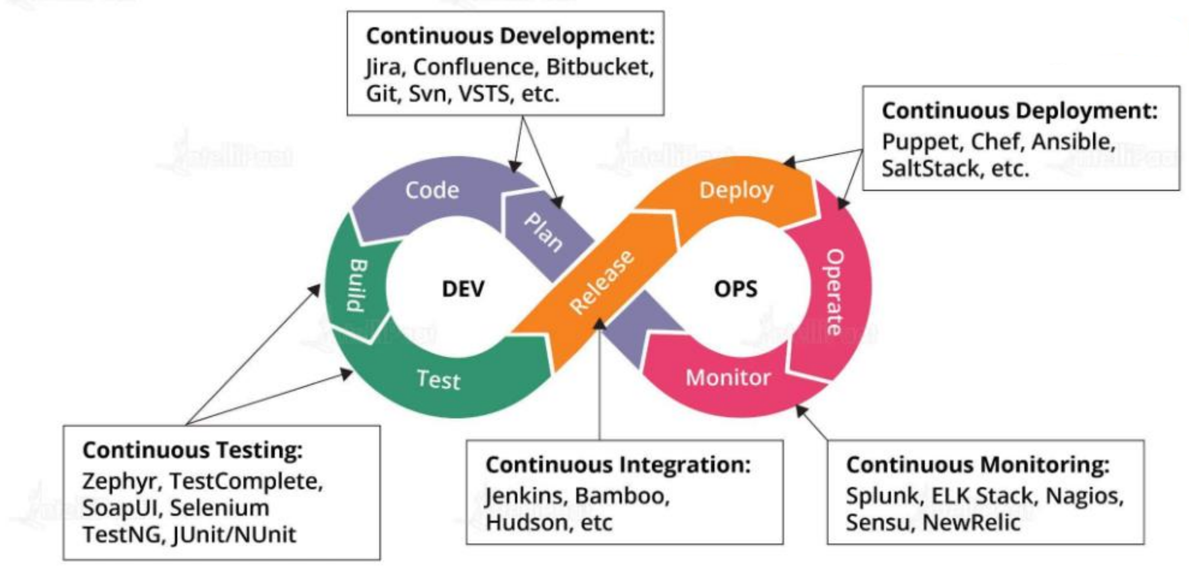
Development Stages

- ✓ **Plan -** Planning the project, technology, environment, structure, and architecture creates a roadmap to successfully reaching the project goals. Planning is also the stage where to decide what software and what tools will.
- ✓ **Code -** In this stage, start writing code for the project. Effectively, getting ready to build a testable product. However, since writing code can take a lot of time, this is a prime opportunity to maximize automation tools.
- ✓ **Build -** In the build stage, take the provided code and build it for testing purposes. The code is built in a development environment to allow testing and bug fixes.
- ✓ **Test - Automated testing** ensures the project is functioning as expected and finds any bugs or issues in behavior. Depending on the team workflow, UI/UX or performance testing will also happen in this stage.

Operation Stages

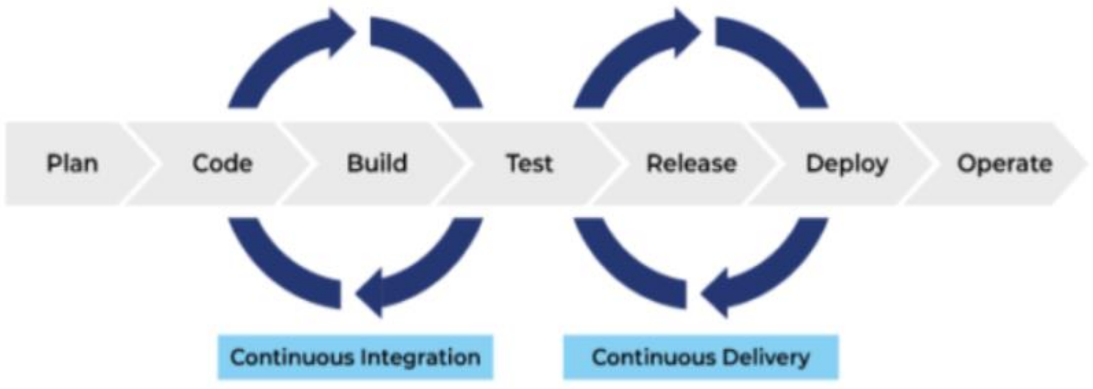
- ✓ **Release -** The release stage is when the Ops team confirms that the project is ready to be released and builds it into the production environment. This stage is critical as it is the last stop after multiple stages for checks — like vulnerabilities and bugs — just before deployment.
- ✓ **Deploy -** Deployment is the stage, move the project — in its current state — to the production environment for the end-users to access. This stage is where approved changes get deployed to the user.
- ✓ **Operate -** In the Operate stage, the operations team will configure and manage the project in the production environment. Typically, the team will rely on automation to help maintain the DevOps project in this stage.
- ✓ **Monitor -** In the Monitor stage, the project is being used. From the recorded results, teams will gain more insight into behavior, user response, and the general success of the product.

Components or Phases of a DevOps pipeline



Continuous Development:
Jira, Confluence, Bitbucket, Git, Svn, VSTS, etc.
Continuous Deployment:
Puppet, Chef, Ansible, SaltStack, etc.
Code
Plan
Build
Test
DEV
Release
Deploy
Operate
Monitor
OPS
Continuous Testing:
Zephyr, TestComplete, SoapUI, Selenium TestNG, JUnit/NUnit
Continuous Integration:
Jenkins, Bamboo, Hudson, etc
Continuous Monitoring:
Splunk, ELK Stack, Nagios, Sensu, NewRelic

1. Continuous integration/continuous delivery/deployment (CI/CD)



Continuous integration is the practice of making frequent commits to a common source code repository.

- ✓ It’s continuously integrating code changes into existing code base so that any conflicts between different developer’s code changes are quickly identified and relatively easy to remediate. This practice is critically important to increasing deployment efficiency.

Continuous delivery ensures that the “main” or “trunk” branch of an application's source code is always in a releasable state.

- ✓ This means having a pre-production environment that is as close to identical to the production environment as possible and ensuring that automated tests are executed, so that every variable that might cause a failure is identified before code is merged into the main or trunk branch.

Continuous deployment entails having a level of continuous testing and operations that is so robust, new versions of software are validated and deployed into a production environment without requiring any human intervention.

- ✓ This is rare and, in most cases, unnecessary. It is typically only the unicorn businesses who have hundreds or thousands of developers and have many releases each day that require, or even want to have, this level of automation.

2. Continuous feedback

Continuous testing is a critical component of every DevOps pipeline and one of the primary enablers of continuous feedback. In a DevOps process, changes move continuously from development to testing to deployment, which leads not only to faster releases, but a higher quality product.

- ✓ This means having automated tests throughout the pipeline, including unit tests that run on every build change, smoke tests, functional tests, and end-to-end tests.

Continuous monitoring is another important component of continuous feedback. A DevOps approach entails using continuous monitoring in the staging, testing, and even development environments.

- ✓ It is sometimes useful to monitor pre-production environments for anomalous behavior, but in general this is an approach used to continuously assess the health and performance of applications in production

3.**Continuous Operations**

This stage is for limiting planned downtime and preventing unplanned downtime. Operations will take steps to avoid issues and set up a plan of attack that leaves the project running, which helps maintain the pipeline flow — a very important step.

To Build a DevOps pipeline

Step 1: Choose CI/CD Tool

- ✓ Although DevOps methodologies might differ from company to company, each company must pick a CI/CD tool that suits its specific needs.
- ✓ The most common examples of these CI/CD tools are Jenkins, GitLab, Bamboo, TeamCity, and CirlceCi. Each tool has specific features and properties to help teams choose the best fit.

Step 2: Establish a Control Environment

- ✓ Whether the development team is small or not, it need a central location or place - known as a control environment — to store the source code.

- ✓ This control environment allows multiple developers to work together on the same codebase, avoid merge conflicts, and be on the same page regarding any changes made to the code.
- ✓ Examples of source control management tools include Git, GitLab, and BitBucket.

Step 3: Set up a build server

- ✓ A build server or continuous integration (CI) server offers developers a centralized and reliable environment to build distributed development projects.
- ✓ Without a build server, it would be difficult to determine if the code works and is ready for production.

Step 4. Initiate Automated Tests

- ✓ After setting up the code on the build server, the next step is to test it.
- ✓ It can run automated unit, regression, functional, and integration tests, ensuring code is error-free as move on to the deployment stage.

Step 5: Deploy to Production

- ✓ At the final stage, choose to deploy the code either manually or automatically.
- ✓ Deploying manually first is often recommended because able to monitor and quickly flag any issues that might come up. If no issues rear its head, then automatically deploy the code to save time and effort.
- ✓ To need a server infrastructure, deploy the code manually or automatically. The server infrastructure usually depends on the function want it to perform and the type of software want to deploy.

Deployment Pipeline

- ✓ In software development, a **deployment pipeline** is a system of automated processes designed to quickly and accurately move new code additions and updates from version control to production. In past development environments, manual steps were necessary when writing, building, and deploying code.
- ✓ This was extremely time consuming for both developers and operations teams, as they were responsible for performing tedious manual tasks such as code testing and code releases.

Components of a Deployment Pipeline

- ✓ **Version Control System (VCS) -** It is a central repository that tracks and manages code changes made by developers. It allows multiple developers to collaborate on the same codebase, facilitates versioning, and provides a historical record of changes. Popular VCS options include Git, SVN, and Mercurial.

- ✓ **Build Server -** It is responsible for automatically compiling and packaging the code whenever changes are pushed to the VCS. It takes the code from the repository and transforms it into executable artifacts that are ready for testing and deployment.
- ✓ **Automated Testing -** It includes various types of tests, such as unit tests, integration tests, functional tests, and performance tests. Automated testing ensures that any defects or issues are caught early in the development process, reducing the risk of introducing bugs into production.
- ✓ **Artifact Repository -** The artifact repository stores the build artifacts generated by the build server. These artifacts are the build process output and represent the compiled and packaged code ready for deployment. A centralized artifact repository ensures consistent and reliable deployments across different environments, such as staging and production.
- ✓ **Deployment Automation -** Deployment automation streamlines the application deployment process to various environments. It involves setting up automated deployment scripts and configuration management tools to ensure a consistent deployment process. By automating the deployment process, organizations can reduce manual errors and maintain high consistency in their deployments.

Main Stages of a Deployment Pipeline

There are four main stages of a deployment pipeline:

- ➢ Version Control
- ➢ Acceptance Tests
- ➢ Independent Deployment
- ➢ Production Deployment

Version Control is the first stage of the pipeline. This occurs after a developer has completed writing a new code addition and committed it to a source control repository such as GitHub.

- ➢ Once the commit has been made, the deployment pipeline is triggered and the code is automatically compiled, unit tested, analyzed, and run through installer creation.
- ➢ If and when the new code passes this version control stage, binaries are created and stored in an artifact repository. The validated code then is ready for the next stage in the deployment pipeline.

In the **Acceptance Tests** stage of the deployment pipeline, the newly compiled code is put through a series of tests designed to verify the code against the team's predefined acceptance criteria.

- ➢ These tests will need to be custom-written based on company goals and user expectations for the product.

- While these tests run automatically once integrated within the deployment pipeline, it’s important to be sure to update and modify the tests as needed to consistently meet rising user and company expectations.

Once code is verified after acceptance testing, it reaches the **Independent Deployment** stage where it is automatically deployed to a development environment.

- The development environment should be identical (or as close as possible) to the production environment in order to ensure an accurate representation for functionality tests.
- Testing in a development environment allows teams to squash any remaining bugs without affecting the live experience for the user.

The final stage of the deployment pipeline is **Production Deployment**. This stage is similar to what occurs in Independent Deployment, however, this is where code is made live for the user rather than a separate development environment.

- Any bugs or issues should have been resolved at this point to avoid any negative impact on user experience.
- DevOps or operations typically handle this stage of the pipeline, with an ultimate goal of zero downtime.

To Build a Deployment Pipeline

The three essential components to include

- ✓ **Build Automation (Continuous Integration) -** Build automation, also referred to as Continuous Integration or CI for short, are automated steps within development designed for continuous integration – the compilation, building, and merging of new code.
- ✓ **Test Automation -** Test automation relies on the creation of custom-written tests that are automatically triggered throughout a deployment pipeline and work to verify new compiled code against the organization’s predetermined acceptance criteria.
- ✓ **Deploy Automation (Continuous Deployment/Delivery) -** Like continuous integration, deploy automation with Continuous Deployment/Delivery (CD for short) helps expedite code delivery by automating the process of releasing code to a shared repository, and then automatically deploying the updates to a development or production environment.

Benefits of a Deployment Pipeline

- ✓ Teams are able to release new product updates and features much faster.
- ✓ There is less chance of human error by eliminating manual steps.

- ✓ Automating the compilation, testing, and deployment of code allows developers and other DevOps team members to focus more on continuously improving and innovating a product.
- ✓ Troubleshooting is much faster, and updates can be easily rolled back to a previous working version.
- ✓ Production teams can better respond to user wants and needs with faster, more frequent updates by focusing on smaller releases as opposed to large, waterfall updates of past production systems.