UNIT-I
INTRODUCTION TO DEVOPS

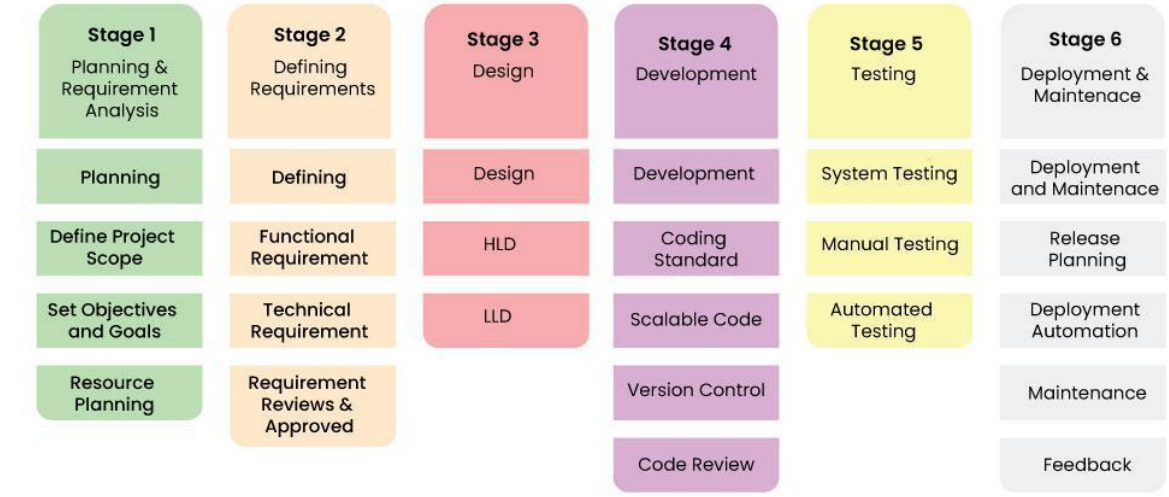
Introduction, Agile development model, DevOps and ITIL. DevOps process and Continuous Delivery, Release management, Scrum, Kanban, delivery pipeline, identifying bottlenecks.

INTRODUCTION

Software Development Life Cycle (SDLC):

- Software Development Life Cycle (SDLC) is a structured process that is used to **design, develop, and test high-quality software**, and diagrammatic representation of the software life cycle.
- A life cycle model **represents all the meth**ods required to make a software product transit through its life cycle stages.
- It also **captures the structure** in which these methods are to be undertaken.
- SDLC consists of a precise plan that **describes how to develop, maintain, replace, and enhance specific software**.

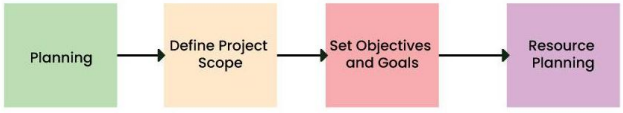
Stages of Software Development Life Cycle:



Stages of Software Development Life Cycle

Stage 1: Planning and requirement analysis

- Requirement Analysis is the most important and necessary stage in SDLC.
- The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry.
- Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.



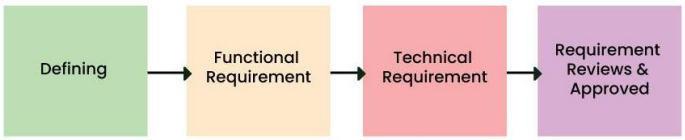
Planning and Requirement Analysis

- Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product.
- Before creating a product, a core understanding or knowledge of the product is very necessary.

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.
Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

Stage 2: Defining Requirements

- Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.
- This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.



Defining Requirements

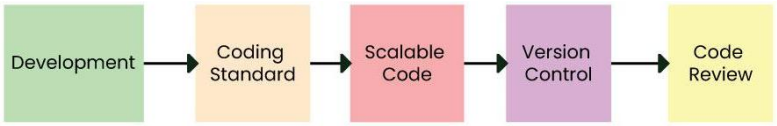
Stage 3: Design

- The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project.
- This phase is the product of the last two, like inputs from the customer and requirement gathering.

Designing Architecture

Stage 4: Developing the project

- In this phase of SDLC, the actual development begins, and the programming is built.
- The implementation of design begins concerning writing code.
- Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.



Developing the product

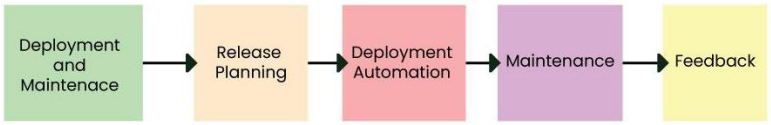
Stage 5: Testing

- After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.
- During this stage, unit testing, integration testing, system testing, acceptance testing are done.

Product Testing and Integration

Stage 6: Deployment

- Once the software is certified, and no bugs or errors are stated, then it is deployed.
- Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.
- After the software is deployed, then its maintenance begins.



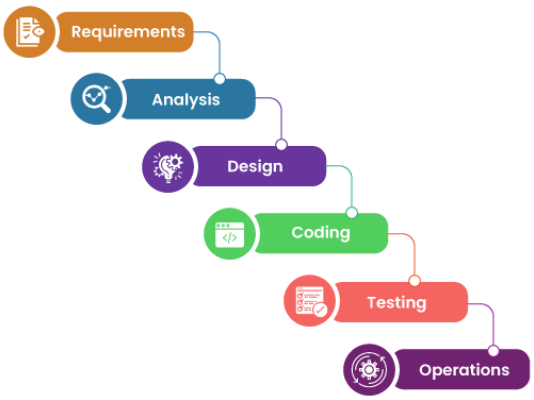
Deployment and Maintenance of Products

Stage 7: Maintenance

- Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.
- This procedure where the care is taken for the developed product is known as maintenance.

Waterfall Model:

- **Winston Royce** introduced the Waterfall Model in **1970**.
- The waterfall model in software development is **a linear, sequential approach** where each phase is completed before the next one begins.
- The phases are
    - Requirements
    - Analysis
    - Design, coding/implementation
    - Testing
    - Operations (deployment, maintenance)



Waterfall Model

- The waterfall model involves **breaking your project into several phases** that must be **completed sequentially**.
- Imagine flowing water in a waterfall. Your project will begin at the top and descend the waterfall into the stream at the bottom. This happens in one, smooth flowing pattern, rather than a scattered, horizontal stream. Each of your waterfall model phases will precede the next until you approach your final project deadline.

Requirements:

- All customer **requirements are collected and approved** before the beginning of the project, meaning **no mid-project changes are allowed**.
- The outcome of this phase is a project requirements document, which means that all necessary data is collected and no further customer interference is required

Analysis:

Project specifications are **reviewed from business perspectives**; **technical and financial resources** are audited for feasibility.

Design:

Project manager relies on project requirements to **develop project specifications**, including **project plan**.

Coding/Implementation:

At this stage, all the previous planning is put into action.

Testing:

After the development is completed, testing is performed to **identify flaws and errors, fix** and **refinement the end-product**. Customer involvement is possible.

Deployment:

The fully functional **product is released or handed over to the customer**. In non-software projects, delivery is most often the final phase.

Maintenance:

Software projects often require maintenance, such as **product improvements and updates**.

All these stages flow downwards one-by-one like a waterfall. The next stage is started only after the previous one is over. Following the process cycle suggested in the original waterfall model and completing each phase step by step, you're bound to produce a properly functioning, high-quality piece of software.

Advantages of the Waterfall Model

- **Easy and straightforward.** Each Waterfall phase has its own review process and deliverables.
- **Predictable**. All the requirements, processes, timelines, deadlines and end-product are fixed and well-documented.
- **Comprehensive documentation.** Waterfall requires many project management deliverables, which ensure a greater understanding of the tasks and the end product.
- **The fastest project delivery** in small and simple projects.

Disadvantages of the Waterfall Model

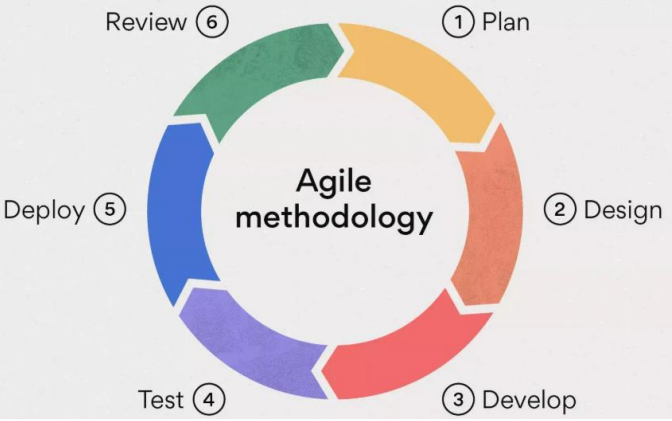
- **No room for changes**. Waterfall is not compatible with changes in client requirements
- **Never backward**. Waterfall is a traditional project management methodology that doesn't allow teams to get back to the previous project development phases
- **Delayed testing**. In the Waterfall model, testing is delayed until the end of the project development, meaning you discover mistakes and flaws too late and have to invest a lot of time in fixing them instead of managing them early on.
- **Reduced efficiency** because project phases don't overlap.
- **Pressure and stress** caused by inflexible deadlines.
- **Lower customer satisfaction** due to the lack of customer feedback during the implementation phase.
- **High risks**. If any unexpected obstacles happen, there's a high chance that they will negatively impact your project's timeline.
- **Not suitable for complex projects** with high risks.

AGILE DEVELOPMENT MODEL

- Agile is a term that describes approaches to software development that emphasize **incremental delivery, team collaboration, continual planning, and continual learning**.
- The term Agile was coined in **2001 in the Agile Manifesto**.
- Agile means **learning from experience and continually improving**.
- Agile development provides more learning cycles than traditional project planning due to the tighter process loops.
- Agile methodology is a project management framework that **breaks projects down into several dynamic phases**, commonly known as **sprints to drive continuous improvement**.
- Each sprint provides something new for the team to learn.
- The Agile framework is an **iterative methodology**.
- After every sprint, teams reflect and look back to see **if there was anything that could be improved** so they can adjust their strategy for the next sprint.

For example:

A team delivers value to the customer, gets feedback, and then modifies their backlog based on that feedback. They learn that their automated builds are missing key tests. They include work in their next sprint to address this issue. They find that certain features perform poorly in production, so they make plans to improve performance. Someone on the team hears of a new practice. The team decides to try it out for a few sprints.



12 Agile principles

1. **Satisfy customers through early, continuous improvement and delivery:** When customers receive new updates regularly, they're more likely to see the changes they want within the product. This leads to happier, more satisfied customers—and more recurring revenue.
2. **Welcome changing requirements, even late in the project:** The Agile framework is all about adaptability. In iterative approaches like Agile, being inflexible causes more harm than good.
3. **Deliver value frequently:** Similar to principle 1, continuous delivery of value to your customers or stakeholders frequently makes it less likely for them to churn.
4. **Break the silos of your projects:** Cross-functional teams and collaboration is a key Agile value. The goal is for people to break out of their individual projects and collaborate more frequently.
5. **Build projects around motivated individuals:** Agile management works best when teams are committed and actively working to achieve a goal.
6. **The most effective way to communicate is face-to-face:** If you're working on a distributed team, spend time communicating in ways that involve face-to-face communication like Zoom calls or daily stand-up meetings.
7. **Working software is the primary measure of progress:** The ultimate goal of software development projects is a working product, and the Agile framework supports this by prioritizing functional software above all.
8. **Maintain a sustainable working pace:** Some aspects of Agile project management can be fast-paced, but it shouldn't be so fast that team members burn out. The goal is to maintain sustainability throughout the development process.
9. **Continuous excellence enhances agility:** If the team develops excellent code in one sprint, they can continue to build off of it the next. Continually creating great work allows teams to move faster in the future.
10. **Simplicity is essential:** Sometimes the simplest solution is the best solution. Agile development aims to not overcomplicate things and find simple answers to complex problems.
11. **Self-organizing teams generate the most value:** Similar to principle 5, proactive teams become valuable assets to the company as they strive to deliver continuous improvement.
12. **Regularly reflect and adjust your way of work to improve effectiveness**: Reviewing meetings are a common Agile practice. It's a dedicated time for Agile teams to look back and reflect on their performance and adapt their behaviors for the future.

Agile software development cycle

The Agile software development cycle can be broken down into the following six steps:

1. Concept
2. Inception
3. Iteration and construction

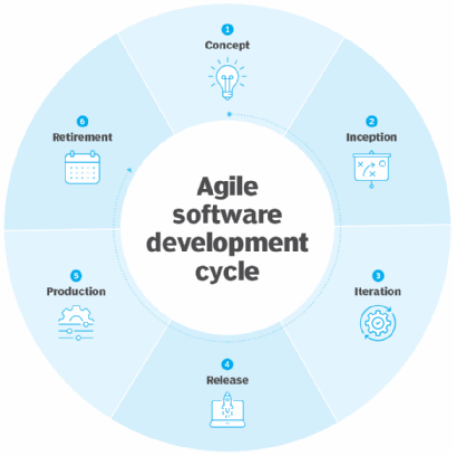
4. Release
5. Production
6. Retirement

Concept

The first step, *concept*, involves identifying business opportunities in each potential project as well as estimating the time and work that will be required to complete the project. This information can then be used to prioritize projects and discern which ones are worth pursuing based on technical and economic feasibility.

Inception

During the second step, *inception*, team members are identified, funding is established, and the initial requirements are discussed with the customer. A timeline should also be created that outlines the various responsibilities of teams and clearly defines when work is expected to be completed for each sprint. A sprint is a set period during which specific work must be completed and made ready for review.



The Agile development cycle is more of a loop than a linear cycle, as the steps are repeated until each item is satisfied.

Iteration and construction

The third step, *iteration and construction*, is when teams start creating working **software based on requirements and continuous feedback**. The Agile software development cycle relies on iterations - or single development cycles - that build upon each other and lead into the next step of the overall development process until the project is completed. Each iteration typically lasts between two to four weeks, with a set completion date. The goal is to have a working product to launch at the end of each iteration.

Multiple iterations occur throughout the development cycle and they each possess their own workflow. A typical iteration flow consists of the following:

- Defining requirements based on the product backlog, sprint backlog and customer and stakeholder feedback.
- Developing software based on the set requirements.
- Conducting quality assurance testing, internal and external training and documentation.
- Delivering and integrating the working product into production.
- Gathering customer and stakeholder feedback on the iteration to define new requirements for the next sprint.

Release

The fourth step, *release*, involves final QA testing, resolution of any remaining defects, finalization of the system and user documentation and, at the end, release of the final iteration into production.

Production

After the release, the fifth step, *production*, focuses on the ongoing support necessary to maintain the software. The development teams must keep the software running smoothly while also teaching users exactly how to use it. The production phase continues until the support has ended or the product is planned for retirement.

Retirement

The final step, *retirement*, incorporates all end-of-life activities, such as notifying customers and final migration. The system release must be removed from production. This is usually done when a system needs to be replaced by a new release or if the system becomes outdated, unnecessary or starts to go against the business model.

Throughout the Agile cycle, different features can be added to the product backlog, but the entire process should consist of repeating each step over and over until every item in the backlog has been satisfied. This makes the Agile cycle more of a loop than a linear process. At any time, an enterprise can have multiple projects occurring simultaneously with iterations that are logged on different product lines and a variety of internal and external customers providing different business needs.

Advantages of Agile

- Superior quality product and higher customer satisfaction due to client involvement in the development process.
- Increased flexibility. Agile divides projects into small sprints that allow teams to implement changes on short notice.
- Reduced risks. Agile works in small sprints that focus on continuous delivery and eliminate chances of project failure.
- Continuous improvement. The cyclic structure and customer feedback allow project teams to improve the product throughout its development process.
- Improved team morale. Another distinct feature of Agile teams is self-organization, meaning more autonomy, less micromanagement and stress.
- Improved performance visibility with daily meetings and project progress charts.

Disadvantages of Agile

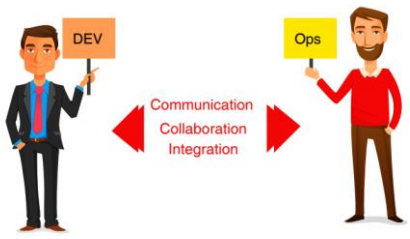
- Poor resource planning. Agile projects are not limited by the end result - they are continuously improved, meaning low control over project resources.
- Difficult to evaluate project time and costs. Agile projects welcome customer feedback and changes to the project requirements that usually bloat project timelines and costs.
- High possibility of scope creep. Projects easily get sidetracked due to high customer involvement and their changes to the project requirements.
- Limited documentation. In Agile projects, documentation is built incrementally, meaning it is not detailed enough to be reliable.

Differences Between Waterfall and Agile Methodologies

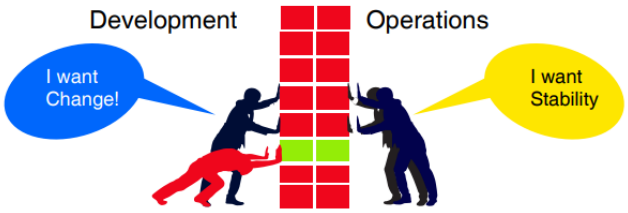
Waterfall	Agile
Sequential development process in pre-defined phases	Iterative development in short sprints
Fixed requirements and structure	Flexible and adaptable
Homogeneous teams with strong hierarchy	Network of empowered teams
Project-focused with the aim of reaching the delivery phase	Focused around collaboration and communication
Sequential approach	Incremental approach
Comprehensive	Light
Slow - only at major milestones	Rapid (weekly / bi-weekly)
Low - issues are not identified until the testing phase	Improved - issues identified after each sprint
Increases as project progresses	Decreases as project progresses
Limited and delayed until project completion	Frequent - after each sprint
Straightforward projects in predictable circumstances	Short projects in high-risk situations

DEVOPS AND ITIL

- Patrick Debois, who's often called "the father of DevOps", coined the word "DevOps" in 2009.
- The word "DevOps" is a combination of the words **"development"** and **"operation".**
- DevOps is a collaborative way of developing and deploying software.
- DevOps is a software development method that **stresses communication, collaboration and integration** between software developers and information technology (IT) operation professionals.

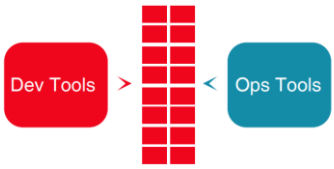


- Developers always **want to deliver change**s as soon as possible.
- Operations want **reliability and stability**.



Wall of Confusion

This wall of confusion not only exists between the mindsets of the two teams but also with the tools they use. **Development uses some tools and operation uses some other tools** to perform the same stuff.



Wall of Confusion

DevOps break down the walls between development and operations team, unifying development to operations for better, faster outcomes.

- DevOps combines development (Dev) and operations (Ops) to increase the efficiency, speed, and security of software development and delivery compared to traditional processes.
- It is defined as a software engineering methodology which aims to integrate the work of development teams and operations teams by facilitating a culture of collaboration and shared responsibility.
- DevOps practices enable software development (dev) and operations (ops) teams to accelerate delivery through automation, collaboration, fast feedback, and iterative improvement.

Core DevOps principles

DevOps is related to the first principle of Agile, "Individuals and interactions over processes and tools." The turnaround for DevOps processes must be fast. DevOps engineers work on making enterprise processes faster, more efficient, and more reliable. Repetitive manual labor, which is error prone, is removed whenever possible. We must keep track of delivering increased business value. increased communication between roles in the organization has clear value. it is useful to be able to deliver incremental improvements of code to the test environments quickly and efficiently. In the test environments, the involved stake holders, such as product owners and, of course, the quality assurance teams, can follow the progress of the development process.

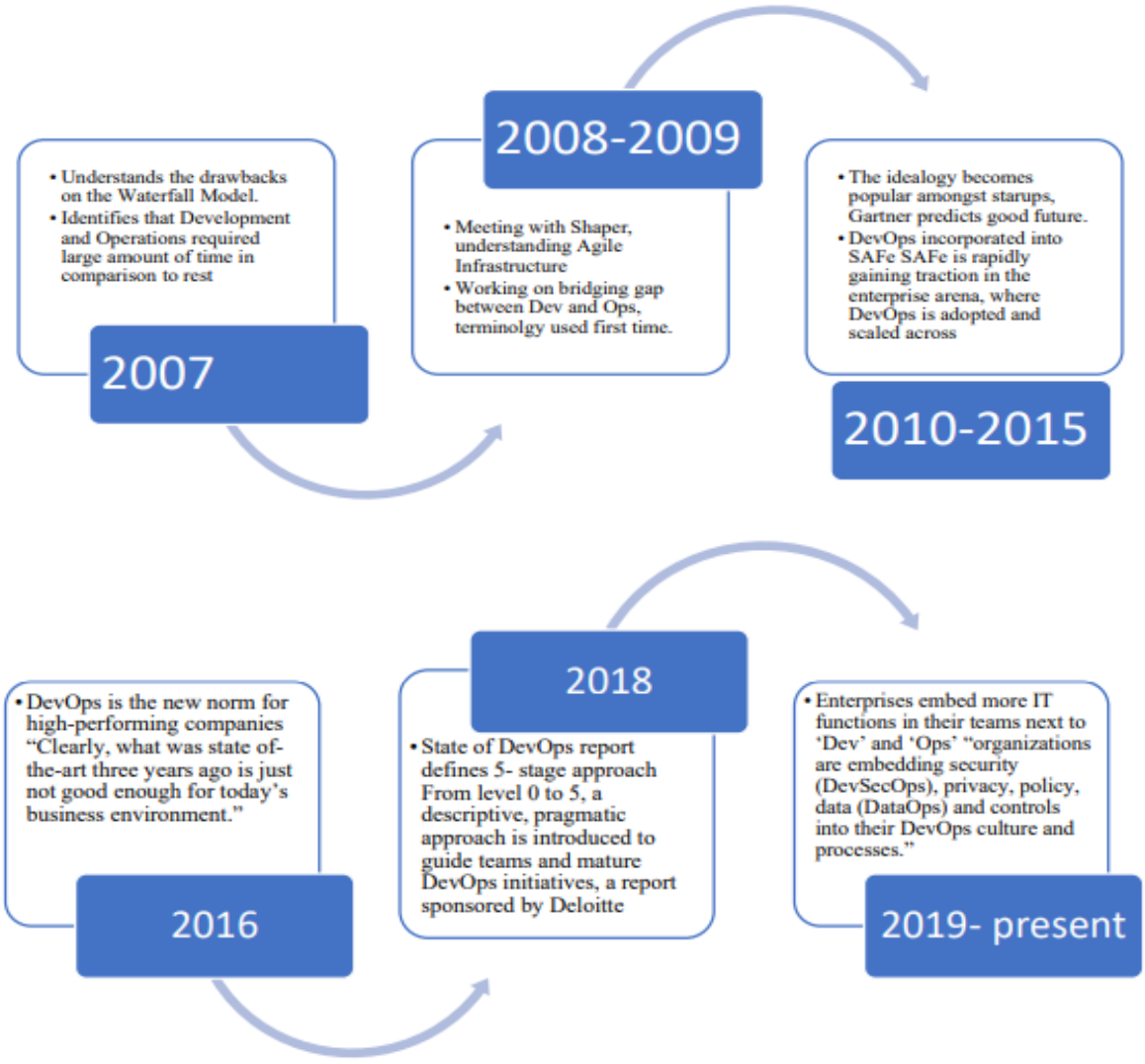
Five Basic Principles of DevOps:

• Eliminate the blame game, Open examinations, Feedback, Rewarding failures
• Continuous Delivery, Monitoring, Configuration Management
• Business value for end user
• Performance Metrics, Logs, Business goals Metrics, People Integration Metrics, KPI
• Ideas, Plans, Goals, Metrics, Complications, Tools

7 Cs of Devops

- Communication
- Collaboration
- Controlled Process
- Continuous Integration
- Continuous Deployment
- Continuous Testing
- Continuous Monitoring

Timelines of Devops



LIFECYCLE OF DEVOPS

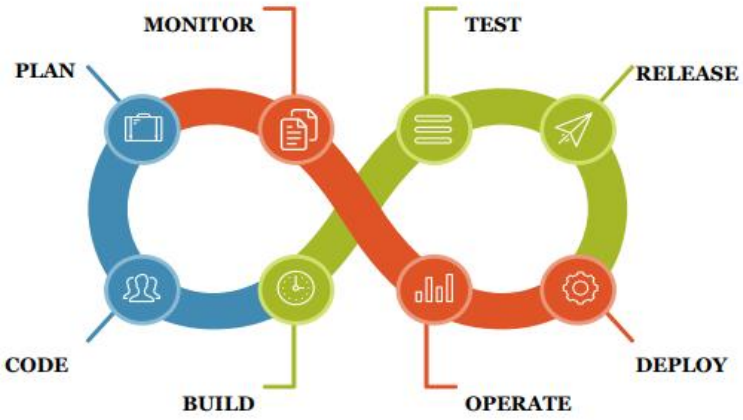
Plan: In DevOps, planning plays a vital function; all the requirements of the project, which include the challenges like time taken for every stage, price, and many other parameters, are discussed, and this can help everyone working on the project teams get an idea of the project.

Code: In this stage, the code is written in line with the client's requirements. The code is split into smaller fragments called Units, which help get a clear picture of the code. For example, on a medical billing system, the login code can be considered as one unit, and billing categories can be regarded as another unit. Tools- JIRA, Git

Build: The units (small codes) are built at this stage. Tools- Gradel, Maven

Test: The team performs tests across all the units to ensure no bugs. If found, it will be returned.

Tools- Pytest, Selenium



Integrate: In this stage, all the units of the codes are integrated, which helps build a connection between the development team and operation team to put into effect Continuous Integration and Continuous Deployment (CICD).

Deploy: The code would be deployed on the client's environment.

Tools- AWS, Docker

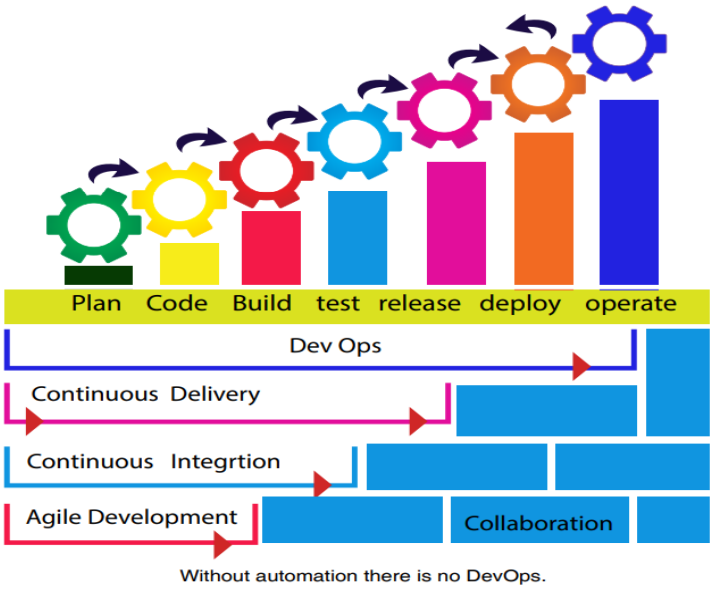
Operate: Any operations on the code are performed in this phase.

Tool- Kubernetes

Monitor: As the application is mounted on the client's environment, the application behavior and the client environment are monitored across different operations making sure of the best application performance.

Tool- Maven

- Automate Provisioning - Infrastructure as Code
- Automate Builds – Continuous Integration
- Automate Deployments – Defined Deployment Pipeline and Continuous Deployments with appropriate configurations for the environments
- Automate Testing – Continuous Testing, Automated tests after each deployment
- Automate Monitoring – Proper monitors in place sending alerts
- Automate Metrics – Performance Metrics, Logs



ITIL:

ITIL (Information Technology Infrastructure Library) is a **framework for IT service management** that aligns IT with business needs. ITIL outlines processes and procedures for managing incidents, service requests, changes, and other aspects of the IT service lifecycle. With its roots dating back to the 1980s, ITIL has evolved through several versions, with ITIL 4 being the latest iteration, emphasizing a flexible, coordinated, and integrated system for effective service management. ITIL improve customer relations, and establish a reliable environment for optimal delivery of IT services.

For example, instead of following a lengthy change management process for every update, changes can be categorized by risk. Low-risk changes, such as minor updates or patches, could be fast-tracked through pre-approved routes, while only high-risk changes, like major releases, go through the full process. This allows for quicker deployment without sacrificing control.

The ITIL framework is built on five core volumes:

1. **Service Strategy:** Focuses on defining the perspective, position, plans, and patterns that a service provider needs to execute to meet an organization's business outcomes.
2. **Service Design:** Covers the design of IT services, including architectures, processes, policies, and documentation.
3. **Service Transition:** Addresses the delivery of services required by a business into operational use, ensuring that changes to services and service management processes are carried out in a coordinated way.
4. **Service Operation:** Includes the practices in the day-to-day operation of services, including fulfilling user requests, resolving service failures, fixing problems, and carrying out routine operational tasks.
5. **Continual Service Improvement:** Focuses on identifying and implementing improvements to IT services that support business processes.

ITIL 4, released in 2019, introduced several new concepts and updated existing ones to align with modern IT practices. It emphasizes the importance of end-to-end value streams.

Pros

- Streamlines IT service delivery
- Increases customer satisfaction
- Allows for better ITSM integration
- Helps align IT goals with company objectives
- Optimizes IT usage

Cons

- Less flexibility
- Can be difficult to learn
- Costly to implement
- Can be challenging to integrate if you're unfamiliar with ITIL

DevOps — A Methodical Approach

- **Increase Deployment Frequency:** DevOps allows you to deploy small software releases regularly, making it easy to fix any errors that pop up.
- **Resolve Incidents:** Use automated systems to locate and resolve simple errors. You can also avoid shifting blame when larger incidents occur with a well-integrated communications system across teams.

ITIL — A Structured Approach

- **Reduce Workloads:** Reduce IT support desk activities with efficient ITSM, which involves offering self-help options for customers to minimize tickets.
- **Increase Customer Satisfaction:** Alleviate customer confusion with SLAs (Service Level Agreements), so customers know exactly what to expect from you.

Difference Between Devops and ITIL

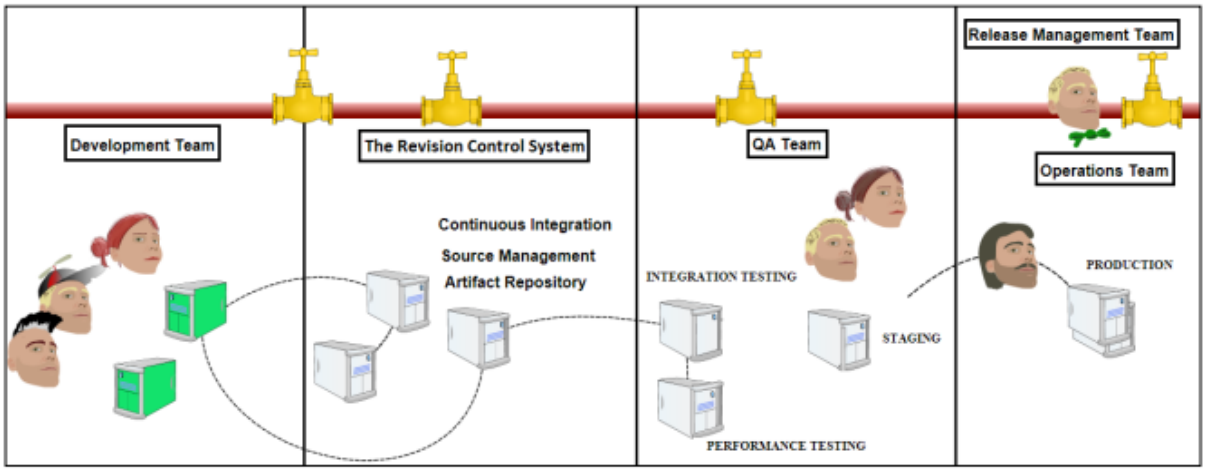
DevOps	ITIL
DevOps refers to effective collaboration between the development team and operations team.	ITIL refers to a set of detailed guidelines for effective and efficient management of an organization's IT services.
Use a methodical approach to minimize the friction between two teams.	It uses a systematic approach to manage the IT service to ensure growth.
CI and CD are the backbone of modern DevOps philosophy.	ITIL aims to increase the delivery process.
Continuous integration and continuous delivery are critical to increasing.	Services are built, discuss, tested, and implemented.
DevOps focus on the concept. It has a dynamic body of knowledge.	DevOps focus on the concept. It has a dynamic body of knowledge. ITIL focus on development. It has a static body of knowledge.

DevOps process and Continuous Delivery

It is set of practices, tools that automate & integrate the processes between software development & IT teams. It focuses on closing the loops between development and operation & driving product thigh continuous development, integration, testing, monitoring, feedback, delivery & deployment.

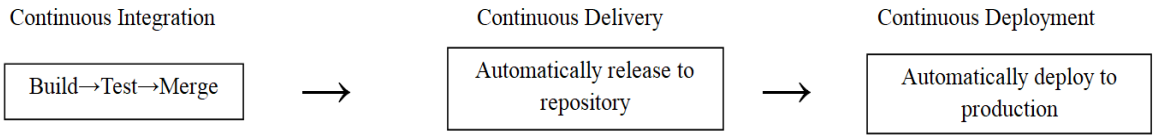
Continuous Delivery (CD) is the process of automating build, test, configuration & deployment from a build to production environment. CD value has become mandatory requirement for organization to deliver value to end users.

A release pipeline can create multiple testing or staging environment to automate infrastructure creation & deploy new builds. Example of CD pipeline in large organization is introduced is the following diagram:



The developers

- The developers work on their workstations.
  They develop code and need many tools to be efficient.
- Ideally, they would each have production-like environments available to work with locally on their workstations or laptops.
- Depending on the type of software that is being developed, this might actually be possible, but it's more common to simulate, or rather, mock, the parts of the production environments that are hard to replicate.
- For example, be the case for dependencies such as external payment systems or phone hardware.
- CD is a lean practice with a goal to keep product fresh with fastest path from new code or component availability to deployment.
- CD makes up a part of CI/CD a method to frequently deliver software by automating some of stages of application development.
- CI is where new code changes to an application that are regularly built tested & merged into shared repository.



Revision control system

- The revision control system is often the heart of the development environment.
- The code that forms the organization's software products is stored here.

- It is also common to store the configurations that form the infrastructure here.
- If you are working with hardware development, the designs might also be stored in the revision control system.

The build server

- The build server is conceptually simple.
- It might be seen as a glorified egg timer that builds your source code at regular intervals or on different triggers.
- The most common usage pattern is to have the build server listen to changes in the revision control system.
- When a change is noticed, the build server updates its local copy of the source from the revision control system. Then, it builds the source and performs optional tests to verify the quality of the changes. This process is called Continuous Integration.

QA Team / Test Environment:

- After the build server has stored the artifacts in the binary repository, they can be installed from there into test environments.
- Test environments should normally attempt to be as production-like as is feasible. Therefore, it is desirable that the they be installed and configured with the same methods as production servers.

Staging:

- Staging environments are the last line of test environments. They are interchangeable with production environments. You install your new releases on the staging servers, check that everything works, and then swap out your old production servers and replace them with the staging servers, which will then become the new production servers. This is sometimes called the blue-green deployment strategy.
- The exact details of how to perform this style of deployment depend on the product being deployed. Sometimes, it is not possible to have several production systems running in parallel, usually because production systems are very expensive. At the other end of the spectrum, we might have many hundreds of production systems in a pool. We can then gradually roll out new releases in the pool. Logged-in users stay with the version running on the server they are logged in to. New users log in to servers running new versions of the software.

Artifact repository

- When the build server has verified the quality of the code and compiled it into deliverables, it is useful to store the compiled binary artifacts in a repository. This is normally not the same as the revision control system. In essence, these binary code repositories are filesystems that are accessible over the HTTP protocol. Normally, they provide features for searching and indexing as well as storing metadata, such as various type identifiers and version information about the artifacts.
- In the Java world, a pretty common choice is Sonatype Nexus. Nexus is not limited to Java artifacts, such as Jars or Ears, but can also store artifacts of the operating system type, such as RPMs, artifacts suitable for JavaScript development, and so on. Amazon S3 is a key-value datastore that can be used to store binary artifacts. Some build systems, such as Atlassian Bamboo, can use Amazon S3 to store artifacts. The S3 protocol is open, and there are open source implementations that can be deployed inside your own

network. One such possibility is the Ceph distributed filesystem, which provides an S3-compatible object store.

Package managers

Red Hat-like systems use a package format called RPM. Debian-like systems use the .deb format, which is a different package format with similar abilities. The deliverables can then be installed on servers with a command that fetches them from a binary repository. These commands are called package managers.

On Red Hat systems, the command is called yum, or, more recently, dnf. On Debian-like systems, it is called aptitude/dpkg. The great benefit of these package management systems is that it is easy to install and upgrade a package; dependencies are installed automatically. If you don't have a more advanced system in place, it would be feasible to log in to each server remotely and then type yum upgrade on each one. The newest packages would then be fetched from the binary repository and installed. Of course, as we will see, we do indeed have more advanced systems of deployment available; therefore, we won't need to perform manual upgrades.

Operation Team

Operation team is like the final aspect of CD pipeline. It will maintain the deployed production by the development team.

RELEASE MANAGEMENT

- Release management is the process of planning, Scheduling, coordinating, and management of software versions, from development to production deployment (new features, bug fixes, and enhancements) in a reliable, efficient, and timely manner.
- It aims to minimize risk, ensure quality, and maximize the value of software releases to end-users.
- It emphasizes automation, collaboration, and traceability across CI/CD pipelines to enable frequent and reliable software releases.

Goals of Release Management

The goals of Release Management in DevOps services are focused on **reducing risk, increasing efficiency, and enhancing collaboration** between development and operations teams. It ensures that software releases are **delivered on time, with minimal disruption to operations, and at a high level of quality**.

Key components of Release Management

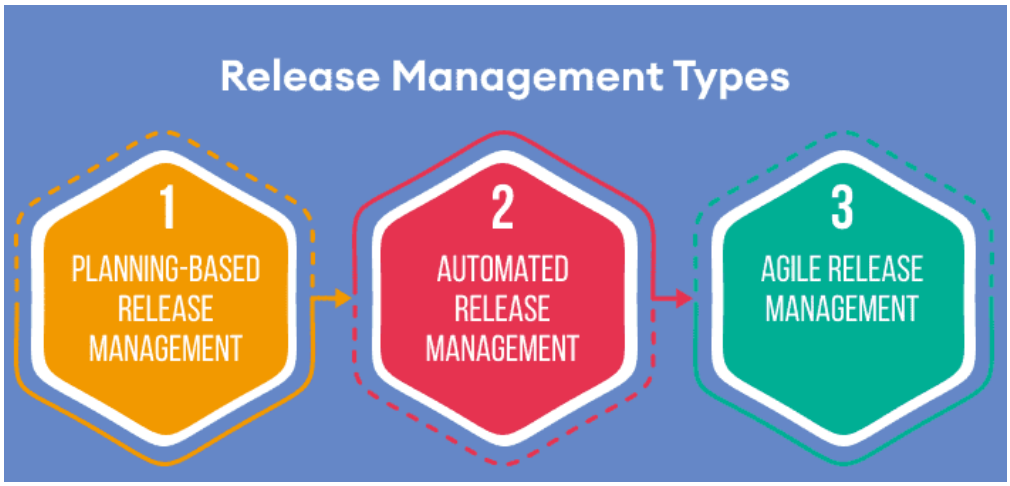
The key components of Release Management include:

- **Version control:** Version control involves keeping track of different versions of code and software to ensure that developers are working on the correct version of the code.
- **Change management:** Change management involves tracking and managing changes to software releases, ensuring that changes are made in a controlled and systematic manner.
- **Deployment management:** Deployment management involves planning and executing the deployment of software releases to production environments. Deployment system usually has a way to support how to describe which software version to use is different environment.
- **Release automation:** Release automation involves using automated tools and processes to streamline and optimize the software delivery pipeline.

Benefits

- **Improved reliability and stability** of software releases by **identifying and fixing bugs**, **reducing errors**, and **ensuring consistent performance** across different environments.
- **Increased collaboration** between development and operations teams, enabling **faster feedback loops, and better communication and coordination** between teams.
- **Faster time to market** by streamlining the software delivery pipeline, reducing manual processes, and automating testing and deployment.
- **Enhanced customer satisfaction** by delivering high-quality software releases that meet customer needs and expectations.
- **Cost savings and greater efficiency** by reducing downtime, minimizing errors, and optimizing resource utilization.

Release Management Types



Planning-based Release Management
It involves a **structured approach** to planning, testing, and deploying software releases. This type of Release Management is suitable for **large and complex projects** with long release cycles.

Automated Release Management
It involves the **use of automated tools and processes** to manage the entire software delivery pipeline, from development to deployment. This type of Release Management is ideal for organizations that **require frequent and rapid releases**.

Agile Release Management
It is a **flexible and iterative approach** to Release Management, focused on delivering incremental changes and continuous improvements to software releases.
This type of Release Management is best suited for organizations that **require quick and frequent releases**, with **the ability to adapt to changing requirements and feedback**.

Continuous Integration and Continuous Deployment (CI/CD)

- Continuous integration (CI) and continuous deployment (CD) involve **automating the software build, test, and deployment process** to reduce the time it takes to deliver software updates.
- CI/CD ensures that **new code is integrated into the existing codebase quickly and efficiently,** with minimal disruption to the development cycle. DevOps CI/CD services can help organizations implement CI/CD best practices.

Testing and Quality Assurance

- Quality assurance **ensures that software releases meet the required quality standards**
- Testing helps to **identify and fix bugs** before they reach the end user.

Version Control and Change Management

- Version control and change management involve **managing changes to software code and tracking revisions** to ensure that developers are working with the latest version of the code.
- Version control also allows developers **to collaborate effectively on code changes**, **reducing the risk of conflicts and errors**.

Documentation

- Documentation helps to ensure that everyone involved in the software development process has access to up-to-date information and instructions.
- Documentation also helps to maintain consistency and clarity across the development process, reducing the risk of errors and miscommunications.

Communication and Collaboration

- Effective communication and collaboration involve ensuring that all team members are **aware of their responsibilities and deadlines** and that everyone is working towards a common goal.
- It helps to reduce the risk of misunderstandings and errors and ensure that software releases are delivered on time and to the required quality standards.

Monitoring and Metrics

- Monitoring and metrics involve tracking the performance of software releases and identifying any issues that arise.
- It also helps to ensure that software releases meet the required performance standards and enable organizations to identify opportunities for improvement.

Advantages

- Aligning business & IT goals
- Minimizes organizational risk
- Direct accelerating change

SCRUM, KANBAN, AND THE DELIVERY PIPELINE

Scrum:

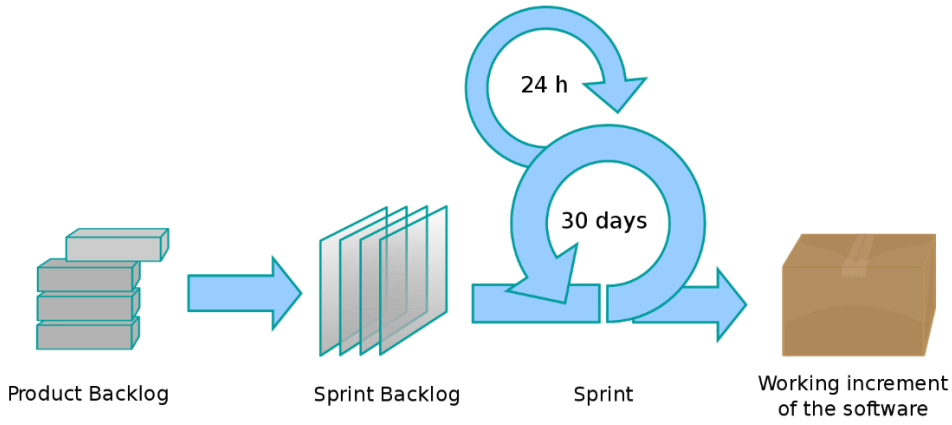
- Scrum is an Agile framework used to **manage complex projects**.
- It focuses on **delivering incremental improvements** through iterative cycles called **sprints**, typically lasting which can occur biweekly or monthly.
- Scrum emphasizes collaboration, accountability, and iterative progress towards a well-defined goal.

There are three primary roles and each role has its own specific responsibilities:

- The **product owner**, **who decides what to build**.
- The **Scrum master, who makes sure the team is using Scrum** effectively and efficiently;
- The **delivery team**, which is **responsible for delivering the final product**.

Key Elements :

- Roles : Product Owner, Scrum Master, and Development Team.
- Artifacts : Product Backlog, Sprint Backlog, and Increment.
- Services : Sprint Planning, Daily Stand-ups, Sprint Review, and Sprint Retrospective.



Kanban:

- Kanban is a **Japanese term** that means **signboard or billboard**.
- Kanban is a **visual workflow management method** that aims to improve efficiency by **visualizing tasks, limiting work-in-progress, and optimizing flow**.
- Unlike Scrum, Kanban is **continuous** and does not prescribe specific roles.

Key Elements :

- Visual Board: Uses columns (e.g., To Do, In Progress, Done) to represent the flow of tasks.
- Work-In-Progress (WIP) Limits: Limits the number of tasks in each stage to avoid overloading the team.
- Continuous Improvement: Focuses on incremental changes to improve efficiency and flow.

Both Scrum and Kanban have their own strengths and are suitable for different types of projects. Scrum is ideal for projects with well-defined goals and a need for frequent team collaboration, while Kanban is better suited for ongoing, operational work where flexibility and continuous delivery are key.

Scrum	Kanban
Scrum helps teams and organizations deliver products in short iterative cycles.	Kanban uses a visual control mechanism to track work.
The roles of team members are not so clearly defined in Kanban.	The roles of team members are not so clearly defined in Kanban.
Continuous delivery is followed by the successful completion of each sprint.	Products and processes are delivered continuously on an as-needed basis.
It uses velocity as the primary metric for measuring productivity.	Productivity is measured using cycle time instead of velocity.
The development team pulls the entire batch for each iteration.	New tasks are pulled as soon as there is room for a new task to be pulled.

From a DevOps perspective, a change starts propagating through the Continuous Delivery pipeline toward test systems and beyond when it is deemed ready enough to start that journey. This might be judged on subjective measurements or objective ones, such as "all unit tests are green." Our pipeline can manage both the following types of scenarios:

- The **build server** supports the **generation of the objective code quality metrics** that we need in order to make decisions. These decisions can either be made automatically or be the basis for manual decisions.
- The **deployment pipeline** can also be directed manually. This can be handled with an issue management system, via configuration code commits, or both. So, again, from a DevOps perspective, it doesn't really matter if we use Scrum, Scaled Agile Framework, Kanban, or another method within the lean or Agile frameworks. Even a traditional Waterfall process can be successfully managed— DevOps serves all!

IDENTIFYING BOTTLENECKS

There is a lot going on for any change that propagates through the pipeline from development to production. It is important for this process to be efficient. As with all Agile work, keep track of what you are doing, and try to identify problem areas.

When everything is working as it should, a commit to the code repository should result in the change being deployed to integration test servers within a 15-minute time span. When things are not working well, a deploy can take days of unexpected disturbances.

Here are some possible causes:

- Database schema changes.
- Test data doesn't match expectations.
- Deploys are person dependent, and the person wasn't available.
- There is unnecessary red tape associated with propagating changes.
- Your changes aren't small and therefore require a lot of work to deploy safely. This might be because your architecture is basically a monolith.