UNIT I – INTRODUCTION TO SOFTWARE ENGINEERING [9 hours]

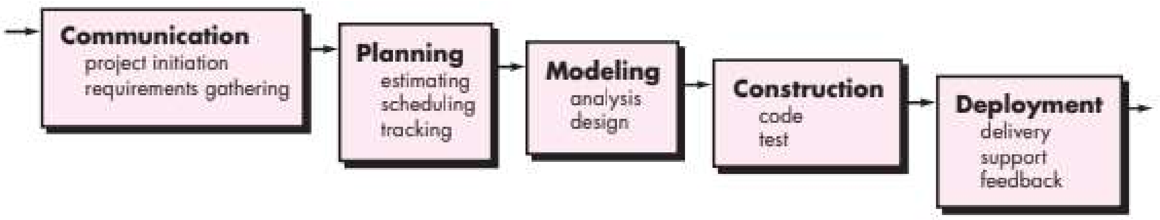
Definition of Software Engineering, Software Development Life Cycle (SDLC) – Phases,Traditional vs Agile Models (Waterfall, Agile, DevOps),Scrum Basics – Roles, Sprint, Backlog,Version Control using Git and GitHub,Introduction to Project Tools (GitHub Projects, Jira, Trello)

PRESCRIPTIVE PROCESS MODELS (OR) LIFE CYCLE MODELS

The process model can be defined as the abstract representation of a process. The appropriate process model can be chosen based on abstract representation of the process. These process models will follow some rules for correct usage. It is called "prescriptive" model because they prescribe a set of process elements—framework activities, software engineering actions, tasks, work products, quality assurance, and change control mechanisms for each project. Each process model also prescribes a process flow (also called a work flow)—that is, the manner in which the process elements are interrelated to one another.

The Waterfall Model (or)classic life cycle (or) sequential life cycle model (or) Software Development Life Cycle (SDLC) (or) Systems development life cycle (SDLC)

The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software. The waterfall model is the oldest paradigm for software engineering.



- In the **requirement gathering and analysis** phase the basic requirements of the system must be understood by a software engineer, who is called an analyst. The design is an intermediate step between requirements analysis and coding.
- **Design** focuses on:
  a. Data Structure
  b. Software architecture
  c. Interface representation
  d. Algorithm details
- **Coding** is a step in which design is translated into machine readable form. Testing begins when coding is done. The purpose of testing is to uncover errors, fix the bugs and meet the customer requirements.
- **Maintenance** is the longest life cycle phase. The purpose of maintenance is when the system is installed and put in practical use then error may get introduced, correcting such errors and putting it in use.

Advantages:

1. The waterfall model is simple to implement
2. For implementation of small systems it is useful.

Problems in waterfall model:

1. Real projects rarely follow the sequential flow that the model proposes. Changes can cause confusion as the project team proceeds.

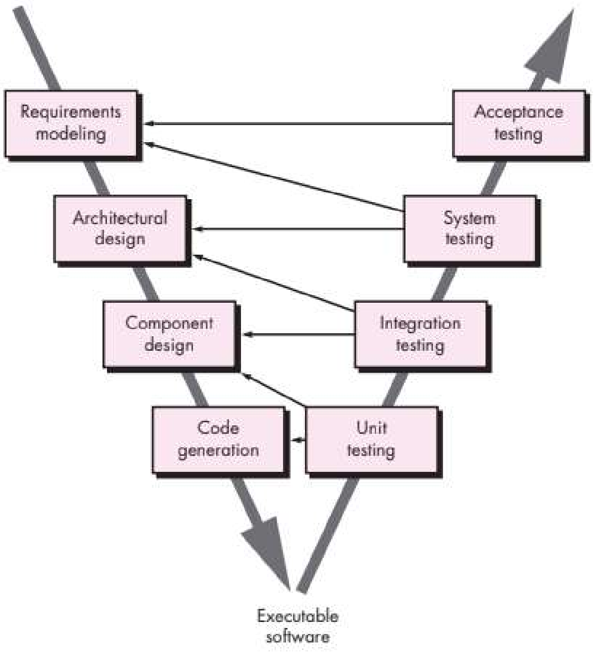
2. It is difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.

3. The customer must have patience. A working version of the program(s) will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous. V-Model: In each phase, testing will be done.

V-Model:

A variation in the representation of the waterfall model is called the V-model. The V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities. As a software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution. Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side.

In reality, there is no fundamental difference between the classic life cycle and the V-model. The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work.



Incremental Process Models

The incremental model combines elements of linear and parallel process flows. The incremental model delivers a series of releases to the customer. These releases are called increments. More and more functionality is associated with each increment.

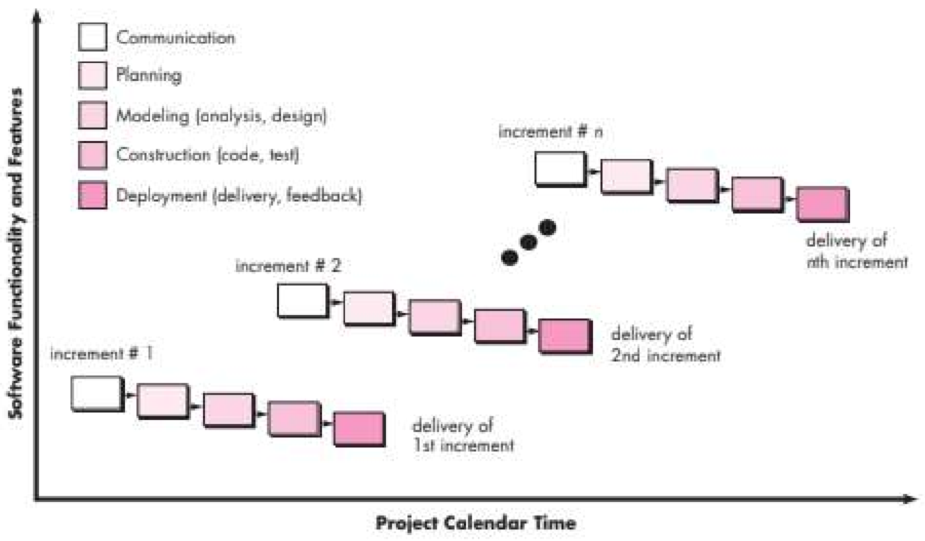
When we can choose incremental:

1) When initial software requirements are reasonably well defined

2) When the overall scope of the development effort precludes a purely linear process.

3) When limited set of software functionality needed quickly

The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable "increments" of the software in a manner that is similar to the increments produced by an evolutionary process flow.


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
Software Functionality and Features
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
Project Calendar Time

The incremental model applies linear sequences in a staggered fashion as calendar time progresses. For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; Spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment. It should be noted that the process flow for any increment can incorporate the prototyping paradigm.

The first increment is often a core product. That is, basic requirements are addressed but many supplementary features remain undelivered. The core product is used by the customer. As a result of use, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project. Early increments can be implemented with fewer people.

If the core product is well received, then additional staff (if required) can be added to implement the next increment. In addition, increments can be planned to manage technical risks.

Advantages:

1) Generates working software quickly and early during the software life cycle.
2) This model is more flexible – less costly to change scope and requirements.
3) It is easier to test and debug during a smaller iteration.
4) In this model customers can respond to each build.
5) Lowers initial delivery cost.
6) Easier to manage risk because risky pieces are identified and handled during its iteration.

Disadvantages:

1) Needs good planning and design.
2) Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
3) The total cost is higher than waterfall.

Evolutionary Process Models

Business and product requirements often change as development proceeds, making a straight line path to an end product unrealistic; In such cases, the iterative approach needs to be adopted. Evolutionary process model is also called as iterative process model

Evolutionary models are iterative. They are characterized in a manner that enables you to develop increasingly more complete versions of the software.

Prototyping: Software prototyping, refers to the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed. It is an activity that can occur in software development and is comparable to prototyping as known from other fields, such as mechanical engineering or manufacturing.

When we can choose Prototype:

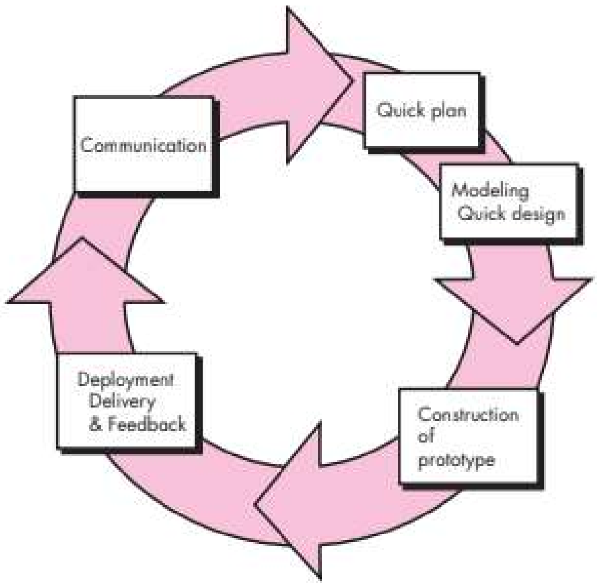
- A customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features.
- The developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system
- When requirements are fuzzy

Although prototyping can be used as a stand-alone process model, it is more commonly used as a technique that can be implemented within the context of any one of the process models.

Although prototyping can be used as a stand-alone process model, it is more commonly used as a technique that can be implemented within the context of any one of the process models.

The prototyping assists you and other stakeholders to better understand what is to be built when requirements are fuzzy.

The prototyping paradigm begins with communication. You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.



A prototyping iteration is planned quickly, and modeling (in the form of a "quick design") occurs. A quick design focuses on a representation of those aspects of the software that will be visible to end users (e.g., human interface layout or output display formats).

The quick design leads to the construction of a prototype. The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.

Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better understand what needs to be done

Ideally, the prototype serves as a mechanism for identifying software requirements. If a working prototype is to be built, you can make use of existing program fragments or apply tools (e.g., report generators and window managers) that enable working programs to be generated quickly.

In most projects, the first system built is barely usable. It may be too slow, too big, awkward in use or all three. There is no alternative but to start again, smarting but smarter, and build a redesigned version in which these problems are solved.

The prototype can serve as "the first system." The one that Brooks recommends you throw away. But this may be an idealized view. Although some prototypes are built as "throwaways," others are evolutionary in the sense that the prototype slowly evolves into the actual system.

Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers get to build something immediately.

Advantages:

1) Users are actively involved in the development

2) Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.

3) Errors can be detected much earlier.

4) Quicker user feedback is available leading to better solutions.

5) Missing functionality can be identified easily

6) Confusing or difficult functions can be identified Requirements validation, Quick implementation of, incomplete, but functional, application.

Disadvantages:

1) Stakeholders see what appears to be a working version of the software, unaware that the prototype is held together haphazardly, unaware that in the rush to get it working you haven't considered overall software quality or long-term maintainability.

2) Software engineer make implementation compromises in order to get a prototype working quickly.

3) An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability.

Usage of prototyping:

Although problems can occur, prototyping can be an effective paradigm for software Engineering. The key is to define the rules of the game at the beginning; that is, all stakeholders should agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded (at least in part), and the actual software is engineered with an eye toward quality.

The Spiral Model.

The spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software. The spiral development model is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.

It has two main distinguishing features.

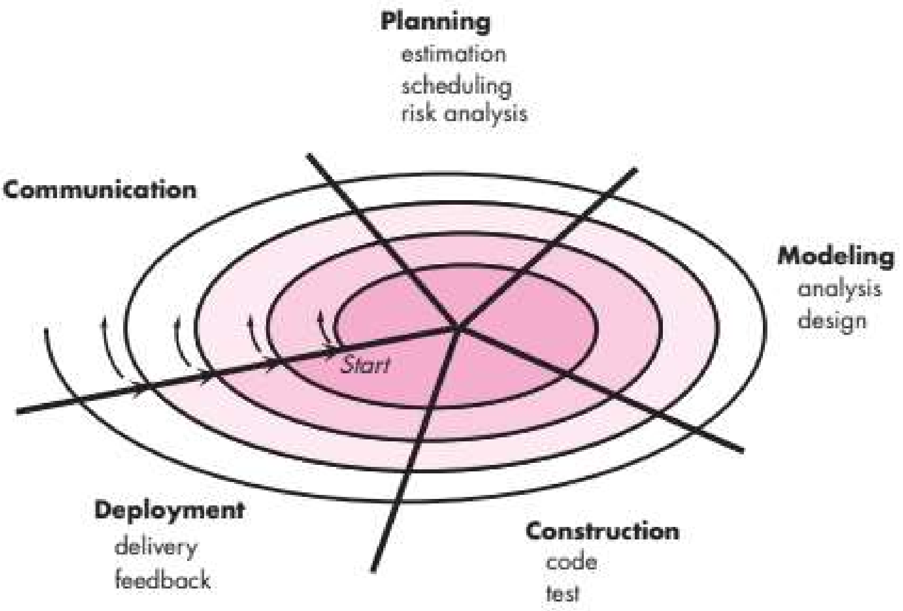
(1) One is a cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.

(2) The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

A spiral model is divided into a set of framework activities defined by the software engineering team. Each of the framework activities represent one segment of the spiral path. The spiral model is a realistic approach to the development of large-scale systems and software. Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level.

The spiral model uses prototyping as a risk reduction mechanism but enables you to apply the prototyping approach at any stage in the evolution of the product. It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world.

The spiral model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic.



The functions of these four quadrants are discussed below-

- **Objective determination and identify alternative solutions (Concept development projects):** Requirements are gathered from the customers and the objectives are identified, elaborated and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
- **Identify and resolve Risks (New product development projects):** During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, Prototype is built for the best possible solution.
- **Develop next version of the Product (Product Enhancement projects):** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
- **Review and plan for the next Phase (product Maintenance projects):** In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

Advantages:

1. High amount of risk analysis hence, avoidance of Risk is enhanced.
2. Good for large and mission-critical projects.
3. Strong approval and documentation control.
4. Additional Functionality can be added at a later date.
5. Software is produced early in the software life cycle.

Disadvantages:

1. Can be a costly model to use.
2. Risk analysis requires highly specific expertise.
3. The project's success is highly dependent on the risk analysis phase.
4. Doesn’t work well for smaller projects.

Concurrent development Models

The concurrent development model, sometimes called concurrent engineering, allows a software team to represent iterative and concurrent elements of any of the process models. For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following software engineering actions: prototyping, analysis, and design. Figure provides a schematic representation of one software engineering activity within the modeling activity using a concurrent modeling approach. The activity—modeling—may be in any one of the states noted at any given time.

Similarly, other activities, actions, or tasks (e.g., communication or construction) can be represented in an analogous manner. All software engineering activities exist concurrently but reside in different states. For example, early in a project the communication activity (not shown in the figure) has completed its first iteration and exists in the awaiting changes state. The modeling activity (which existed in the inactive state while initial communication was completed, now makes a transition into the under development state.

If the customer indicates that changes in requirements must be made, the modeling activity moves from the under development state into the awaiting changes state.

Concurrent modeling defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks. For example, during early stages of design (a major software engineering action that occurs during the modeling activity), an inconsistency in the requirements model is uncovered. This generates the event analysis model correction, which will trigger the requirements analysis action from the done state into the awaiting changes state.

Concurrent modeling is applicable to all types of software development and provides an accurate picture of the current state of a project. Rather than confining software engineering activities, actions, and tasks to a sequence of events, it defines a process network. Each activity, action, or task on the network exists simultaneously with other activities, actions, or tasks. Events generated at one point in the process network trigger transitions among the states.

Advantages:

1) The concurrent development model, sometimes called concurrent engineering. It can be represented schematically as a series of framework activities, software engineering actions, software engineering tasks and their associated states.

2) The concurrent process model defines a series of events that will trigger transition from state to state for each of the software engineering activities and action or task.

3) The concurrent process model is applicable to all types of software development and provides an accurate picture of the current state of a project.

Disadvantages:

1) The SRS must be continually updated to reflect changes.

2) It requires discipline to avoid adding too many new features too late in the project.

Comparison of different SDLC models:

Waterfall Model	**Spiral Model**	**Prototyping Model**	**Incremental Model**
Requirements must be clearly understood and defined at the beginning only.	The requirements analysis and gathering can be done in iteration because requirements get changed quite often.	Requirements analysis can be made in the later stages of the development cycle. Because requirements get changed quite often.	Requirements analysis can be made in the later stages of the development cycle.
The development team having the adequate experience of working on the similar project is chose to work on this type of process model	The development team having the adequate experience of working on the similar project is allowed in this process model.	The development team having the adequate experience of working on the similar project is allowed in this process model.	The development team having the adequate experience of working on the similar project is chosen to work on this type of process model
There is no user involvement in all the phases of the development process.	There is no user involvement in all the phases of the development process.	There is user involvement in all the phases of the development process.	There is user involvement in all the phases of the development process.
When the requirements are reasonably well	Due to the iterative nature of this model, the risk	When a developer is unsure about the efficiency of an	When the requirements are reasonably well defined and the

defined and the development effort suggests a purely linear effort then the waterfall model is chosen.	identification and rectification is done before they get problematic. Hence for handling real time problems the spiral model is chosen.	algorithm or the adaptability of an operating system then the prototyping model is chosen.	development effort suggests a purely linear effort and when a limited set of software functionality is needed quickly then the incremental model is chosen.

AGILE PROCESS:

Agile is a time-bound, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver all at once. The agile manifesto for agile software development is a formal declaration of four values and principles to guide an iterative and people centric approach to software development.

Why Agile?

Technology in this current era is progressing faster than ever, enforcing the global software companies to work in a fast-paced changing environment. Because these businesses are operating in an ever-changing environment, it is impossible to gather a complete and exhaustive set of software requirements. Without these requirements, it becomes practically hard for any conventional software model to work.

Agile was specially designed to meets the needs of the rapidly changing environment by embracing the idea of incremental development and develop the actual final product.

Agile Process:

In 1980's the heavy weight, plan based software development approach was used to develop any software product. In this approach too many things are done which are not directly related to software products being produced. If requirements get changed, then rework was essential. Hence new methods were proposed in 1990's which are known as agile process. The agile process is light-weight methods which are people-based rather than plan-based methods.

The agile process forces the development team to focus on software itself rather than design and documentation. The agile process believes in iterative method. The aim of agile process is to deliver the working model of software quickly to the customer. **Conventional Software Development Methodology:**

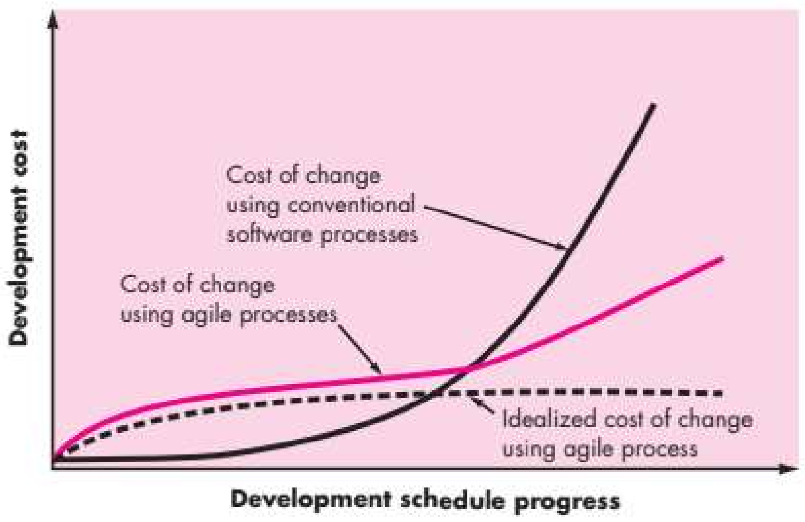
The conventional wisdom in software development is that the cost of change increases nonlinearly as a project progresses. It is relatively easy to accommodate a change when a software team is gathering requirements. A usage scenario might have to be modified, a list of functions may be extended, or a written specification can be edited.

As the process progresses and if the customer suggests the changes during the testing phase of the SDLC then to accommodate these changes the architectural design needs to be modified and ultimately these changes will affect other phases of SDLC. These changes are actually costly to execute.

Agile Methodology:

When incremental delivery is coupled with other agile practices such as continuous unit testing and pair programming then the cost of changes can be controlled.

The following graph represents how the software development approach has a strong influence on the development cost due to changes suggested.



Principles:

There are famous 12 principles used as agile principles:

1. Highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. It welcomes changing requirements, even late in development.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shortest timescale.
4. Business people and developers must work together throughout the project.
5. Build projects around motivated individuals. Give them the environment and the support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote constant development. The sponsors, developers, and users should be able to maintain a constant.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity the art of maximizing the amount of work not done is essential.
11. The team must be self– organizing teams for getting best architectures, requirements, and designs emerge from
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

---

DevOps

DevOps is a software development approach emphasizing collaboration, automation, and continuous delivery to provide high-quality products to customers quickly and efficiently. DevOps breaks down silos between development and operations teams to enable seamless communication, faster time-to-market, and improved customer satisfaction.

It allows a team to handle the complete application lifecycle, from development to testing, operations, and deployment. It shows cooperation between Development and

Operations groups to deploy code to production quickly in an automated and repeatable manner.

Every phase of the software development lifecycle, including planning, coding, testing, deployment, and monitoring, is heavily automated in DevOps. This improves productivity, ensures consistency, and lowers error rates in the development process.

A culture of continuous improvement is also promoted by DevOps, where feedback loops are incorporated into the procedure to facilitate quicker iteration and better decision-making. Organizations can increase their agility, lower costs, and speed up innovation by adopting DevOps.

Why is DevOps Needed?

- Before DevOps, the development and operation team worked in complete isolation.
- Testing and Deployment were isolated activities done after design-build. Hence they consumed more time than actual build cycles.
- Without using DevOps, team members spend a large amount of their time testing, deploying, and designing instead of building the project.
- Manual code deployment leads to human errors in production.
- Coding & operation teams have separate timelines and are not sync, causing further delays.

How is DevOps different from traditional IT

In this DevOps training, let's compare the traditional software waterfall model with DevOps to understand the changes DevOps brings. We assume the application is scheduled to go live in 2 weeks, and coding is 80% done. We assume the application is a fresh launch, and the process of buying servers to ship the code has just begun-

Sl.No	**Old Process**	**DevOps**
1.	After placing an order for new servers, the Development team works on testing. The Operations team works on extensive	After placing an order for new servers, the Development and Operations team work together on the paperwork to set up

	paperwork as required in enterprises to deploy the infrastructure.	the new servers. This results in better visibility of infrastructure requirements.
2.	Projections about failover, redundancy, data center locations, and storage requirements are skewed as no inputs are available from developers who have deep knowledge of the application.	Projections about failover, redundancy, disaster recovery, data center locations, and storage requirements are pretty accurate due to the inputs from the developers.
3.	The operations team has no clue about the progress of the Development team. The operations team develops a monitoring plan as per their understanding.	In DevOps, the Operations team is completely aware of the developers' progress. Operations teams interact with developers and jointly develop a monitoring plan that caters to IT and business needs. They also use advanced Application Performance Monitoring (APM) Tools.
4.	Before going go-live, the load testing crashes the application, and the release is delayed.	Before going go-live, the load testing makes the application a bit slow. The development team quickly fixes the bottlenecks, and the application is released on time.

Why is DevOps used?

DevOps allows Agile Development Teams to implement Continuous Integration and Continuous Delivery, which helps them launch products faster into the market.

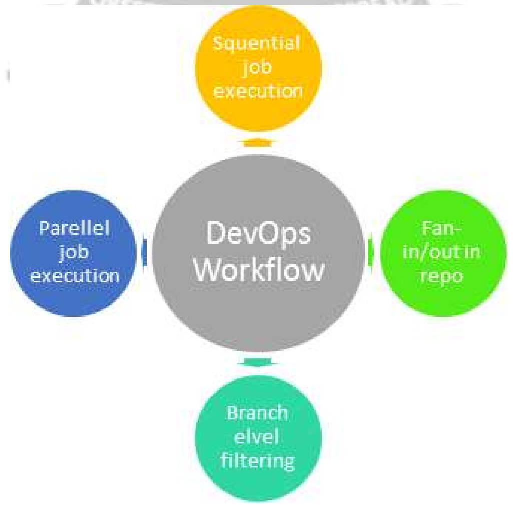
Other Important reasons are:

1. **Predictability**: DevOps offers a significantly lower failure rate of new releases.

2. **Reproducibility**: Version everything so that earlier versions can be restored anytime.
3. **Maintainability**: Effortless recovery process in the event of a new release crashing or disabling the current system.
4. **Time to market:** DevOps reduces the time to market up to 50% through streamlined software delivery. It is particularly the case for digital and mobile applications.
5. **Greater Quality:** DevOps helps the team improve application development quality by incorporating infrastructure issues.
6. **Reduced Risk:** DevOps incorporates security aspects in the software delivery lifecycle, and it helps reduce defects across the lifecycle.
7. **Resiliency:** The Operational state of the software system is more stable, secure, and changes are auditable.
8. **Cost Efficiency:** DevOps offers cost efficiency in the software development process, which is always an aspiration of IT management.
9. **Breaks larger code base into small pieces:** DevOps is based on the agile programming method. Therefore, it allows breaking larger codebases into smaller and manageable chunks.

DevOps Workflow

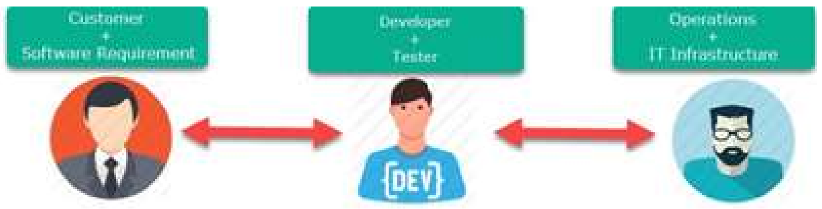
Workflows provide a visual overview of the sequence in which input is provided. It also tells about performed actions, and output is generated for an operations process.



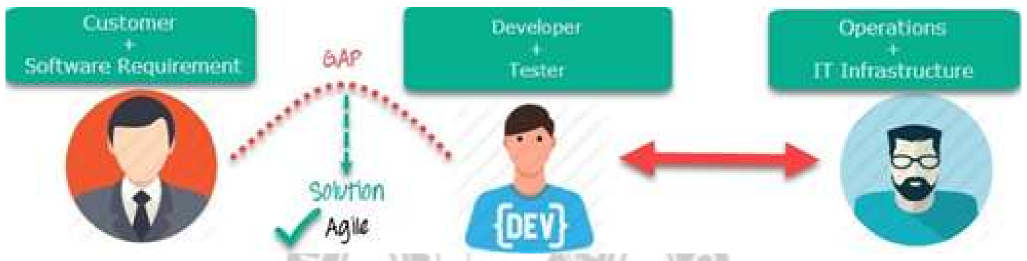
Workflow allows the ability to separate and arrange jobs that the users top request. It also can mirror their ideal process in the configuration jobs.

How is DevOps different from Agile? DevOps Vs Agile

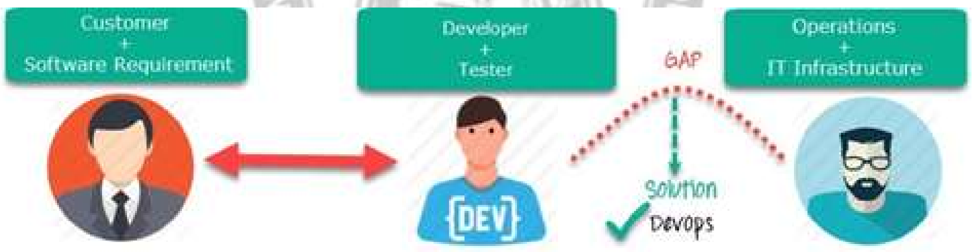
Stakeholders and communication chain a typical IT process.



Agile addresses gaps in Customer and Developer communications



DevOps addresses gaps in Developer and IT Operations communications



Difference between DevOps and Agile

Agile	Devops
Emphasize breaking down barriers between developers and management	DevOps is about software deployment and operation teams.
Addresses gaps between customer requirements and development teams.	Addresses the gap between the development and Operation team
Focuses more on functional and non-functional readiness	It focuses on operational and business readiness.

Agile development pertains mainly to the company's way development is thought out.	DevOps emphasises deploying software in the most reliable and safest ways that aren't always the fastest.
Agile development emphasises training all team members to have varieties of similar and equal skills. So that, when something goes wrong, any team member can get assistance from any member in the absence of the team leader	DevOps likes to divide and conquer, spreading the skill set between the development and operation teams. It also maintains consistent communication
Agile development manages "sprints". It means that the timetable is much shorter (less than a month), and several features are to be produced and released in that period.	DevOps strives for consolidated deadlines and benchmarks with significant releases rather than smaller and more frequent ones.

DevOps Automation Tools

Automating all the testing processes and configuring them to achieve speed and agility is vital. This process is known as DevOps automation.

Classified briefly into six different categories.

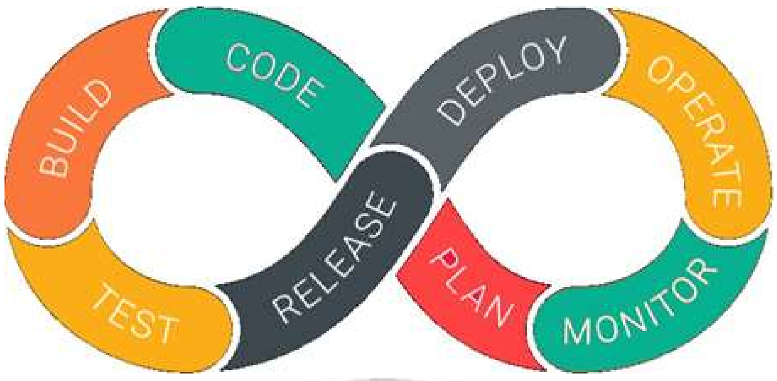
1. Infrastructure Automation- ex: AWS
2. Configuration Management- ex: Chef
3. Deployment Automation- ex: Jenkins
4. Performance Management- ex: App dynamic
5. Log Management- ex: Splunk
6. Monitoring- ex: Nagios

What is DevOps Lifecycle?

The DevOps Lifecycle is a series of development stages that guide everyone as efficiently as possible through the end-to-end process of product development. All of these components of the DevOps lifecycle is necessary to take the maximum leverage of the DevOps methodology.

DevOps Lifecycle: Key Components

Here are some important DevOps Lifecycle phases / Key components of DevOps:



Stage 1) Continuous Development:

This practice spans the planning and coding phases of the DevOps lifecycle. Version-control mechanisms might be involved.

Stage 2) Continuous Integration:

This software engineering practice develops software by frequently integrating its components. It helps to ensure that changes in the source code do not break the build or cause other problems.

Stage 3) Continuous Testing:

This DevOps lifecycle stage incorporates automated, prescheduled, continued code tests as application code is written or updated. Such tests can be written manually or in conjunction with continuous integration tools.

Stage 4) Continuous Deployment:

The deployment process takes place continuously in this DevOps lifecycle phase. It is performed so that any changes made in the code should not affect the functioning of a high traffic website.

Stage 5) Continuous Monitoring:

During this phase, developers collect data, monitor each function, and spot errors like low memory or server connection are broken. For example, when users log in, they should access their account, and a failure to do so means there's a problem with your application.

Stage 6) Continuous Feedback:

Continuous feedback is like a progress report. In this DevOps stage, the software automatically sends out information about performance and issues experienced by the end-user. It's also an opportunity for customers to share their experiences and provide feedback.

Stage 7) Continuous Operations:

It is the last, shortest, and most straightforward phase of DevOps. It also involves automating the application's release and all these updates that help you keep cycles short and give developers more time to focus on developing.