DEVOPS

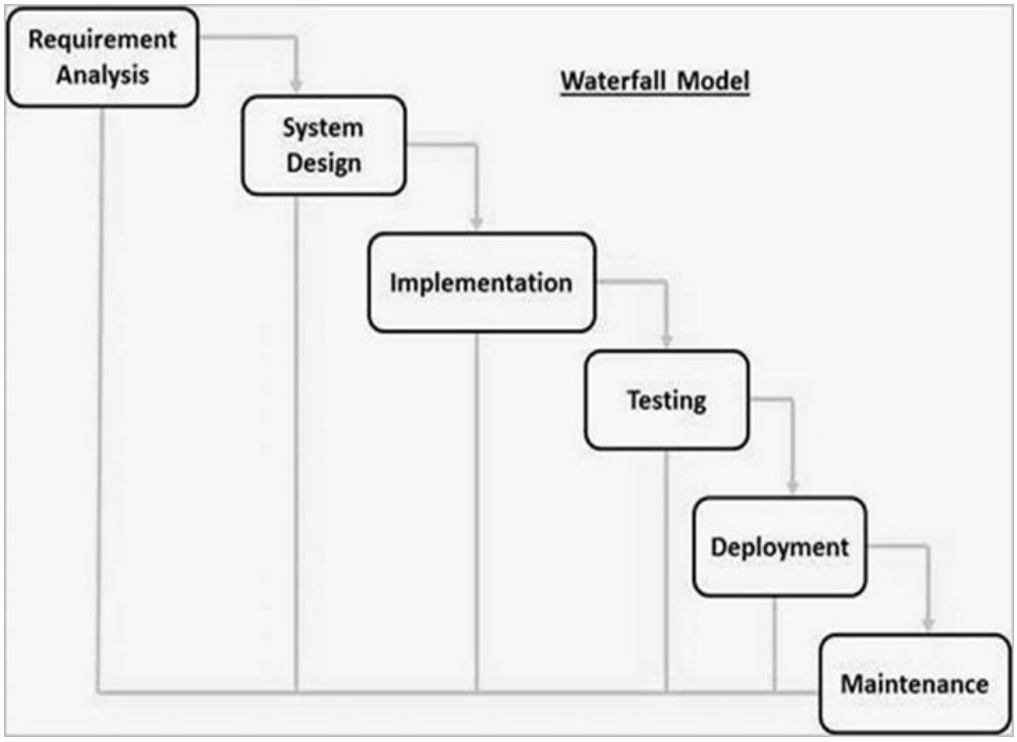
UNIT - I

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

Waterfall Model

The Waterfall model is a linear sequential approach to software development, where the process flows downwards through the phases of requirements gathering, design, implementation, testing, deployment, and maintenance. Each phase of the development process is completed before moving onto the next phase, with no overlapping of phases.

The Waterfall model is the earliest SDLC approach that was used for software development.



- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

- **System Design** – the requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

- **Implementation** – with inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

- **Integration and Testing** – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

- **Deployment of system** – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

- **Maintenance** – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Advantages

1. It's very simple and easy to implement
2. Best suitable for small projects
3. Best suitable if requirements are fixed

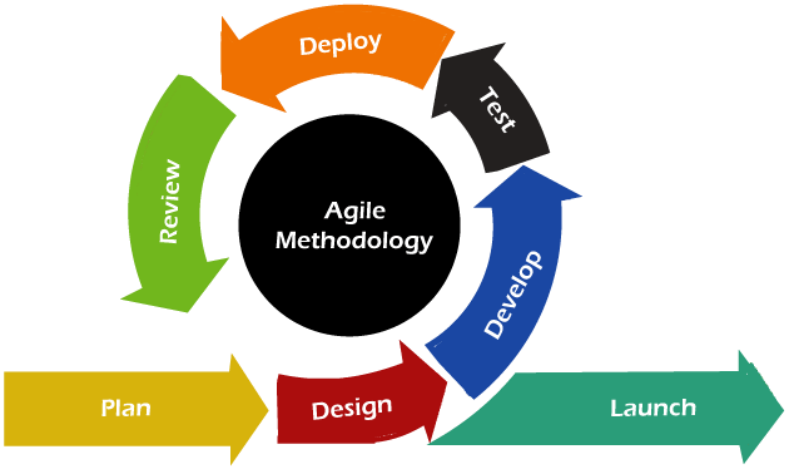
Limitations

1. The Development time will increase
2. The Cost of development will increase
3. It won't accept requirement changes in the middle
4. Client satisfaction is very low
5. Bug fixing is very costly because we can't identify bugs in early stages of life cycle
6. Not at all suitable if requirements keep on changing
7. Not suitable for large projects

Agile Model

The meaning of Agile is swift or versatile."**Agile process model**" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning.

Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.



Iteration

Phases of Agile Model:

Following are the phases in the Agile model:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

1. Requirements gathering: Here is where you define the project's requirements. This phase includes explaining business opportunities and planning the time and effort required for the project. Once you quantify this information, you can evaluate the technical and economic feasibility of your project

2. Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the

high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. Deployment: In this phase, the team issues a product for the user's work environment.

6. Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Popular Agile Testing Methods:

- Scrum
- Kanban
- Crystal
- Dynamic Software Development Method(DSDM)
- Lean Software Development
- eXtreme Programming(XP)

When to use the Agile Model?

- When frequent changes are required.
- When a highly qualified and experienced team is available.
- When a customer is ready to have a meeting with a software team all the time.
- When project size is small.

Advantages(pros) **of Agile Method:**

1. Frequent Delivery
2. Face-to-Face Communication with clients.
3. Efficient design and fulfils the business requirement.
4. Anytime changes are acceptable.
5. It reduces total development time.

Disadvantages(Cons) of Agile Model:

While the Agile methodology has numerous advantages, there are also some disadvantages that should be considered:

1. **Limited documentation:** Agile methodologies place less emphasis on documentation than traditional methods, which can lead to a lack of documentation and transparency in the development process.
2. **Time and resource constraints:** Agile projects can be challenging to manage because of their time-boxed sprints and frequent iterations. This can put a strain on resources, particularly if the team is working on multiple projects concurrently.
3. **Lack of predictability:** Agile development is iterative and adaptive, which means that the end product is not always predictable. This can be problematic for clients who want a clear understanding of what they will receive at the end of the project.
4. **Requires experienced team:** Agile methodologies rely heavily on the expertise and collaboration of the development team. This means that inexperienced or unskilled team members may struggle to keep up with the fast-paced development cycles.
5. **Dependencies on customer availability:** Agile methodologies depend on the customer's active involvement and feedback, which can be challenging if the customer is not available or responsive.
6. **Costly for small projects:** Agile methodologies can be more expensive for small projects because of the overhead involved in setting up the development process and infrastructure.
7. **Constant Changes:** Because of the iterative nature of Agile development, changes are frequent, which can be challenging for stakeholders who may be uncomfortable with change or have difficulty keeping up with the pace of development.

Overall, while Agile has many benefits, it may not be suitable for every project or organization. It's essential to weigh the pros and cons of Agile before adopting it as a development methodology.

S. no.	Purpose	Agile model	Waterfall model
1.	**Definition**	Agile model follows the incremental approach, where each incremental part is developed through iteration after every timebox.	Waterfall model follows a sequential design process.
2.	**Progress**	In the agile model, the measurement of progress is in terms of developed and delivered functionalities.	In the waterfall model, generally the measurement of success is in terms of completed and reviewed artifacts.
3.	**Nature**	Agile model is flexible as there is a possibility of changing the requirements even after starting the development process.	On the other hand, the waterfall model is rigid as it does not allow to modify the requirements once the development process starts.
4.	**Customer interaction**	In Agile model, there is a high customer interaction. It is because, after every iteration, an incremental version is deployed to the customer.	Customer interaction in waterfall model is very less. It is because, in a waterfall model, the product is delivered to the customer after overall development.
5.	**Team size**	It has a small team size. As smaller is the team, the fewer people work on it so that they can move faster.	In the waterfall model, the team may consist more members.
6.	**Suitability**	Agile model is not a suitable model for small projects. The expenses of developing the small projects using agile is more than compared to other models.	Waterfall model works well in smaller size projects where requirements are easily understandable. But waterfall model is not suitable for developing the large projects.
7.	**Test plan**	The test plan is reviewed after each sprint.	Test plan is reviewed after complete development.
8.	**Testing**	Testing team can take part in the requirements change phase without problems.	It is difficult for the testing team to initiate any change in needs.

INTRODUCTION AND BENEFITS OF WORKING IN A DEVOPS ENVIRONMENT

DevOps and Agile are both different models

Similarities

- Both are software development methodologies
- Both models concentrating on rapid software development with team collaboration.

Differences

- The difference will come once the development of the project is completed. The agile model talks about only development but not operations. The DevOps model is concerned with the entire product life cycle, including development and operations.

- In the Agile model, separate people are responsible for development, testing, deployment, etc. But in Devops, the Devops engineer is responsible for everything from development to operations and operations to development.

What is DevOps

- For any software product, two types of engineers will work.

1. Development group
2. Operations /Non development/administrators groups

Again this classification is divided into small set of groups

Development group

The people who are involving in

planning

coding

Build

Testing

are considered as development team

To develop a project persons required in development are:

Business analyst (BA)

System analyst (SA)

Design architect(DA)

Developers/coders

Build Engineer

Test Engineer/QA

- Once the code has been developed by the developers, it must be integrated and released as executable code.

Operations group

- A project has been developed and is ready to be moved to the client machine (assuming that the development team's work has been completed), so configuration is required to move the software to the client machine.

The people who are involving in

1. Release engineer
2. Deployment engineer
3. Operations
4. monitoring are considered as operations group

Ex: Release engineer

Deployment engineer

system admin

Database admin

Network admin etc

Operations Team responsibilities

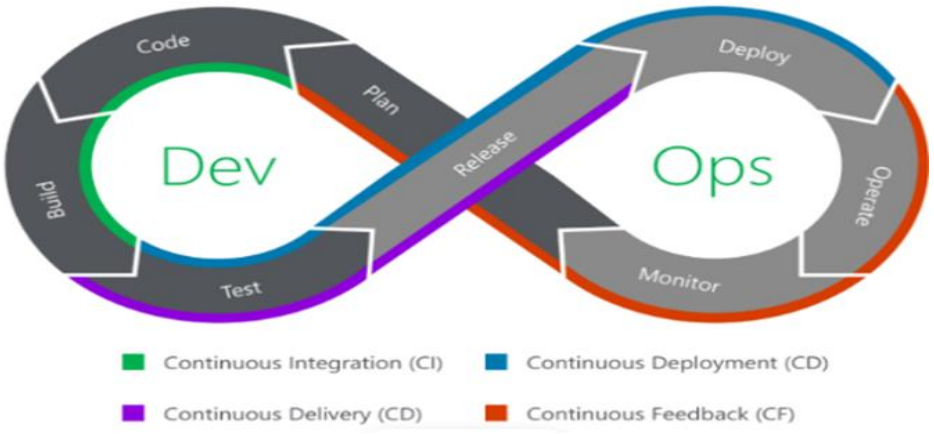
- Installation of server hardware and OS
- Configuration of servers, networks, storage, etc…
- Monitoring of servers
- IT security

- Backup and disaster recovery planning

The term DevOps was first introduced by Patrick Debois in 2009. The term DevOps combines the words development and operations.

Difinition: DevOps is a methodology that promotes collaboration between Development and Operations Team. This allows deploying code to production faster and in an automated way. It helps to enables rapid development of products.

DEVOPS DELIVERY PIPELINE



The Eight Phases of a DevOps Pipeline

Plan:
In plan stage business owners and software development team discuss project goals and create a plan.

Code:
In this stage, programmers design and code the application. They use tools like GIT to store application code.

Build:
Maven and Gradie are examples of build tools. In this stage, developers take the code from various repositories and combine it to create the entire application.

Test:
Application is tested using automation testing tools like selenium and Junit to ensure software quality

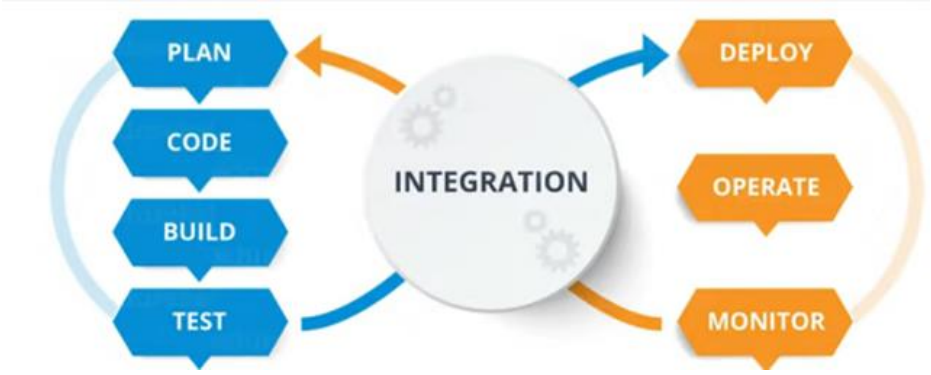
Release:

When testing is complete new features are integrated automatically to the already existing code bash

Deploy:
Application is packaged after release and deployed from development server to production server.

Operate:
Once software is deployed operations Team activities such as configuring servers and provisioning them with the required resources.

Monitor:
Monitoring allows IT organizations to identify specific issues of specific releases and understand the impact on end users



USE CASE FOR DevOps

Benefits of working in a DevOps environment

Working in a DevOps environment has numerous benefits, including:

1. **Improved collaboration:** DevOps promotes collaboration between teams, including development, operations, and quality assurance, resulting in better communication and a shared responsibility for the success of the project.
2. **Faster time to market:** DevOps promotes the use of automation and continuous delivery practices, enabling teams to release software faster and more frequently, reducing the time to market.
3. **Increased efficiency**: By automating repetitive tasks and using infrastructure as code, DevOps teams can reduce the time and effort required for deployment, testing, and maintenance, resulting in increased efficiency.

4. **Better quality:** DevOps emphasizes testing and quality assurance throughout the development process, resulting in higher-quality software and fewer defects.
5. **Improved customer satisfaction:** DevOps enables teams to deliver software that meets customer needs and expectations, resulting in improved customer satisfaction and retention.
6. **Greater agility:** DevOps promotes the use of agile development methodologies, enabling teams to respond quickly to changing customer needs and market conditions.
7. **Cost savings:** By reducing the time and effort required for deployment and maintenance, DevOps can result in cost savings for organizations.

Overall, DevOps can provide significant benefits for organizations looking to improve their software development processes and deliver high-quality software faster and more efficiently.

DevOps and ITIL

This section explains how DevOps and other ways of working coexist and fit together as a larger whole.

ITIL(Information Technology Infrastructure Library) is a set of best practices for IT service management. It focuses on the processes and procedures for delivering and maintaining IT services in a reliable and efficient manner. DevOps, on the other hand, focuses on collaboration and automation between development and operations teams to deliver software quickly and reliably.

ITIL is a framework that formalizes different aspects of the software life cycle. DevOps and CD emphasize delivering small changes to production often, while ITIL may seem to hold the opposite view. However, ITIL can be useful for managing complex changes in large monolithic legacy systems.

Large organizations often deal with monolithic legacy systems, which require complex changes and are managed by processes such as ITIL. However, many practices in ITIL can also be translated into corresponding DevOps practices. Both ITIL and DevOps emphasize the use of configuration management systems and databases.

Following are the various popular services of IT which are covered by the Information Technology Infrastructure Library **(ITIL)**:

- Cloud Services
- Data processing and storage
- IT Consulting
- Help desk support
- IoT
- Network security

Why ITIL?

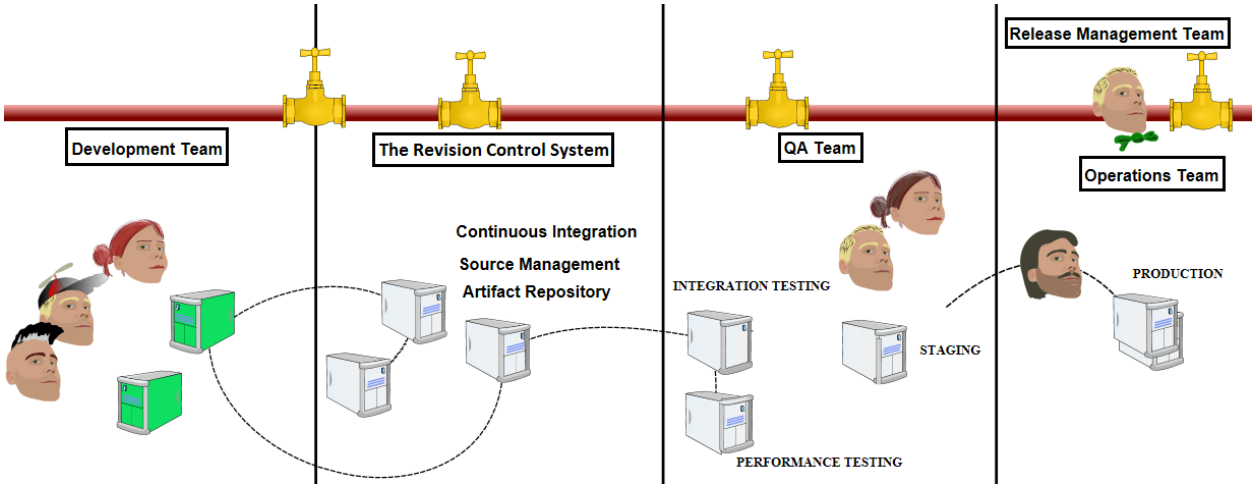
Following are the various reasons which allows the IT and Business managers to use this method:

- It is used to enhance the resource and capabilities.
- It is used for planning the processes with particular goals in mind.
- It helps in integrating business and service strategies.
- It also helps to control the IT budget and investment.
- This framework also helps in changing the organizational culture.
- It also helps in developing the IT and business partnerships.
- It also offers the maximum values to the customers.
- It also improves the relationship with the costumers by delivering the services.
- It also offers the good management of services.
- It also helps in measuring, optimizing, and monitoring the performance of the service provider.
- It allows them to define the roles for each task.
- It manages the service failure.
- It helps managers by providing the service which is reliable and useful.

DevOps process and Continuous Delivery:

Continuous delivery (CD) is a software engineering practice in which teams develop, build, test, and release software in short cycles. It depends on automation at every stage so that cycles can be both quick and reliable.

An example of a CD pipeline in a large organization is introduced in the following diagram:

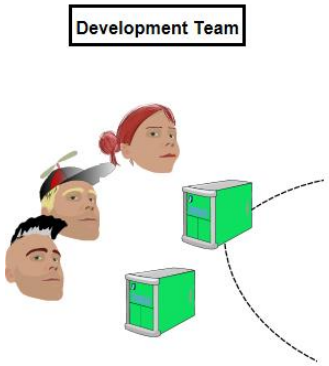


CD pipeline varies depending on size, complexity, testing environments, and production environments.

The developers

Developers work on their personal computers or workstations where they write and develop code using various tools to increase their productivity.

The following diagram illustrates the development team:



Developers ideally should have production-like environments on their own machines to work with. However, it may not always be possible, especially when dealing with external dependencies like payment systems or phone hardware. In such cases, simulating or mocking these dependencies is a common approach.

If you have a developer background in DevOps, you may focus on prepackaged developer environments to save time setting up development environments. This could include specific versions of tools like Java Development Kit and Eclipse for Java, or Python and its dependencies using tools like Virtualenv or Anaconda.

When working with DevOps, we need to maintain multiple environments separately. The developer environment includes all the necessary development

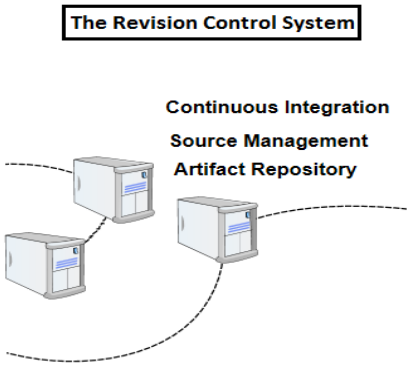
tools and is different from the test and production systems. Additionally, developers need a way to deploy their code in a production-like way. This can be achieved through various methods such as a virtual machine with Vagrant, a cloud instance on AWS, or a Docker container.

In DevOps, it's preferable to use a development environment that's similar to the production environment. For example, if production servers run Red Hat Linux, the development machine could use CentOS Linux or Fedora Linux. This approach saves time and minimizes hassle since you can use much of the same software locally as in production. Additionally, using CentOS or Fedora can be motivated by the license costs of Red Hat, and enterprise distributions typically lag behind with software versions. Similarly, if production runs on Windows servers, it's more convenient to use a Windows development machine.

The Revision Control System

In DevOps, the Revision Control System (RCS) is essential for storing an organization's software products, configurations, and hardware designs. Git is a commonly used RCS. Choosing an RCS is just one part of the larger picture, as considerations such as directory structure conventions and branching strategy also need to be made. If an organization has many independent components, separate repositories may be used for each.

The following diagram shows the systems dealing with code, CI, and artifact storage in the CD pipeline in greater detail:



The build server

The build server is simple and builds source code at regular intervals or triggers. It listens to changes in the RCS and updates its local copy, then builds the source and runs tests to ensure quality (Continuous Integration). Jenkins is a widely used open source solution for build servers that is easy to install and provides a simple, robust experience.

The Artifact repository:

After code is verified and compiled, it's useful to store the binary artifacts in a separate repository accessible over HTTP. These repositories are file systems with features for searching, indexing, and storing metadata. One common choice for Java artifacts is Sonatype Nexus, which can also store OS-specific artifacts like RPMs and JavaScript artifacts. Amazon S3 is another option for storing artifacts, and some build systems like Atlassian Bamboo can use it. Package managers can also act as artifact repositories, and there are open source implementations like Ceph that provide S3-compatible object storage.

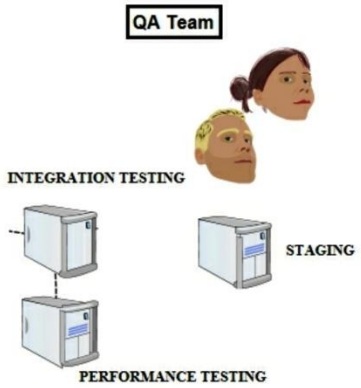
Package managers:

Linux servers use package managers to deploy software packages, with Red Hat-like systems using RPM and Debian-like systems using .deb format. Package managers such as yum/dnf for Red Hat and aptitude/dpkg for Debian-like systems make it easy to install and upgrade packages with automatic dependency installation. Manually logging into each server and typing upgrade commands can be feasible but there are more advanced deployment systems available.

Test Environments:

After the build server stores artefacts in the binary repository, they can be installed in test environments. These environments should be similar to production servers in terms of installation and configuration.

The following diagram shows the test systems in greater detail



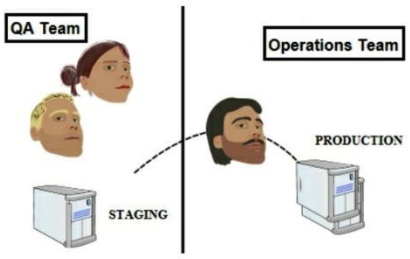
Staging/production:

Staging environments are a final type of test environment that is interchangeable with production environments. They allow for new releases to

be installed, checked, and then swapped with old production servers to become the new production servers. This deployment strategy is called blue-green deployment.

The exact process depends on the product being deployed and the number of production systems available. Sometimes, it is not possible to have several production systems running in parallel, while other times there may be hundreds of production systems in a pool. In the latter case, new releases can be gradually rolled out to new users while logged-in users stay with the older version. While not all organizations have the resources for production-quality staging servers, it is a safe and effective way to handle upgrades when possible.

The following diagram from the larger CD diagram shows the final systems and the roles involved:



Release management:

So far, we have assumed that the release process is mostly automatic. This is the dream scenario for people working with DevOps.

This dream scenario is a challenge to achieve in the real world. One reason for this is that it is usually hard to reach the level of test automation needed in order to have complete confidence in automated deploys. Another reason is simply that the cadence of business development doesn't always match the cadence of technical development. Therefore, it is necessary to enable human intervention in the release process.

A faucet is used in the following diagram to symbolize human interaction—in this case, by a dedicated release manager:

How this is done in practice varies, but deployment systems usually have a way to support how to describe which software versions to use in different environments.

The integration test environments can then be set to use the latest versions that have been deployed to the binary artifact repository. The staging and production servers have particular versions that have been tested by the quality assurance team.

Scrum, Kanban, and the delivery pipeline

How does the CD pipeline that we described in this chapter support Agile processes such as Scrum and Kanban?

Scrum focuses on Sprint cycles, which can occur biweekly or monthly. Kanban can be said to focus more on shorter cycles, which can occur daily.

The philosophical differences between Scrum and Kanban are a bit deeper, although not mutually exclusive. Many organizations use both Kanban and Scrum together.

From a software-deployment viewpoint, both Scrum and Kanban are similar. Both require frequent, hassle-free deployments. From a DevOps perspective, a change starts propagating through the CD pipeline towards test systems and beyond when they are deemed ready enough to start that journey. This may be judged on subjective measurements or objective ones, such as all unit tests are green.

Our pipeline can manage the following types of scenarios:

- The build server supports the generation of 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** (**SAFe®**), Kanban, or another method within the lean or Agile frameworks. Even a traditional waterfall process can be successfully managed—DevOps serves all!

Wrapping up – a complete example

So far, we have covered a lot of information at a cursory level.

To make it clearer, let's have a look at what happens to a concrete change as it propagates through the systems, using an example:

- The development team has been given the responsibility of developing a change to the organization's system. The change revolves around adding new roles to the authentication system. This seemingly simple task is hard in reality because many different systems will be affected by the change.
- To make life easier, it is decided that the change will be broken down into several smaller changes, which will be tested independently and mostly automatically by automated regression tests.
- The first change, the addition of a new role to the authentication system, is developed locally on developer machines and given best-effort local testing. To really know whether it works, the developer needs access to systems not available in their local environment—in this case, a **Lightweight Directory Access Protocol** (**LDAP**) server containing user information and roles.
- If test-driven development is used, a failing test is written before any actual code is written. After the failing test is written, new code that makes the test pass is written.
- The developer checks the change to the organization's revision control system, a Git repository.
- The build server picks up the change and initiates the build process. After unit testing, the change is deemed fit enough to be deployed to the binary repository, which is a Nexus installation.
- The configuration management system, Puppet, notices that there is a new version of the authentication component available. The integration test server is described as requiring the latest version to be installed, so Puppet goes ahead and installs the new component.
- The installation of the new component now triggers automated regression tests. When these have been finished successfully, manual tests by the quality assurance team commence.
- The quality assurance team gives the change its seal of approval. The change moves on to the staging server, where final acceptance testing commences.
- After the acceptance test phase is completed, the staging server is swapped into production, and the production server becomes the new staging server. This last step is managed by the organization's load-balancing server.

The process is then repeated as necessary. As you can see, there is a lot going on!

Identifying bottlenecks

As is apparent from the previous example, 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. It is important to continuously improve your processes, which is often referred to as *Kaizen*, which is a Japanese term. You may discover new technical improvements as times goes on, or simplifications of your processes—"*Everything should be made as simple as possible, but no simpler*," a saying which is often attributed to Albert Einstein.

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 and cause hassle. Here are some possible causes:

- The database schema changes.
- The 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 may be because your architecture is basically a monolith.

Scrum:

Scrum is a framework used in agile software development to manage and organize work. It is one of the most popular and widely used agile frameworks.

Scrum involves the following key roles:

1. **Product Owner**: The person responsible for defining and prioritizing the product backlog, and ensuring that the team is working on the most important things.
2. **Scrum Master:** The person responsible for facilitating the Scrum process, removing obstacles that the team faces, and ensuring that the team follows the Scrum framework.
3. **Development Team:** The group of people responsible for delivering a potentially releasable product increment at the end of each Sprint.

Scrum also involves the following ceremonies:

1. **Sprint Planning:** The team comes together to plan the work they will do during the upcoming Sprint.

2. **Daily Scrum:** Daily meetings where team members share progress, discuss any obstacles, and plan their work for the next 24 hours.
3. **Sprint Review:** At the end of each Sprint, the team demonstrates the work they have completed and receives feedback from stakeholders.
4. **Sprint Retrospective:** The team comes together to reflect on the Sprint, discuss what went well, what could be improved, and make a plan for implementing those improvements in the next Sprint.

   The Scrum framework emphasizes iterative development, self-organizing teams, and constant communication between team members. It is designed to be flexible and adaptable to changing requirements and priorities.

Kanban

Kanban is a methodology used in lean manufacturing and agile software development to manage and improve the flow of work. It is focused on visualizing work, limiting work in progress, and optimizing the flow of work through a system.

Kanban involves the following key principles:

1. **Visualize the workflow:** Create a visual representation of the work that needs to be done, typically in the form of a board with columns that represent different stages of the workflow.

2. **Limit work in progress:** Set a limit on the amount of work that can be in progress at any given time to prevent overloading the system and improve efficiency.

3. **Manage flow:** Continuously monitor and improve the flow of work through the system, ensuring that work is being completed in a timely and efficient manner.

4. **Make process policies explicit:** Clearly define and communicate the rules and policies that govern the work being done.

5. **Implement feedback loops:** Use feedback loops to identify and address issues and opportunities for improvement.

Kanban is designed to be flexible and adaptable to a wide range of situations. It can be used in conjunction with other methodologies, such as Scrum, and is particularly well-suited to environments where work items have varying sizes and priorities.

Prepared by:
Dr.N.Chandramouli
Vaageswari College of engineering