III B.Tech I Sem CSE

23CS503 : DevOps

Unit-I

Introduction to DevOps

Mr. P. Thirupathi
Assistant Professor, Department of CSE NRCM

Introduction to DevOps

Definition: **DevOps** is a modern way of working in software development in which the **development team** (who writes the code and builds the software) and the **operations team** (which sets up, runs, and manages the software) work together as a single team.

Before DevOps, the development and operations teams worked separately. This caused:

- Delays in launching software
- Miscommunication between teams
- Slow fixing of problems

Why DevOps?

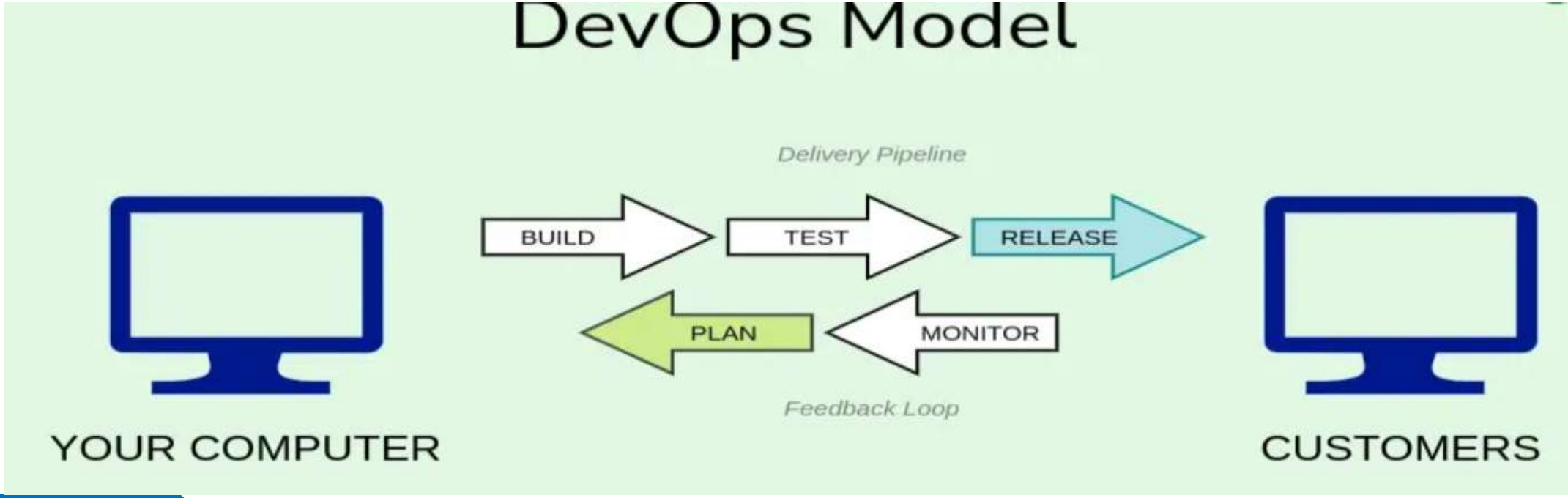
DevOps was created to resolve these issues by making both Development and Operations teams work together in entire software development lifecycle.

The following are some reasons why DevOps was needed:

1. **Faster Delivery of Software**
2. **Better Teamwork and Communication**
3. **More Reliable Software with Fewer Errors**
4. **Automation Saves Time and Reduces Errors**
5. **Helps Businesses Be More Flexible and Competitive**
6. **Better Experience for Customers**

DevOps Model Defined

DevOps is a software development approach that emphasizes collaboration and communication between development (Dev) and operations (Ops) teams. It aims to shorten the software development lifecycle and improve the quality and reliability of software releases.



History of DevOps

2007-2008:

The initial seeds of DevOps were planted as IT operations and software development communities started raising concerns about the traditional model.

2009:

Patrick Debois organized the first DevOpsDays conference in Ghent, Belgium, where the term "DevOps" was first publicly used.

Early 2010s:

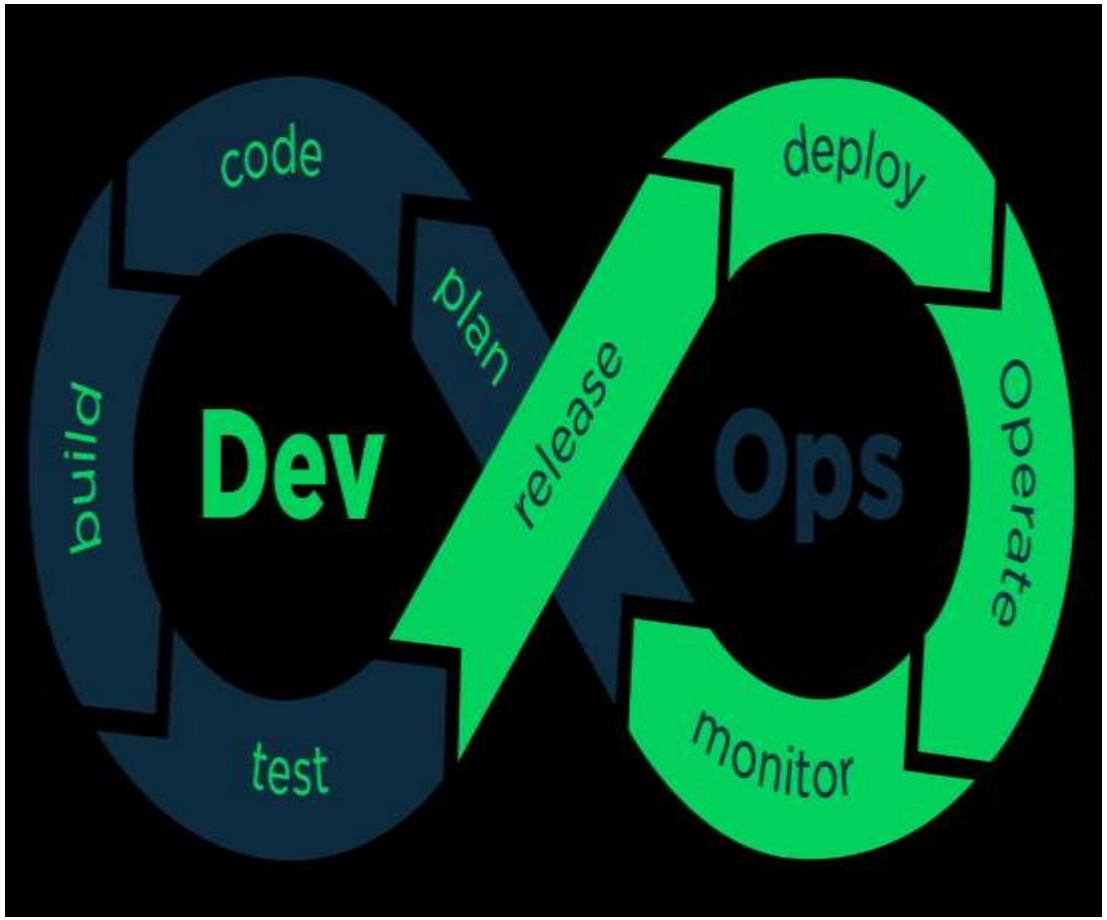
DevOps gained momentum with the publication of books like "The Phoenix Project" and the rise of continuous delivery practices.

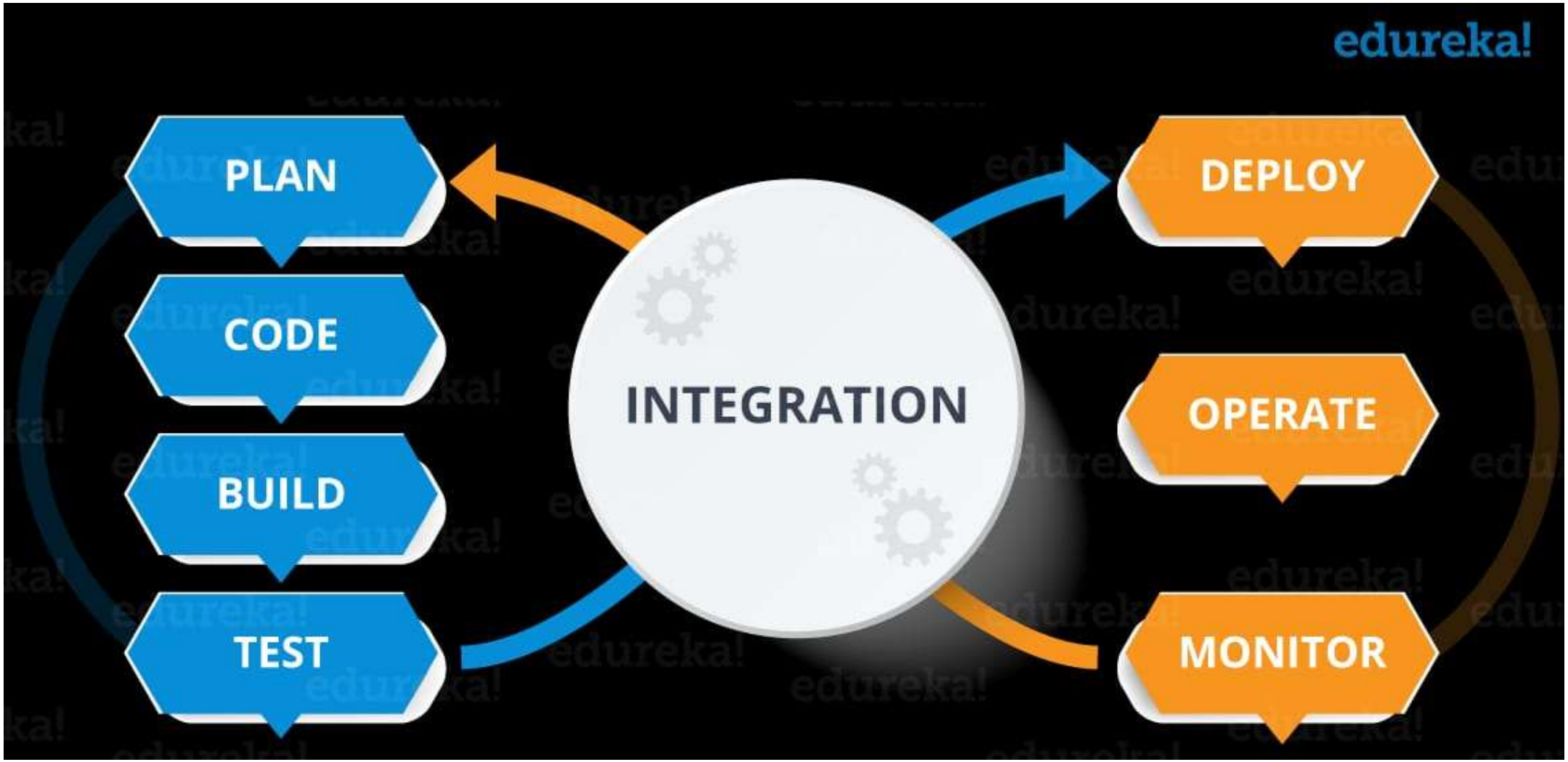
Later Years:

DevOps principles and practices have become widely adopted by organizations of all sizes, driving digital transformation and enabling faster, more reliable software

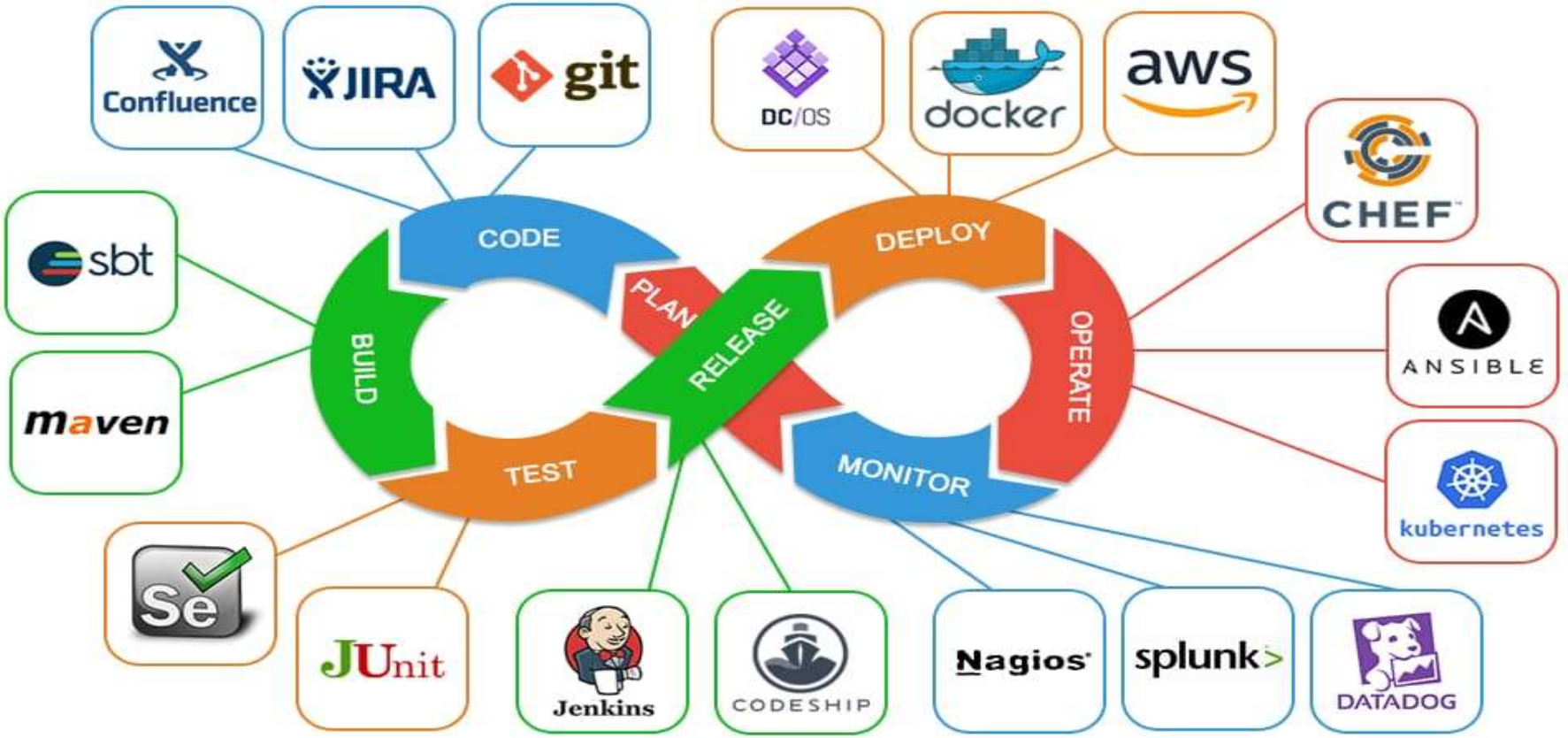
Stages in DevOps

1.Plan
2.Code
3.Build
4.Test
5.Release
6.Deploy
7.Operate
8.Monitor




edureka!
PLAN
CODE
BUILD
TEST
INTEGRATION
DEPLOY
OPERATE
MONITOR

Automation Tools in DevOps



The Waterfall model

Waterfall model is a linear, sequential approach to software development, where each phase is completed before moving on to the next one. It's like a waterfall, where each stage flows into the next, without going back.

- **1. Requirements gathering**
- **2. Design**
- **3. Implementation (coding)**
- **4. Testing**
- **5. Deployment**
- **6. Maintenance**

The Waterfall model is easy to understand and manage, but it can be inflexible and prone to errors if requirements change during the development process.

Advantages:

- **Easy to understand and manage**: The Waterfall model is a linear approach, making it simple to comprehend and manage.
- **Clear requirements:** Requirements are defined upfront, reducing ambiguity.
- **Predictable timeline**: The sequential nature allows for a predictable timeline.

Disadvantages:

- **Inflexible:** The Waterfall model doesn't accommodate changes in requirements well.
- **High risk:** If requirements are incorrect or incomplete, the entire project can be impacted.
- **Testing is delayed**: Testing occurs only after implementation, which can lead to costly rework.

The Agile model

The Agile model is an iterative and incremental approach to software development that emphasizes flexibility, collaboration, and rapid delivery. It's based on the Agile Manifesto, which values:

1. **Individuals and interactions**
2. **Working software**
3. **Customer collaboration**
4. **Responding to change**

Key principles:

1. **Iterative development:** Break down work into small, manageable chunks (sprints)
2. **Continuous improvement**: Regularly reflect and improve processes
3. **Flexibility:** Embrace change and adapt to new requirements
4. **Collaboration:** Cross-functional teams work together closely

Advantages:

- **Faster time-to-market:** Agile's iterative approach enables quicker delivery
- **Improved adaptability:** Agile teams respond quickly to changing requirements
- **Increased customer satisfaction:** Regular feedback and involvement ensure customer needs are met

Disadvantages:

- **Lack of Documentation**

Agile focuses more on working software than on comprehensive documentation.

- **Scope Creep**

Due to flexible requirements, there's a risk of uncontrolled changes or feature overload.

- **Requires Experienced Team**

Agile teams need strong decision-making and technical skills to be effective.

- **Customer Availability**

Agile relies on regular input from customers, which may not always be feasible.

Aspect	Waterfall Model	Agile Model
Approach	Linear and sequential	Iterative and incremental
Process Flow	Each phase must be completed before the next starts	Development is divided into small sprints or cycles
Flexibility	Inflexible to changes after the requirement phase	Highly flexible; changes can be made anytime
Customer Involvement	Minimal once requirements are gathered	Continuous involvement throughout development
Testing	Performed after the build phase	Testing happens in every sprint
Delivery	Delivered once after the full project is complete	Delivered in parts after each sprint
Requirements	Defined at the beginning and fixed	Can evolve over time
Best Suited For	Projects with clearly defined, unchanging requirements	Projects with evolving or unclear requirements
Project Size	Suitable for small to medium projects	Suitable for all project sizes, especially large
Examples	Construction, Manufacturing	Software development, Startups, Product iterations

Aspect	Agile	DevOps
Definition	A software development methodology focusing on iterative, incremental development.	A set of practices that combines development (Dev) and operations (Ops) to shorten the software lifecycle.
Goal	Rapid delivery of high-quality software that meets customer needs.	Faster deployment and stable operation of software in production.
Focus	**Development** side: collaboration, flexibility, and feedback.	**Both Development and Operations**: automation, integration, and continuous delivery.
Team Structure	Small, cross-functional, mainly development teams.	Unified team involving **developers**, **testers**, and **IT operations**.
Key Practices	Scrum, Kanban, User Stories, Backlogs, Sprint planning.	CI/CD, Infrastructure as Code (IaC), Automation, Monitoring.
Release Cycle	Short, time-boxed sprints (usually 2–4 weeks).	Continuous delivery or deployment.
Feedback	Customer feedback after each sprint.	Continuous feedback from **monitoring and logs**.
Automation	Not mandatory, but used in testing.	Core component (CI/CD, test, deploy, monitor).
Tools	JIRA, Trello, Confluence	Jenkins, Docker, Kubernetes, Git, Ansible, Prometheus
Cultural Aspect	Encourages collaboration between dev and customers.	Encourages collaboration between dev and ops teams.

ITIL

ITIL stands for **Information Technology Infrastructure Library**. It is a globally recognized framework that provides **best practices for delivering IT services** effectively and efficiently. ITIL helps organizations manage risk, strengthen customer relations, establish cost-effective practices, and build a stable IT environment that allows for growth, scale, and change.

- A service provided to customers that delivers value by facilitating outcomes they want to achieve without the ownership of specific costs and risks.

ITIL Service Lifecycle

1. Service Strategy
2. Service Design
3. Service Transition
4. Service Operation
5. Continual Service Improvement (CSI)

1.Service Strategy

- This phase focuses on aligning IT services with the overall business strategy and defining the services that will be offered. It involves understanding business needs, defining service value, and making strategic decisions about which services to provide.

2.Service Design

- This phase is concerned with the design and development of new or changed IT services. It involves designing the service, its architecture, and the supporting processes, policies, and documentation.

3. Service Transition

- This phase focuses on the transition of new or changed services into the live environment. It includes activities like planning the transition, managing changes, testing the service, and deploying it to users.

4. Service Operation

- This phase focuses on the day-to-day management of live IT services. It includes activities like incident management, problem management, request fulfillment, and access management.

5. Continual Service Improvement (CSI)

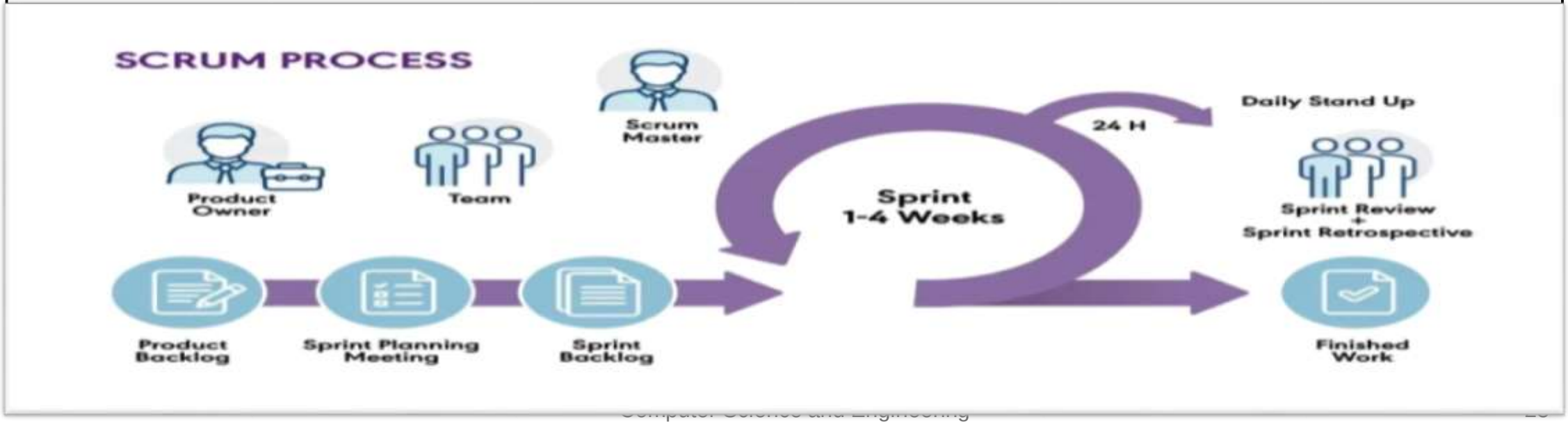
- This phase focuses on continuously improving the quality and efficiency of IT services. It involves monitoring service performance, identifying areas for improvement, and implementing changes to optimize service delivery.

Benefits of ITIL

- Improved service delivery and customer satisfaction
- Better alignment between IT and business
- Efficient utilization of resources
- Reduced service disruption and downtime
- Clear roles and responsibilities in IT operations

Scrum

Scrum is an Agile framework for managing complex work, particularly in software development. It emphasizes iterative development, collaboration, and continuous improvement. A Scrum team typically consists of a Product Owner, Scrum Master, and Developers, working in time-boxed iterations called Sprints.



Main roles of Scrum

- **Product Owner:** Defines and prioritizes the product backlog.
- **Scrum Master:** Facilitates the Scrum process and helps the team remove impediments.
- **Developers:** The team members who build the product.

Artifacts:

- **Product Backlog:** A prioritized list of features and requirements for the product.
- **Sprint Backlog:** A subset of the product backlog selected for a specific sprint.
- **Increment:** The potentially shippable product increment created during a sprint.

Events

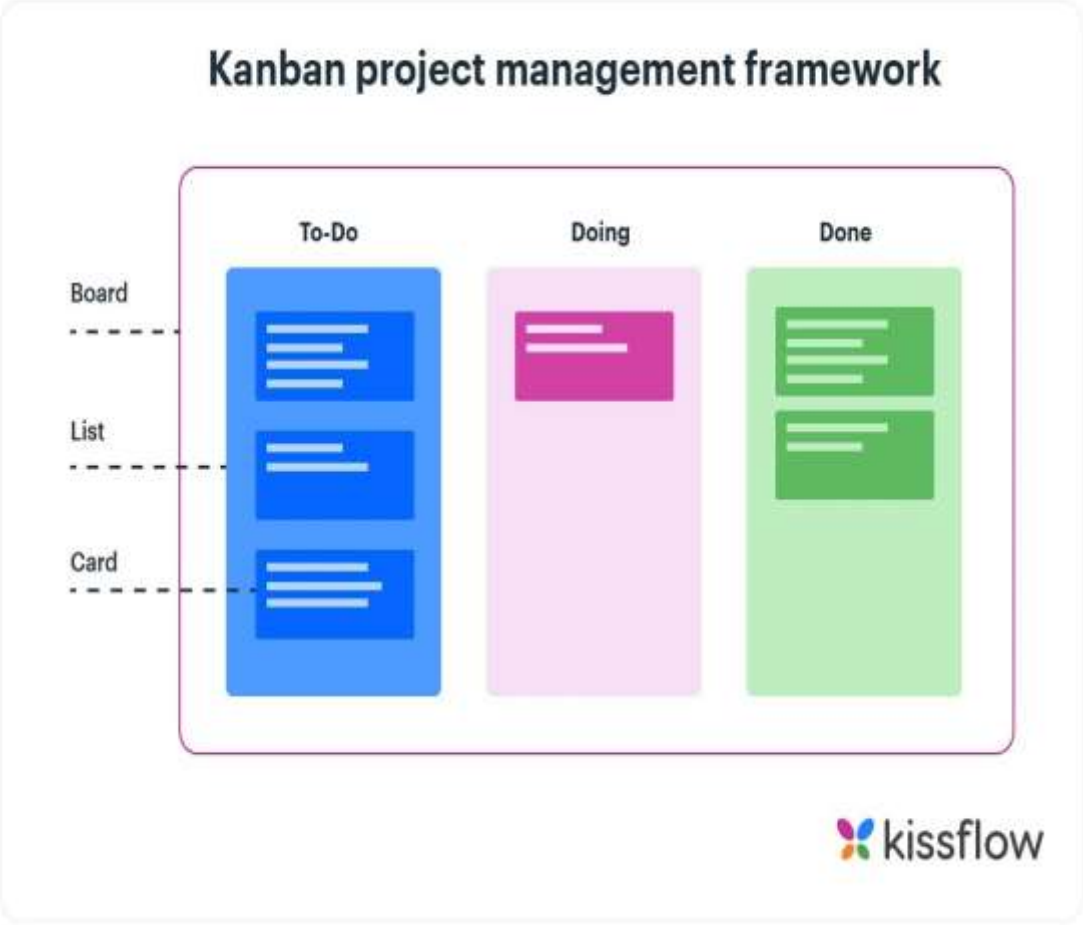
- **Sprint Planning:** The team selects work from the product backlog for the sprint.
- **Daily Scrum:** A short daily meeting where the team synchronizes on progress.
- **Sprint Review:** The team demonstrates the completed work to stakeholders.
- **Sprint Retrospective:** The team reflects on the sprint and identifies areas for improvement.

How Scrum Works:

1. **Product Backlog Creation:** The Product Owner creates and maintains a prioritized list of features and requirements.
2. **Sprint Planning:** The team selects items from the product backlog for a specific sprint (typically 1-4 weeks).
3. **Sprint Execution:** The team works on the selected items, holding daily Scrum meetings to track progress.
4. **Sprint Review:** The team demonstrates the completed work and gathers feedback.
5. **Sprint Retrospective:** The team reflects on the sprint and identifies areas for improvement.
6. **Iteration:** The process repeats with a new sprint, continuously delivering value.

Kanban

Kanban is a visual system for managing and optimizing work flow, originating from Toyota's lean manufacturing practices and now widely used in various industries, including software development. It uses a Kanban board to visualize work items and their progression through different stages, enabling teams to identify bottlenecks and improve efficiency.



Core Components of Kanban

Kanban Cards:

These represent individual work items, such as tasks, user stories, or bugs, and contain details about the work.

Columns:

These represent different stages of the workflow, like "To Do," "In Progress," and "Done," or more granular steps depending on the process

Swim lanes:

Optional horizontal lanes that can be used to categorize work items, like by team, type of work, or priority

How it Works:

1. Visualize Work:

Teams visualize their workflow by moving cards across the columns as work progresses through the different stages.

2. Limit Work in Progress (WIP):

Kanban boards can be configured with WIP limits to prevent teams from taking on too much work at once, promoting focus and efficiency.

3. Continuous Improvement:

By tracking the flow of work and identifying bottlenecks, teams can make adjustments to optimize their processes and improve cycle times.

Key Benefits:

Increased Visibility:

Provides a clear overview of all work items and their current status.

Improved Workflow:

Helps teams identify and address bottlenecks, leading to smoother and faster workflows.

Enhanced Collaboration:

Makes it easier for team members to understand what others are working on and collaborate effectively.

Reduced Waste:

By limiting WIP and focusing on delivering value, teams can minimize waste and maximize efficiency.

- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- Common Bottle Necks In Devops

Continuous Integration(CI)

Developers regularly merge their code changes into shared repository, triggering automated builds and tests.

Dev → code check-in → source rep→ CI Serv(GitHub)→ result→ again repeat

Continuous Integration (CI)

- **Frequent Integration:**
Developers merge their code changes into a central repository (like Git) multiple times a day.
- **Automated Builds and Tests:**
Every integration triggers automated builds and tests to verify the changes and identify potential issues early.
- **Early Bug Detection:**
By catching errors quickly, CI helps avoid integration challenges and reduces the risk of releasing buggy code.
- **Key Steps:**
Involves committing code, building the application, running automated tests (unit, integration, etc.), and reporting results

Continuous Integration(CD)

Continuous Delivery extends CI by automating the entire software release process up to the point of deployment. After the code successfully passes CI's automated builds and tests, it is automatically prepared for release.

Dev → code check-in → source rep→ CI Serv(GitHub)→ result→ CI (succ)→ Mock Serv(Build, test)→placed in to final server

Continuous Delivery (CD)

- **Automated packaging and artifact creation:**
  The application and its dependencies are packaged into deployable artifacts.
- **Automated deployment to staging/testing environments**:
The prepared artifacts are automatically deployed to environments that mirror production, allowing for further testing, performance analysis, and user acceptance testing (UAT).
- **Readiness for production deployment:**
While the deployment to production itself might still require manual approval in Continuous Delivery, the system ensures that a deployable artifact is always available and ready to be pushed to production at any time.

Continuous Delivery (CD)

- **Automated packaging and artifact creation:**
  The application and its dependencies are packaged into deployable artifacts.
- **Automated deployment to staging/testing environments**:
The prepared artifacts are automatically deployed to environments that mirror production, allowing for further testing, performance analysis, and user acceptance testing (UAT).
- **Readiness for production deployment:**
While the deployment to production itself might still require manual approval in Continuous Delivery, the system ensures that a deployable artifact is always available and ready to be pushed to production at any time.

Release management in DevOps

What is release management?

Release management is a structured model that refers to the process of planning, designing, scheduling, testing, deploying, and controlling software releases. It ensures that release teams efficiently deliver the applications and upgrades required by the business while maintaining the integrity of the existing production environment.

Release management in DevOps

What is release management?

Release management is a structured model that refers to the process of planning, designing, scheduling, testing, deploying, and controlling software releases. It ensures that release teams efficiently deliver the applications and upgrades required by the business while maintaining the integrity of the existing production environment.

Release management in DevOps

What is release management?

Release management is a structured model that refers to the process of planning, designing, scheduling, testing, deploying, and controlling software releases. It ensures that release teams efficiently deliver the applications and upgrades required by the business while maintaining the integrity of the existing production environment.

Release management in DevOps

What is release management?

Release management is a structured model that refers to the process of planning, designing, scheduling, testing, deploying, and controlling software releases. It ensures that release teams efficiently deliver the applications and upgrades required by the business while maintaining the integrity of the existing production environment.

Benefits of DevOps Release Management

- **Faster Time to Market:**

Streamlined processes and automation enable faster and more frequent releases, allowing businesses to respond quickly to market changes and user demands.

- **Reduced Risk:**

Automated testing, consistent environments, and clear rollback plans minimize the risk of deploying faulty releases.

- **Improved Quality:**

Focus on quality throughout the release cycle, from development to deployment, leads to higher quality software.

- **Increased Collaboration:**

Breaking down silos between teams and fostering communication leads to better coordination and more efficient releases.

- **Cost Efficiency:**

Automation reduces manual effort and associated costs, while faster releases can lead to quicker revenue generation.