DevOps

Syllabus Copy

DEVOPS

B.Tech. III Year II Sem. **L T P C**

3104

Course Objectives: The main objectives of this course are to: 1. Describe the agile relationship between development and IT operations. 2. Understand the skill sets and high-functioning teams involved in DevOps and related methods to reach a continuous delivery capability. 3. Implement automated system update and DevOps lifecycle.

Course Outcomes: On successful completion of this course, students will be able to: 1. Identify components of Devops environment. 2. Describe Software development models and architectures of DevOps. 3. Apply different project management, integration, testing and code deployment tool. 4. Investigate different DevOps Software development models. 5. Assess various Devops practices. 6. Collaborate and adopt Devops in real-time projects.

UNIT - I

Introduction:

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

UNIT - II

Software development models and DevOps:

DevOps Lifecycle for Business Agility, DevOps, and Continuous Testing. DevOps influence on Architecture: Introducing software architecture, The monolithic scenario, Architecture rules of thumb, The separation of concerns, Handling database migrations, Microservices, and the data tier, DevOps, architecture, and resilience.

UNIT – III

Introduction to project management: The need for source code control, The history of source code management, Roles and code, source code management system and migrations, Shared authentication, Hosted Git servers, Different Git server implementations, Docker intermission, Gerrit, The pull request model, GitLab.

UNIT - IV

Integrating the system:

Build systems, Jenkins build server, Managing build dependencies, Jenkins plugins, and file system layout, The host server, Build slaves, Software on the host, Triggers, Job chaining and build pipelines, Build servers and infrastructure as code, Building by dependency order, Build phases, Alternative build servers, Collating quality measures.

UNIT - V

Testing Tools and automation: Various types of testing, Automation of testing Pros and cons, Selenium - Introduction, Selenium features, JavaScript testing, Testing backend integration points, Test-driven development, REPL-driven development Deployment of the system: Deployment systems, Virtualization stacks, code execution at the client, Puppet master and agents, Ansible, Deployment tools: Chef, Salt Stack and Docker

TEXT BOOKS:

Joakim Verona. Practical Devops, Second Edition. Ingram short title; 2nd edition (2018). ISBN10: 1788392574 2. Deepak Gaikwad, Viral Thakkar. DevOps Tools from Practitioner's Viewpoint. Wiley publications. ISBN: 9788126579952

REFERENCE BOOK:

1. Len Bass, Ingo Weber, Liming Zhu. DevOps: A Software Architect's Perspective. Addison Wesley; ISBN-10.

DevOps

Lecture notes

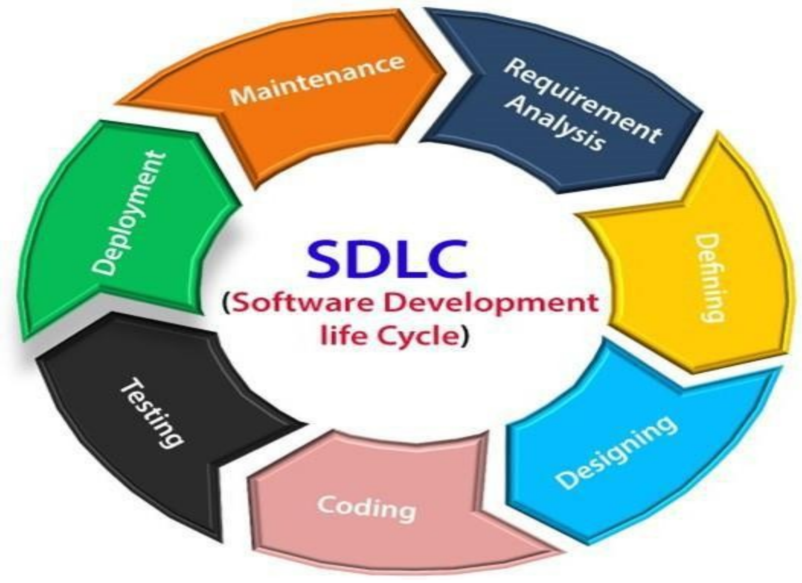
Unit 1

Introduction

History

Software Development Life Cycle (SDLC)

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods 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.



Stage1: 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.

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.

Stage2: 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.

Stage3: Designing the Software

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.

Stage4: 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.

Stage5: 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.

Stage6: 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.

Stage7: 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.This model has five phases: Requirements analysis and specification, design, implementation, and unit testing, integration and system testing, and operation and maintenance. The steps always follow in this order and do not overlap. The developer must complete every phase before the next phase begins. This model is named "**Waterfall Model**", because its diagrammatic representation resembles a cascade of waterfalls.

1. Requirements analysis and specification phase: The aim of this phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. It describes the "what" of the system to be produced and not "how."In this phase, a large document called **Software Requirement Specification (SRS)** document is created which contained a detailed description of what the system will do in the common language.

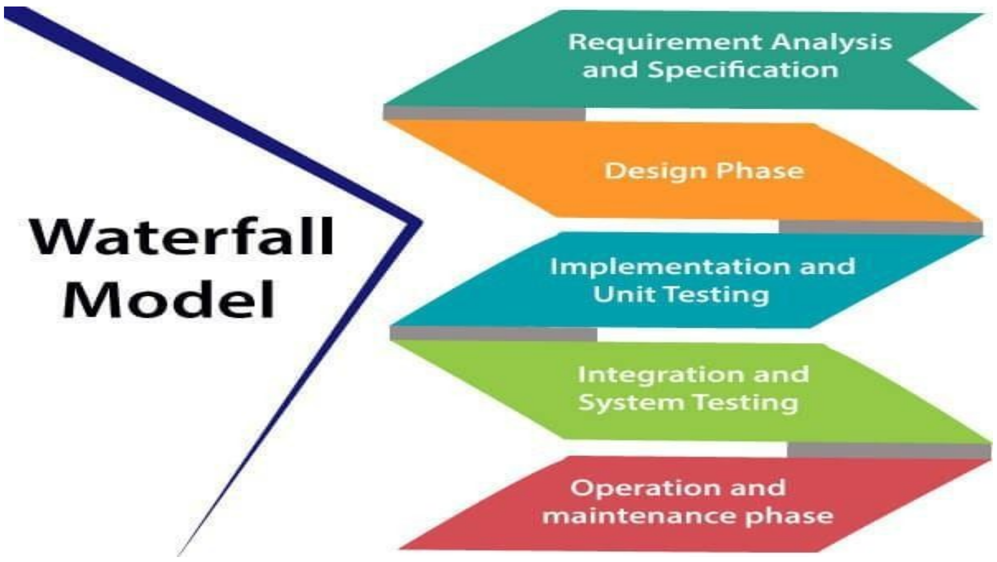
2. Design Phase: This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).

3. Implementation and unit testing: During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.

During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

4. Integration and System Testing: This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs, and accurate results. Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.

5. Operation and maintenance phase: Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.


Waterfall Model
Requirement Analysis and Specification
Design Phase
Implementation and Unit Testing
Integration and System Testing
Operation and maintenance phase

Advantages of Waterfall model

- This model is simple to implement also the number of resources that are required for it is minimal.
- The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
- The start and end points for each phase is fixed, which makes it easy to cover progress.
- The release date for the complete product, as well as its final cost, can be determined before development.

- It gives easy to control and clarity for the customer due to a strict reporting system.

Disadvantages of Waterfall model

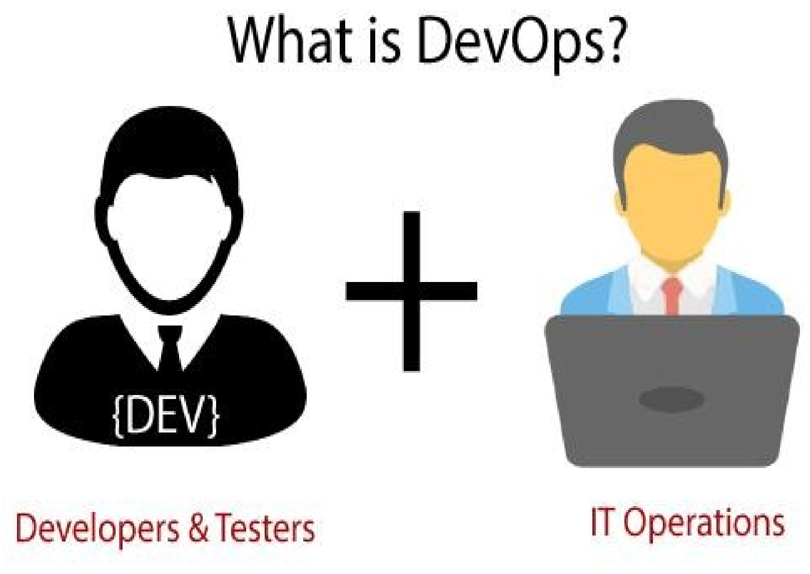
- In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
- This model cannot accept the changes in requirements during development.
- It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.
- Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

Introduction

The DevOps is the combination of two words, one is **Development** and other is **Operations**. It is a culture to promote the development and operation process collectively.

The DevOps tutorial will help you to learn DevOps basics and provide depth knowledge of various DevOps tools such as **Git, Ansible, Docker, Puppet, Jenkins, Chef, Nagios**, and **Kubernetes**.

What is DevOps?



DevOps

Why DevOps?

Before going further, we need to understand why we need the DevOps over the other methods.

- The operation and development team worked in complete isolation.
- After the design-build, the testing and deployment are performed respectively. That's why they consumed more time than actual build cycles.
- Without the use of DevOps, the team members are spending a large amount of time on designing, testing, and deploying instead of building the project.
- Manual code deployment leads to human errors in production.
- Coding and operation teams have their separate timelines and are not in synch, causing further delays.
- **DevOps History**

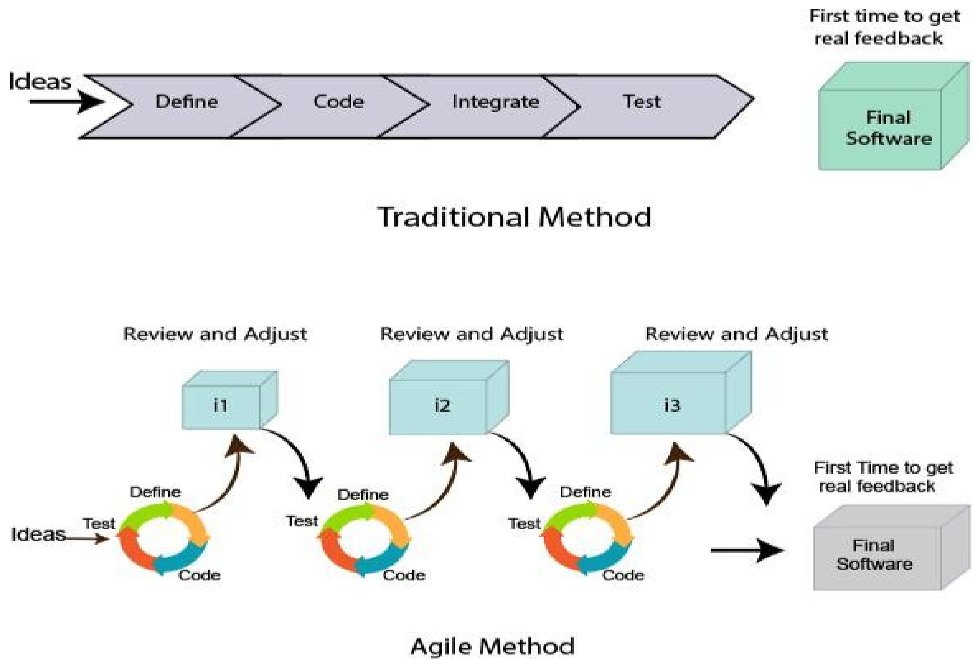
- In 2009, the first conference named **DevOpsdays** was held in Ghent Belgium. Belgian consultant and Patrick Debois founded the conference.
- In 2012, the state of DevOps report was launched and conceived by Alanna Brown at Puppet.
- In 2014, the annual State of DevOps report was published by Nicole Forsgren, Jez Humble, Gene Kim, and others. They found DevOps adoption was accelerating in 2014 also.
- In 2015, Nicole Forsgren, Gene Kim, and Jez Humble founded DORA (DevOps Research and Assignment).
- In 2017, Nicole Forsgren, Gene Kim, and Jez Humble published "Accelerate: Building and Scaling High Performing Technology Organizations".

Agile development

An agile methodology is an iterative approach to software development. Each iteration of agile methodology takes a short time interval of 1 to 4 weeks. The agile development process is aligned to deliver the changing business requirement. It distributes the software with faster and fewer changes.

The single-phase software development takes 6 to 18 months. In single-phase development, all the requirement gathering and risks management factors are predicted initially.

The agile software development process frequently takes the feedback of workable product. The workable product is delivered within 1 to 4 weeks of iteration.



ITIL

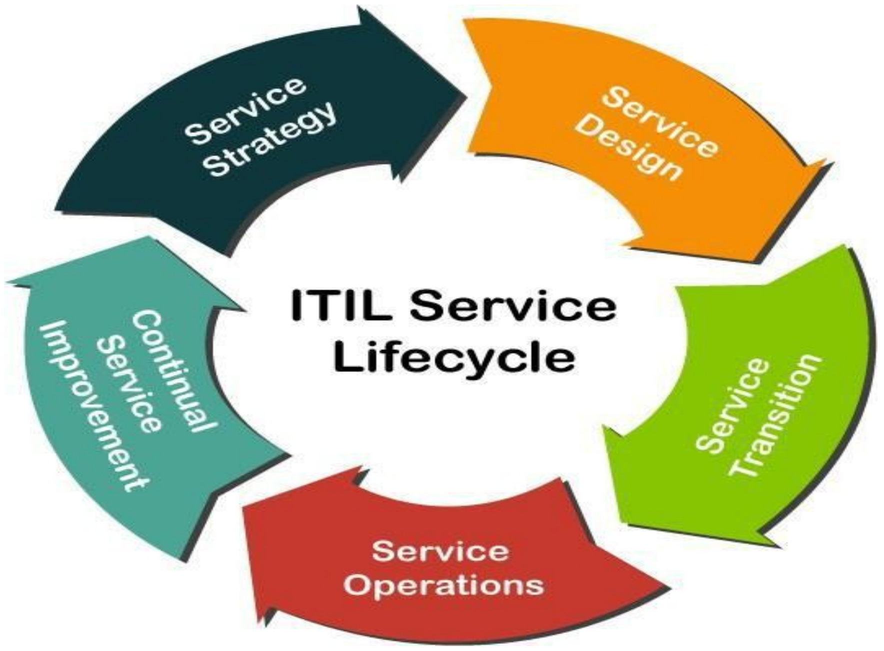
ITIL is an abbreviation of **Information Technology Infrastructure Library**.

It is a framework which helps the IT professionals for delivering the best services of IT. This framework is a set of best practices to create and improve the process of ITSM (IT Service Management). It provides a framework within an organization, which helps in planning, measuring, and implementing the services of IT.

The main motive of this framework is that the resources are used in such a way so that the customer get the better services and business get the profit.

It is not a standard but a collection of best practices guidelines.

Service Lifecycle in ITIL



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

Service Strategy

Service Strategy is the first and initial stage in the lifecycle of the ITIL framework. The main aim of this stage is that it offers a strategy on the basis of the current market scenario and business perspective for the services of IT.

This stage mainly defines the plans, position, patters, and perspective which are required for a service provider. It establishes the principles and policies which guide the whole lifecycle of IT service.

Following are the various essential services or processes which comes under the **Service Strategy** stage:

- Financial Management
- Demand Management
- Service Portfolio Management
- Business Relationship Management
- Strategy Management

Strategy Management:

The aim of this management process is to define the offerings, rivals, and capabilities of a service provider to develop a strategy to serve customers.

According to the version 3 (V3) of ITIL, this process includes the following activities for IT services:

1. Identification of Opportunities
2. Identification of Constraints
3. Organizational Positioning
4. Planning
5. Execution

Following are the three sub-processes which comes under this management process:

1. Strategic Service Assessment
2. Service Strategy Definition
3. Service Strategy Execution

Financial Management:

This process helps in determining and controlling all the costs which are associated with the services of an IT organization. It also contains the following three basic activities:

1. Accounting 2. charging 3. Budgeting

Following are the four sub-processes which comes under this management process:

1. Financial Management Support
2. Financial Planning
3. Financial Analysis and Reporting
4. Service Invoicing

Demand Management

This management process is critical and most important in this stage. It helps the service providers to understand and predict the customer demand for the IT services. Demand management is a process which also work with the process of Capacity Management. Following are basic objectives of this process:

- This process balances the resources demand and supply. o
It also manages or maintains the quality of service.

According to the version 3 (V3) of ITIL, this process performs the following 3 activities:

1. Analysing current Usage of IT services
2. Anticipate the Future Demands for the Services of IT.
3. Influencing Consumption by Technical or Financial Means

Following are the two sub-processes which comes under this management process:

1. Demand Prognosis
2. Demand Control.

Business Relationship Management

This management process is responsible for maintaining a positive and good relationship between the service provider and their customers. It also identifies the needs of a customer. And, then ensure that the services are implemented by the service provider to meet those requirements.

This process has been released as a new process in the ITIL 2011.

According to the version 3 (V3) of ITIL, this process performs the following various activities:

- This process is used to represent the service provider to the customer in a positive
manner. o This process identifies the business needs of a customer.
- It also acts as a mediator if there is any case of conflicting requirements from the different businesses.

Following are the six sub-processes which comes under this management process:

1. Maintain Customer Relationships
2. Identify Service Requirements
3. Sign up Customers to standard Services

4. Customer Satisfaction Survey
5. Handle Customer Complaints
6. Monitor Customer Complaints.

Service Portfolio Management

This management process defines the set of customer-oriented services which are provided by a service provider to meet the customer requirements. The primary goal of this process is to maintain the service portfolio.

Following are the three types of services under this management process:

1. Live Services 2. Retired Services 3. Service Pipeline.

Following are the three sub-processes which comes under this management process:

1. Define and Analyse the new services or changed services of IT.
2. Approve the changes or new IT services
3. Service Portfolio review.

Service Design

It is the second phase or a stage in the lifecycle of a service in the framework of ITIL. This stage provides the blueprint for the IT services. The main goal of this stage is to design the new IT services. We can also change the existing services in this stage.

Following are the various essential services or processes which comes under the Service Design stage:

o Service Level Management
o Capacity Management
o Availability Management
o Risk Management
o Service Continuity Management o
Service Catalogue Management o
Information Security Management o
Supplier Management
o Compliance Management
o Architecture Management

Service Level Management

In this process, the **Service Level Manager** is the process owner. This management is fully redesigned in the ITIL 2011.

Service Level Management deals with the following two different types of agreements:

1. Operational Level Agreement
2. Service Level Agreement

According to the version 3 (V3) of ITIL, this process performs the following activities:

- It manages and reviews all the IT services to match service Level Agreements.
- It determines, negotiates, and agrees on the requirements for the new or changed IT services.

Following are the four sub-processes which comes under this management process:

1. Maintenance of SLM framework
2. Identifying the requirements of services
3. Agreements sign-off and activation of the IT services
4. Service level Monitoring and Reporting.

Capacity Management

This management process is accountable for ensuring that the capacity of the IT service can meet the agreed capacity in a cost-effective and timely manner. This management process is also working with other processes of ITIL for accessing the current infrastructure of IT.

According to the version 3 (V3) of ITIL, this process performs the following activities:

- It manages the performance of the resources so that the IT services can easily meet their SLA targets.
- It creates and maintains the capacity plan which aligns with the strategic plan of an organization. o It reviews the performance of a service and the capacity of current service periodically.
- It understands the current and future demands of customer for the resources of IT.

Following are the four sub-processes which comes under this management process:

1. Business Capacity Management
2. Service Capacity Management

3. Component Capacity Management
4. Capacity Management Reporting

Availability Management

In this process, the **Availability Manager** is the owner. This management process has a responsibility to ensure that the services of IT meet the agreed availability goals. This process also confirms that the services which are new or changed does not affect the existing services.

It is used for defining, planning, and analysing all the availability aspects of the services of IT.

According to the version 3 (V3) of ITIL, this process contains the following two activities:

1. Reactive Activity
2. Proactive Activity

Following are the four sub-processes which comes under this management process:

1. Design the IT services for availability
2. Availability Testing
3. Availability Monitoring and Reporting

Risk Management

In this process, the **Risk Manager** is the owner. This management process allows the risk manager to check, assess, and control the business risks. If any risk is identified in the process of business, the risk of that entry is created in the ITIL Risk Register.

According to the version 3 (V3) of ITIL, this process performs the following activities in the given order:

o It identifies the threats.

o It finds the probability and impact of
risk. o It checks the way for reducing those
risks. o It always monitors the risk factors.

Following are the four sub-processes which comes under the Risk process:

1. Risk Management Support
2. Impact on business and Risk analysis
3. Monitoring the Risks.

4. Assessment of Required Risk Mitigation

Service Catalogue Management (SCM)

In this process, the **Service Catalogue Manager** is the owner. This management process allows the Catalogue Manager to give the huge information about all the other management processes.

It contains the services in the service operation phase which are presently active.

It is a process which certifies that the service catalogue is maintained, produced, and contains all the accurate information for all the operational IT services.

Following are the two types or aspects of service catalogue in ITIL framework:

1. BSC or Business Service Catalogue
2. TSC or Technical Service Catalogue

Under this management process, no sub-process is specified or defined.

Service Continuity Management

In this process, the **IT Service Continuity Manager** is specified as the owner. It allows the continuity manager to maintain the risks which could impact on the service of IT.

This process is bound with other processes of ITIL such as capacity and availability management to access and plan the resources which are needed to manage the desired service level.

The ITSCM consists of the following four activities or stages:

1. Initiation
2. Requirements and Strategy
3. Implementation
4. Ongoing Operation

Information Security Management

In this process, the **Information Security Manager** is specified as the owner. The main aim of this management process is to verify the confidentiality, integrity, and availability of the data, information, and services of an IT organization.

The main objective of this process is to control the access of information in the organizations.

According to the version 3 (V3) of ITIL, this process performs the following four activities:

1. Plan 2. Implement 3. Evaluation 4. Maintain

According to the version 3 (V3) of ITIL, following are the four sub-processes which comes under this management process:

1. Design of Security controls
2. Validation and Testing of Security
3. Management of Security Incidents
4. Security Review

Supplier Management

In this process, the **Supplier Manager** plays a role as an owner. The supplier manager is responsible to verify that all the suppliers meet their contractual commitments.

It also works with the Financial and knowledge management, which helps in selecting the suppliers on the basis of previous knowledge.

Following are the various activities which are involved in this process:

- o It manages the sub-contracted suppliers.
- o It manages the relationship with the suppliers.
- o It helps in implementing the supplier policy.
- o It also manages the supplier policy and supports the SCMIS. o
It also manages or maintains the performance of the suppliers.

According to the version 3 (V3) of ITIL, following are the six sub-processes which comes under this management process:

1. Provide the Framework of Supplier Management
2. Evaluation and selection of new contracts and suppliers
3. Establish the new contracts and suppliers
4. Process the standard orders
5. Contract and Supplier Review
6. Contract Renewal or Termination.

Compliance Management

In this process, the **Compliance Manager** plays a role as an owner. This management process allows the compliance manager to check and address all the issues which are associated with regulatory and non-regulatory compliances.

Under this compliance management process, no sub-process is specified or defined.

Here, the role of Compliance Manager is to certify that the guidelines, legal requirements, and standards are being followed properly or not. This manager works in parallel with the following three managers:

1. Information Security Manager
2. Financial Manager
3. Service Design Manager.

Architecture Management

In this process, the **Enterprise Architect** plays a role as an owner. The main aim of Enterprise Architect is to maintain and manage the architecture of the Enterprise.

This management process helps the **Enterprise Architect** by verifying that all the deployed services and products operate according to the specified architecture baseline in the Enterprise.

This process also defines and manages a baseline for the future technological development.

Under this Architecture management process, no sub-process is specified or defined.

Service Transition

Service Transition is the third stage in the lifecycle of ITIL Management Framework.

The main goal of this stage is to build, test, and develop the new or modified services of IT. This stage of service lifecycle manages the risks to the existing services. It also certifies that the value of a business is obtained.

This stage also makes sure that the new and changed IT services meet the expectation of the business as defined in the previous two stages of service strategy and service design in the lifecycle.

It can also easily manage or maintains the transition of new or modified IT services from the **Service Design** stage to **Service Operation** stage.

There are following various essential services or processes which comes under the Service Transition stage:

- o Change Management
- o Release and Deployment Management
- o Service Asset and Configuration Management o Knowledge Management
- o Project Management (Transition Planning and Support)
- o Service Validation and Testing
- o Change Evaluation

Change Management

In this process, the **Change Manager** plays a role as an owner. The Change Manager controls or manages the service lifecycle of all changes. It also allows the change Manager to implement all the essential changes to be required with the less disruption of IT services.

This management process also allows its owner to recognize and stop any unintended change activity. Actually, this management process is tightly bound with the process **"Service Asset and Configuration Management"**.

Following are the three types of changes which are defined by the ITIL.

1. Normal Change
2. Standard Change
3. Emergency Change

All these changes are also known as the Change Models.

According to the version 3 (V3) of ITIL, following are the eleven sub-processes which comes under this Change management process:

1. Change Management Support
2. RFC (Request for Change) Logging and Review
3. Change Assessment by the Owner (Change Manager)
4. Assess and Implement the Emergency Changes
5. Assessment of change Proposals

6. Change Scheduling and Planning
7. Change Assessment by the CAB
8. Change Development Authorization
9. Implementation or Deployment of Change
10. Minor change Deployment
11. Post Implementation Review and change closure

Release and Deployment Management

In this process, the **Release Manager** plays a role as an owner. Sometimes, this process is also known as the 'ITIL Release Management Process'.

This process allows the Release Manager for managing, planning, and controlling the updates & releases of IT services to the real environment.

Following are the three types of releases which are defined by the ITIL.

1. Minor release
2. Major Release
3. Emergency Release

According to the version 3 (V3) of ITIL, following are the six sub-processes which comes under this Change management process:

1. Release Management Support
2. Release Planning
3. Release build
4. Release Deployment
5. Early Life Support
6. Release Closure

Service Asset and Configuration Management

In this process, the **Configuration Manager** plays a role as an owner.

This management process is a combination of two implicit processes:

1. Asset Management
2. Configuration Management

The aim of this management process is to manage the information about the (CIs) Configuration Items which are needed to deliver the services of IT. It contains information about versions, baselines, and the relationships between assets.

According to the version 3 (V3) of ITIL, following are the five sub-processes which comes under this Change management process:

1. Planning and Management
2. Configuration Control and Identification
3. Status Accounting and reporting
4. Audit and Verification
5. Manage the Information

Knowledge Management

In this process, the **Knowledge Manager** plays a role as an owner. This management process helps the Knowledge Manager by analysing, storing and sharing the knowledge and the data or information in an entire IT organization.

Under this Knowledge Management Process, no sub-process is specified or defined.

Transition Planning and Support

In this process, the **Project Manager** plays a role as an owner. This management process manages the service transition projects. Sometimes, this process is also known as the Project Management Process.

In this process, the project manager is accountable for planning and coordinating resources to deploy IT services within time, cost, and quality estimates.

According to the version 3 (V3) of ITIL, this process performs the following activities:

o It manages the issues and risks.

o It defines the tasks and activities which are to be performed by the separate processes. o It makes a group with the same type of releases.

o It manages each individual deployment as a separate project.

According to the version 3 (V3) of ITIL, following are the four sub-processes which comes under this Project management process:

1. Initiate the Project
2. Planning and Coordination of a Project
3. Project Control
4. Project Communication and Reporting

Service Validation and Testing

In this process, the **Test Manager** plays a role as an owner. The main goal of this management process is that it verifies whether the deployed releases and the resulting IT service meets the customer expectations.

It also checks whether the operations of IT are able to support the new IT services after the deployment. This process allows the Test Manager to remove or delete the errors which are observed at the first phase of the service operation stage in the lifecycle.

It provides the quality assurance for both the services and components. It also identifies the risks, errors and issues, and then they are eliminated through this current stage.

This management process has been released in the version 3 of ITIL as a new process.

Following are the various activities which are performed under this process:

o Validation and Test
Management o Planning and Design
o Verification of Test Plan and Design
o Preparation of the Test Environment
o Testing
o Evaluate Exit Criteria and Report
o Clean up and closure

According to the version 3 (V3) of ITIL, following are the four sub-processes which comes under this management process:

1. Test Model Definition
2. Release Component Acquisition
3. Release Test
4. Service Acceptance Testing

Change Evaluation

In this process, the **Change Manager** plays a role as an owner. The goal of this management process is to avoid the risks which are associated with the major changes for reducing the chances of failures.

This process is started and controlled by the change management and performed by the change manager.

Following are the various activities which are performed under this process:

- It can easily identify the risks.
- It evaluates the effects of a change.

According to the version 3 (V3) of ITIL, following are the four sub-processes which comes under this management process:

1. Change the Evaluation prior to Planning
2. Change the Evaluation prior to Build
3. Change the Evaluation prior to Deployment
4. Change the Evaluation prior after Deployment

Service Operations

Service Operations is the fourth stage in the lifecycle of ITIL. This stage provides the guidelines about how to maintain and manage the stability in services of IT, which helps in achieving the agreed level targets of service delivery.

This stage is also responsible for monitoring the services of IT and fulfilling the requests. In this stage, all the plans of transition and design are measured and executed for the actual efficiency. It is also responsible for resolving the incidents and carrying out the operational tasks.

There are following various essential services or processes which comes under the stage of Service Operations:

o Event Management o
Access Management o
Problem Management

- o Incident Management
- o Application Management
- o Technical Management

Event Management

In this process, the **IT Operations Manager** plays a role as an owner. The main goal of this management process is to make sure that the services of IT and CIs are constantly monitored. It also helps in categorizing the events so that appropriate action can be taken if needed.

In this Management process, the process owner takes all the responsibilities of processes and functions for the multiple service operations.

Following are the various purposes of Event Management Process:

- o It allows the IT Operations Manager to decide the appropriate action for the events.
- o It also provides the trigger for the execution of management activities of many services. o It helps in providing the basis for service assurance and service improvement.

The Event Monitoring Tools are divided into two types, which are defined by the Version 3 (V3) of ITIL:

1. Active Monitoring Tool
2. Passive Monitoring Tool

Following are the three types of events which are defined by the ITIL:

1. Warning
2. Informational
3. Exception

According to the version 3 (V3) of ITIL, following are the four sub-processes which comes under this management process:

1. Event Monitoring and Notification
2. First level Correlation and Event Filtering
3. Second level Correlation and Response Selection
4. Event Review and Closure.

Access Management

In this process, the **Access Manager** plays a role as an owner. This type of Management process is also sometimes called as the **'Identity Management'** or **'Rights Management'**.

The role of a process manager is to provide the rights to use the services for authorized users.

In this Management process, the owner of a process follows those policies and guidelines which are defined by the (ISM) '**Information Security Management**'.

Following are the six activities which come under this management process and are followed sequentially:

1. Request Access
2. Verification
3. Providing Rights
4. Monitoring or Observing the Identity Status
5. Logging and Tracking Status
6. Restricting or Removing Rights

According to the version 3 (V3) of ITIL, following are the two sub-processes which comes under this management process:

1. Maintenance of Catalogue of User Roles and Access profiles
2. Processing of User Access Requests.

Problem Management

In this process, the **Problem Manager** plays a role as an owner. The main goal of this management process is to maintain or manage the life cycle of all the problems which happen in the services of IT. In the ITIL Framework, the problem is referred to as "**an unknown cause or event of one or more incident**".

It helps in finding the root cause of the problem. It also helps in maintaining the information about the problems.

Following are the ten activities which come under this management process and are followed sequentially. These ten activities are also called as a lifecycle of Problem Management:

1. Problem Detection
2. Problem Logging
3. Categorization of a Problem

4. Prioritization of a Problem
5. Investigation and Diagnosis of a Problem
6. Identify Workaround
7. Raising a Known Error Record
8. Resolution of a Problem
9. Problem Closure
10. Major Problem Review

Incident Management

In this process, the **Incident Manager** plays a role as an owner. The main goal of this management process is to maintain or manage the life cycle of all the incidents which happen in the services of IT.

An incident is a term which is defined as the failure of any Configuration Item (CI) or reduction in the quality of services of IT.

This management process maintains the satisfaction of users by managing the qualities of IT service. It increases the visibility of incidents.

According to the version 3 (V3) of ITIL, following are the nine sub-processes which comes under this management process:

1. Incident Management Support
2. Incident Logging and Categorization
3. Pro-active User Information
4. First Level Support for Immediate Incident Resolution
5. Second Level Support for Incident Resolution
6. Handling of Major Incidents
7. Incident Monitoring and Escalation
8. Closure and Evaluation of Incident
9. Management Reporting of Incident

Application Management

In this function, the **Application Analyst** plays a role as an owner.

DevOps

This management function maintains or improves the applications throughout the entire service lifecycle. This function plays an important and essential role in the applications and system management.

Under this management function, no sub-process is specified or defined. But, this management function into the following six activities or stages:

1. Define 2. Design 3. Build 4. Deploy 5. Operate 6. Optimize

Technical Management

In this function, the **Technical Analyst** plays a role as an owner. This function acts as standalone in the IT organizations, which basically consists of technical people and teams. The main goal of this function is to provide or offer the technical expertise. And, it also supports for maintaining or managing of IT infrastructure throughout the entire lifecycle of a service.

The role of the Technical Analyst is to develop the skills, which are required to

operate the day-to-day operations of IT infrastructure. Under this management function, no sub-process is specified or defined.

Continual Service Improvement

It is the fifth stage in the lifecycle of ITIL service. This stage helps to identify and implement strategies, which is used for providing better services in future.

Following are the various objectives or goals under this CSI:

- It improves the quality services by learning from the past failures.
- It also helps in analyzing and reviewing the improvement opportunities in every phase of the service lifecycle.
- It also evaluates the service level achievement results.
- It also describes the best guidelines to achieve the large-scale improvements in the quality of service.
- It also helps in describing the concept of KPI, which is a process metrics-driven for evaluating and reviewing the performance of the services.

There are following various essential services or processes which comes under the stage of CSI:

- Service Review
- Process Evaluation

- o Definition of CSI Initiatives
- o Monitoring of CSI Initiatives

This stage follows the following six-step approach (pre-defined question) for planning, reviewing, and implementing the improvement process:

Service Review

In this process, the **CSI Manager** plays a role as an owner. The main aim of this management process is to review the services of business and infrastructure on a regular basis.

Sometimes, this process is also called as "**ITIL Service Review and Reporting**". Under this management process, no sub-process is specified or defined.

Process Evaluation

In this process, the **Process Architect** plays a role as an owner. The main aim of this management process is to evaluate the processes of IT services on a regular basis. This process accepts inputs from the process of **Service Review** and provides its output to the process of **Definition of CSI Initiatives**.

In this process, the process owner is responsible for maintaining and managing the process architecture and also ensures that all the processes of services cooperate in a seamless way.

According to the version 3 (V3) of ITIL, following are the five sub-processes which comes under this management process:

1. Process Management support
2. Process Benchmarking
3. Process Maturity Assessment
4. Process Audit
5. Process Control and Review

Definition of CSI Initiatives

In this process, the **CSI Manager** plays a role as an owner. This management process is also called/known as a "**Definition of Improvement Initiatives**".

Definition of CSI Initiatives is a process, which is used for describing the particular initiatives whose aim is to improve the qualities of IT services and processes.

In this process, the CSI Manager (process owner) is accountable for managing and maintaining the CSI registers and also helps in taking the good decisions regarding improvement initiatives.

Under this management process, no sub-process is specified or defined.

Monitoring of CSI Initiatives

In this process, the **CSI Manager** plays a role as an owner. This management process is also called as a "**CSI Monitoring**".

Under this management process, no sub-process is specified or defined.

Advantages of ITIL

Following are the various advantages or benefits of ITIL:

1. One of the best advantages of ITIL is that it helps in increasing the customer satisfaction.
2. It allows managers to improve the decision-making process.
3. It is also used for creating the clear structure of an organization.
4. It also helps managers by controlling the infrastructure services.
5. It improves the interaction between the customers and the service provider.
6. With the help of this framework, service delivery is also improved.
7. It establishes the framework of ITSM for the organization.

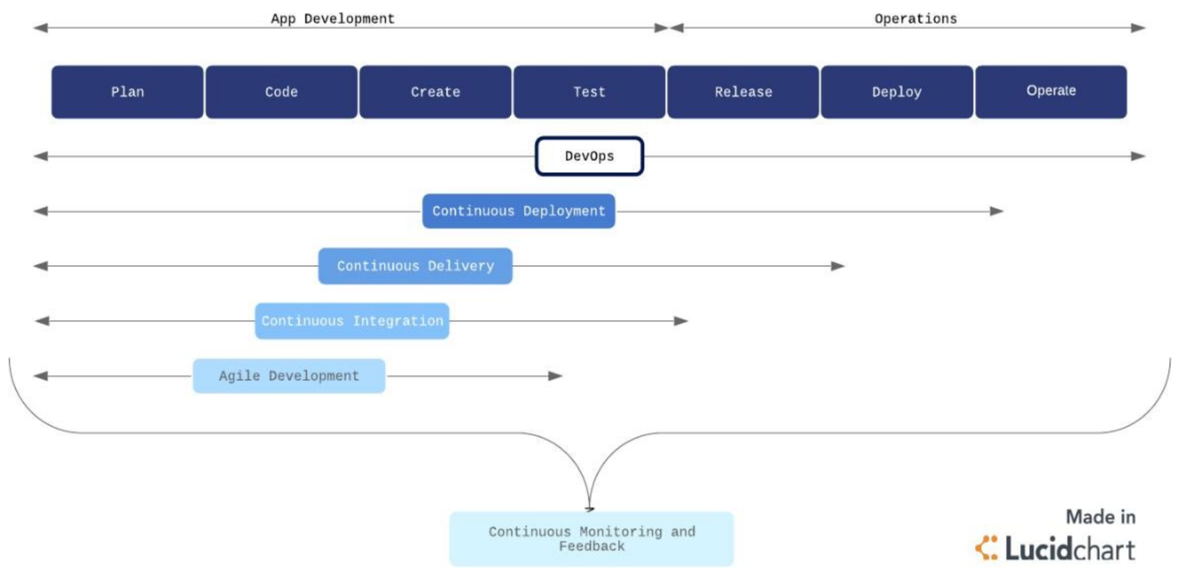
DevOps

DevOps Process

The DevOps process flow

The DevOps process flow is all about agility and automation. Each phase in the DevOps lifecycle focuses on closing the loop between development and operations and driving production through continuous development, integration, testing, monitoring and feedback, delivery, and deployment.

DevOps Process Flow (Click on image to modify



DevOps

Continuous development

Continuous development is an umbrella term that describes the iterative process for developing software to be delivered to customers. It involves continuous integration, continuous testing, continuous delivery, and continuous deployment.

By implementing a continuous development strategy and its associated sub-strategies, businesses can achieve faster delivery of new features or products that are of higher quality and lower risk, without running into significantly bandwidth barriers.

Continuous integration

Continuous integration (CI) is a software development practice commonly applied in the DevOps process flow. Developers regularly merge their code changes into a shared repository where those updates are automatically tested.

Continuous integration ensures the most up-to-date and validated code is always readily available to developers. CI helps prevent costly delays in development by allowing multiple developers to work on the same source code with confidence, rather than waiting to integrate separate sections of code all at once on release day.

This practice is a crucial component of the DevOps process flow, which aims to combine speed and agility with reliability and security.

Continuous testing

Continuous testing is a verification process that allows developers to ensure the code actually works the way it was intended to in a live environment. Testing can surface bugs and particular aspects of the product that may need fixing or improvement, and can be pushed back to the development stages for continued improvement.

Continuous monitoring and feedback

Throughout the development pipeline, your team should have measures in place for continuous monitoring and feedback of the products and systems. Again, the majority of the monitoring process should be automated to provide continuous feedback.

This process allows IT operations to identify issues and notify developers in real time. Continuous feedback ensures higher security and system reliability as well as more agile responses when issues do arises.

Continuous delivery

Continuous delivery (CD) is the next logical step from CI. Code changes are automatically built, tested, and packaged for release into production. The goal is to release updates to the users rapidly and sustainably.

To do this, CD automates the release process (building on the automated testing in CI) so that new builds can be released at the click of a button.

Continuous deployment

For the seasoned DevOps organization, continuous deployment may be the better option over CD. Continuous deployment is the fully automated version of CD with no human (i.e., manual) intervention necessary.

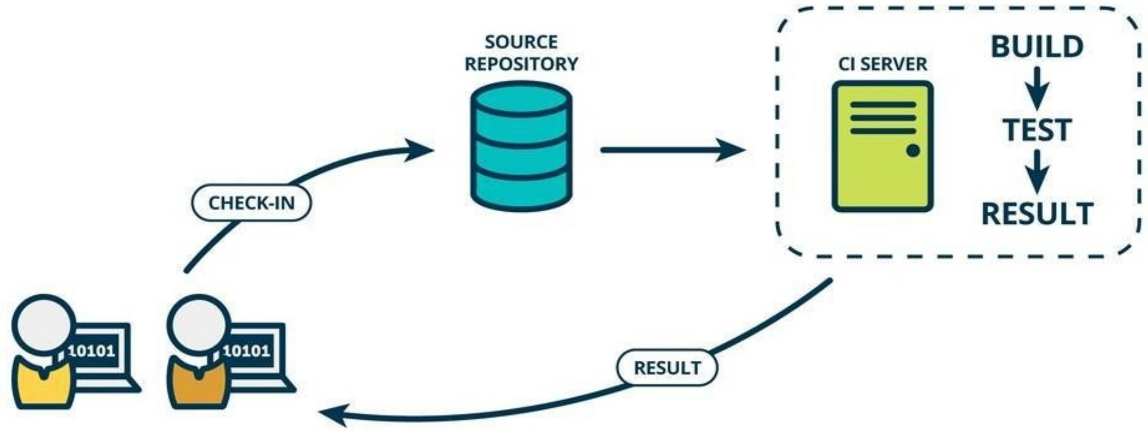
In a continuous deployment process, every validated change is automatically released to users. This process eliminates the need for scheduled release days and accelerates the feedback loop. Smaller, more frequent releases allow developers to get user feedback quickly and address issues with more agility and accuracy.

Continuous deployment is a great goal for a DevOps team, but it is best applied after the DevOps process has been ironed out. For continuous deployment to work well, organizations need to have a rigorous and reliable automated testing environment. If you're not there yet, starting with CI and CD will help you get there.

Continuous Delivery

Continuous delivery is an approach where teams release quality products frequently and predictably from source code repository to production in an automated fashion.

Some organizations release products manually by handing them off from one team to the next, which is illustrated in the diagram below. Typically, developers are at the left end of this spectrum and operations personnel are at the receiving end. This creates delays at every hand-off that leads to frustrated teams and dissatisfied customers. The product eventually goes live through a tedious and error-prone process that delays revenue generation.



How does continuous delivery work?

A continuous delivery pipeline could have a manual gate right before production. A manual gate requires human intervention, and there could be scenarios in your organization that require manual gates in pipelines. Some manual gates might be questionable, whereas some could be legitimate. One legitimate scenario allows the business team to make a last-minute release decision. The engineering team keeps a shippable version of the product ready after every sprint, and the business team makes the final call to release the product to all customers, or a cross-section of the population, or perhaps to people who live in a certain geographical location.

The architecture of the product that flows through the pipeline is a key factor that determines the anatomy of the continuous delivery pipeline. A highly coupled product architecture generates a complicated graphical pipeline pattern where various pipelines could get entangled before eventually making it to production.

The product architecture also influences the different phases of the pipeline and what artifacts are produced in each phase. The pipeline first builds components - the smallest distributable and testable units of the product. For example, a library built by the pipeline can be termed a component. This is the component phase.

Loosely coupled components make up subsystems - the smallest deployable and runnable units. For example, a server is a subsystem. A microservice running in a container is also an example of a subsystem. This is the subsystem phase. As opposed to components, subsystems can be stood up and tested.

The software delivery pipeline is a product in its own right and should be a priority for businesses. Otherwise, you should not send revenue-generating products through it. Continuous delivery adds value in three ways. It improves velocity, productivity, and sustainability of software development teams.

Velocity

Velocity means responsible speed and not suicidal speed. Pipelines are meant to ship quality products to customers. Unless teams are disciplined, pipelines can shoot faulty code to production, only faster! Automated software delivery pipelines help organizations respond to market changes better.

Productivity

A spike in productivity results when tedious tasks, like submitting a change request for every change that goes to production, can be performed by pipelines instead of humans. This lets scrum teams focus on products that wow the world, instead of draining their energy on logistics. And that can make team members happier, more engaged in their work, and want to stay on the team longer.

Sustainability

Sustainability is key for all businesses, not just tech. “Software is eating the world” is no longer true — software has already consumed the world! Every company at the end of the day, whether in healthcare, finance, retail, or some other domain, uses technology to differentiate and outmaneuver their competition. Automation helps reduce/eliminate manual tasks that are error-prone and repetitive, thus positioning the business to innovate better and faster to meet their customers' needs.

Release Management

Release management is the process of overseeing the planning, scheduling, and controlling of software builds throughout each stage of development and across various environments. Release management typically included the testing and deployment of software releases as well.

DevOps

Release management has had an important role in the software development lifecycle since before it was known as release management. Deciding when and how to release updates was its own unique problem even when software saw physical disc releases with updates occurring as seldom as every few years.

Now that most software has moved from hard and fast release dates to the software as a service (SaaS) business model, release management has become a constant process that works alongside development. This is especially true for businesses that have converted to utilizing continuous delivery pipelines that see new releases occurring at blistering rates. DevOps now plays a large role in many of the duties that were originally considered to be under the purview of release management roles; however, DevOps has not resulted in the obsolescence of release management.

Advantages of Release Management for DevOps

With the transition to DevOps practices, deployment duties have shifted onto the shoulders of the DevOps teams. This doesn't remove the need for release management; instead, it modifies the data points that matter most to the new role release management performs.

Release management acts as a method for filling the data gap in DevOps. The planning of implementation and rollback safety nets is part of the DevOps world, but release management still needs to keep tabs on applications, its components, and the promotion schedule as part of change orders. The key to managing software releases in a way that keeps pace with DevOps deployment schedules is through automated management tools.

Aligning business & IT goals

The modern business is under more pressure than ever to continuously deliver new features and boost their value to customers. Buyers have come to expect that their software evolves and continues to develop innovative ways to meet their needs. Businesses create an outside perspective to glean insights into their customer needs. However, IT has to have an inside perspective to develop these features.

Release management provides a critical bridge between these two gaps in perspective. It coordinates between IT work and business goals to maximize the success of each release.

Release management balances customer desires with development work to deliver the greatest value to users.

Minimizes organizational risk

Software products contain millions of interconnected parts that create an enormous risk of failure. Users are often affected differently by bugs depending on their other software, applications, and tools. Plus, faster deployments to production increase the overall risk that faulty code and bugs slip through the cracks.

Release management minimizes the risk of failure by employing various strategies. Testing and governance can catch critical faulty sections of code before they reach the customer. Deployment plans ensure there are enough team members and resources to address any potential issues before affecting users. All dependencies between the millions of interconnected parts are recognized and understood.

Direct accelerating change

Release management is foundational to the discipline and skill of continuously producing enterprise-quality software. The rate of software delivery continues to accelerate and is unlikely to slow down anytime soon. The speed of changes makes release management more necessary than ever.

The move towards CI/CD and increases in automation ensure that the acceleration will only increase. However, it also means increased risk, unmet governance requirements, and potential disorder. Release management helps promote a culture of excellence to scale DevOps to an organizational level.

Release management best practices

As DevOps increases and changes accelerate, it is critical to have best practices in place to ensure that it moves as quickly as possible. Well-refined processes enable DevOps teams to more effectively and efficiently. Some best practices to improve your processes include:

Define clear criteria for success

Well-defined requirements in releases and testing will create more dependable releases. Everyone should clearly understand when things are actually ready to ship.

Well-defined means that the criteria cannot be subjective. Any subjective criteria will keep you from learning from mistakes and refining your release management process to identify what works best. It also needs to be defined for every team member. Release managers, quality supervisors, product vendors, and product owners must all have an agreed-upon set of criteria before starting a project.

Minimize downtime

DevOps is about creating an ideal customer experience. Likewise, the goal of release management is to minimize the amount of disruption that customers feel with updates.

Strive to consistently reduce customer impact and downtime with active monitoring, proactive testing, and real-time collaborative alerts that enable you to quickly notify you of issues during a release. A good release manager will be able to identify any problems before the customer.

The team can resolve incidents quickly and experience a successful release when proactive efforts are combined with a collaborative response plan.

Optimize your staging environment

The staging environment requires constant upkeep. Maintaining an environment that is as close as possible to your production one ensures smoother and more successful releases. From QA to product owners, the whole team must maintain the staging environment by running tests and combing through staging to find potential issues with deployment. Identifying problems in staging before deploying to production is only possible with the right staging environment.
Maintaining a staging environment that is as close as possible to production will enable DevOps teams to confirm that all releases will meet acceptance criteria more quickly.

DevOps

Strive for immutable

Whenever possible, aim to create new updates as opposed to modifying new ones. Immutable programming drives teams to build entirely new configurations instead of changing existing structures. These new updates reduce the risk of bugs and errors that typically happen when modifying current configurations.

The inherently reliable releases will result in more satisfied customers and employees.

Keep detailed records

Good records management on any release/deployment artifacts is critical. From release notes to binaries to compilation of known errors, records are vital for reproducing entire sets of assets. In most cases, tacit knowledge is required.

Focus on the team

Well-defined and implemented DevOps procedures will usually create a more effective release management structure. They enable best practices for testing and cooperation during the complete delivery lifecycle.

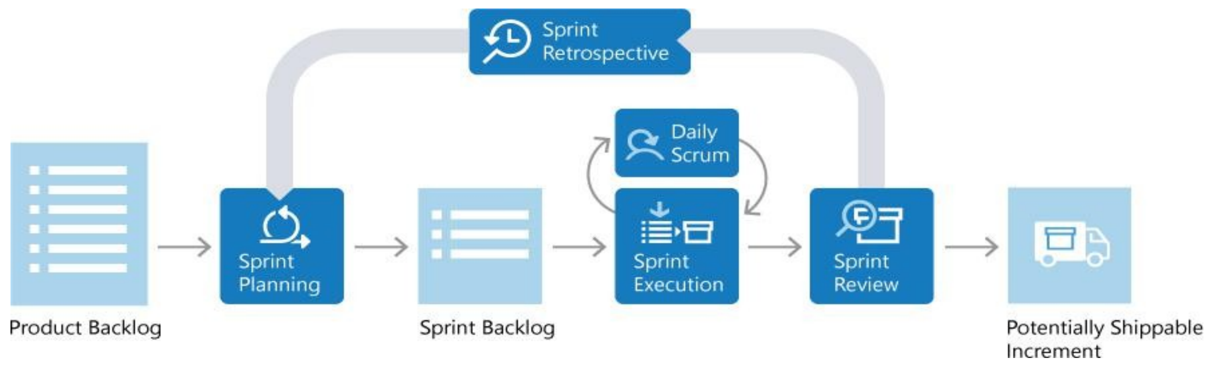
Although automation is a critical aspect of DevOps and release management, it aims to enhance team productivity. The more that release management and DevOps focus on decreasing human error and improving operational efficiency, the more they'll start to quickly release dependable services.

Scrum

Scrum is a framework used by teams to manage work and solve problems collaboratively in short cycles. Scrum implements the principles of Agile as a concrete set of artifacts, practices, and roles.

The Scrum lifecycle

The diagram below details the iterative Scrum lifecycle. The entire lifecycle is completed in fixed time periods called sprints. A sprint is typically one-to-four weeks long



Scrum roles

There are three key roles in Scrum: the product owner, the *Scrum master*, and the *Scrum team*.

Product owner

The product owner is responsible for what the team builds, and why they build it. The product owner is responsible for keeping the backlog of work up to date and in priority order.

Scrum master

The Scrum master ensures that the Scrum process is followed by the team. Scrum masters are continually on the lookout for how the team can improve, while also resolving impediments and other blocking issues that arise during the sprint. Scrum masters are part coach, part team member, and part cheerleader.

Scrum team

The members of the Scrum team actually build the product. The team owns the engineering of the product, and the quality that goes with it.

Product backlog

The product backlog is a prioritized list of work the team can deliver. The product owner is responsible for adding, changing, and reprioritizing the backlog as needed. The items at the top of the backlog should always be ready for the team to execute on.

Plan the sprint

In sprint planning, the team chooses backlog items to work on in the upcoming sprint. The team chooses backlog items based on priority and what they believe they can complete in the sprint. The *sprint backlog* is the list of items the team plans to deliver in the sprint. Often, each item on the sprint backlog is broken down into tasks. Once all members agree the sprint backlog is achievable, the sprint starts.

Execute the sprint

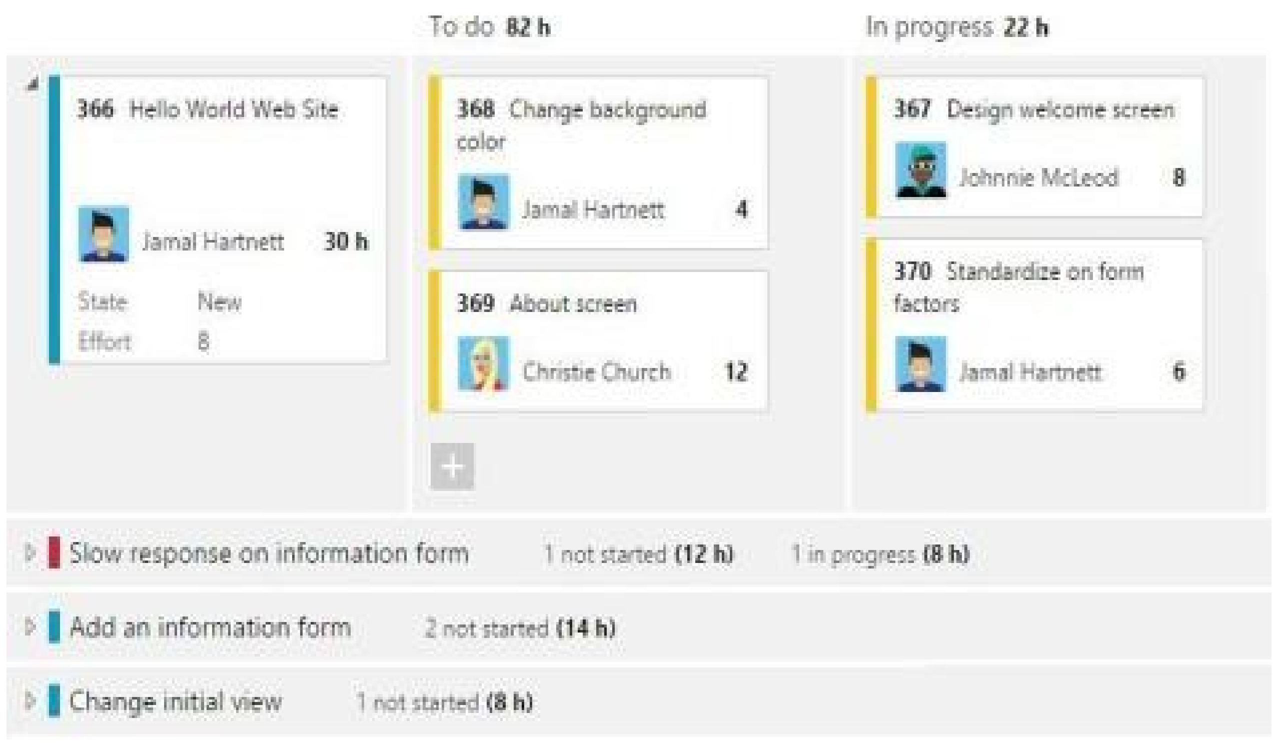
Once the sprint starts, the team executes on the sprint backlog. Scrum does not specify how the team should execute. The team decides how to manage its own work.

Scrum defines a practice called a *daily Scrum*, often called the *daily standup*. The daily Scrum is a daily meeting limited to fifteen minutes. Team members often stand during the meeting to ensure it stays brief. Each team member briefly reports their progress since yesterday, the plans for today, and anything impeding their progress.

To aid the daily Scrum, teams often review two artifacts:

Task board

The task board lists each backlog item the team is working on, broken down into the tasks required to complete it. Tasks are placed in **To do**, **In progress**, and **Done** columns based on their status. The board provides a visual way to track the progress of each backlog item.


To do 82 h

In progress 22 h

366 Hello World Web Site
Jamal Hartnett 30 h
State New
Effort 8

368 Change background color
Jamal Hartnett 4

369 About screen
Christie Church 12

367 Design welcome screen
Johnnie McLeod 8

370 Standardize on form factors
Jamal Hartnett 6

Slow response on information form 1 not started (12 h) 1 in progress (8 h)

Add an information form 2 not started (14 h)

Change initial view 1 not started (8 h)

.

Sprint burndown chart

The sprint burndown is a graph that plots the daily total of remaining work, typically shown in hours. The burndown chart provides a visual way of showing whether the team is on track to complete all the work by the end of the sprint.

Sprint review and sprint retrospective

At the end of the sprint, the team performs two practices:

Sprint review

The team demonstrates what they've accomplished to stakeholders. They demo the software and show its value.

Sprint retrospective

The team takes time to reflect on what went well and which areas need improvement. The outcome of the retrospective are actions for the next sprint.

Increment

The product of a sprint is called the *increment* or *potentially shippable increment*. Regardless of the term, a sprint's output should be of shippable quality, even if it's part of something bigger and can't ship by itself. It should meet all the quality criteria set by the team and product owner.

Repeat, learn, improve

The entire cycle is repeated for the next sprint. Sprint planning selects the next items on the product backlog and the cycle repeats. While the team executes the sprint, the product owner ensures the items at the top of the backlog are ready to execute in the following sprint.

This shorter, iterative cycle provides the team with lots of opportunities to learn and improve. A traditional project often has a long lifecycle, say 6-12 months. While a team can learn from a traditional project, the opportunities are far less than a team who executes in two-week sprints, for example.

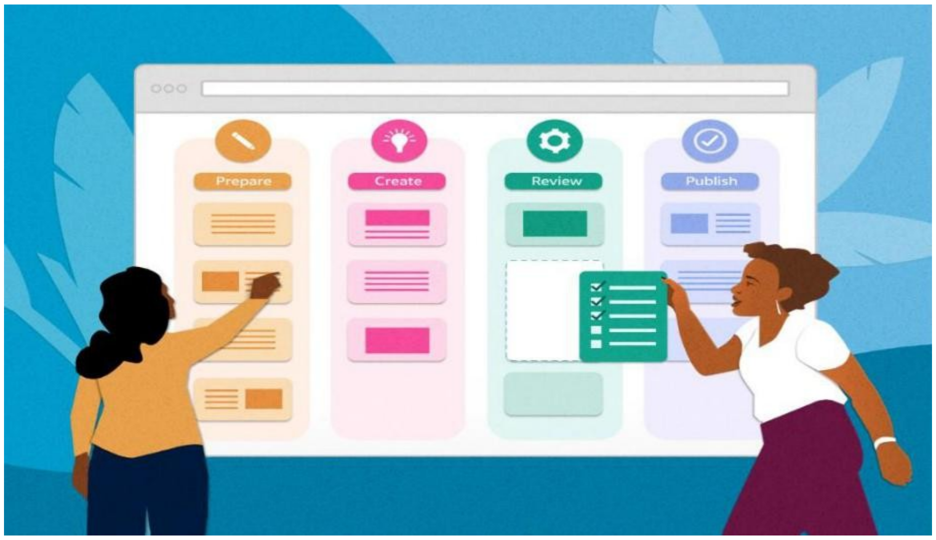
This iterative cycle is, in many ways, the essence of Agile.

Scrum is very popular because it provides just enough framework to guide teams while giving them flexibility in how they execute. Its concepts are simple and easy to learn. Teams can get started quickly and learn as they go. All of this makes Scrum a great choice for teams just starting to implement Agile principles.

Kanban

Kanban is a Japanese term that means signboard or billboard. An industrial engineer named Taiichi Ohno developed Kanban at Toyota Motor Corporation to improve manufacturing efficiency.

Although Kanban was created for manufacturing, software development shares many of the same goals, such as increasing flow and throughput. Software development teams can improve their efficiency and deliver value to users faster by using Kanban guiding principles and methods.



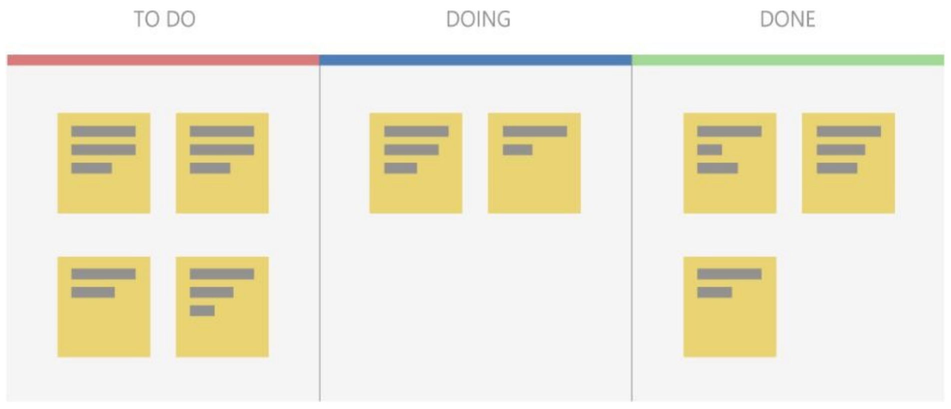
Kanban principles

Adopting Kanban requires adherence to some fundamental practices that might vary from teams' previous methods.

Visualize work

Understanding development team status and work progress can be challenging. Work progress and current state is easier to understand when presented visually rather than as a list of work items or a document.

Visualization of work is a key principle that Kanban addresses primarily through *Kanban boards*. These boards use cards organized by progress to communicate overall status. Visualizing work as cards in different states on a board helps to easily see the big picture of where a project currently stands, as well as identify potential bottlenecks that could affect productivity.



Use a pull model

Historically, stakeholders requested functionality by pushing work onto development teams, often with tight deadlines. Quality suffered if teams had to take shortcuts to deliver the functionality within the timeframe.

Kanban focuses on maintaining an agreed-upon level of quality that must be met before considering work done. To support this model, stakeholders don't push work on teams that are already working at capacity. Instead, stakeholders add requests to a backlog that a team pulls into their workflow as capacity becomes available.

Impose a WIP limit

Teams that try to work on too many things at once can suffer from reduced productivity due to frequent and costly context switching. The team is busy, but work doesn't get done, resulting in unacceptably high lead times. Limiting the number of backlog items a team can work on at a time helps increase focus while reducing context switching. The items the team is currently working on are called *work in progress (WIP)*.

Teams decide on a *WIP limit*, or maximum number of items they can work on at one time. A well-disciplined team makes sure not to exceed their WIP limit. If teams exceed their WIP limits, they investigate the reason and work to address the root cause.

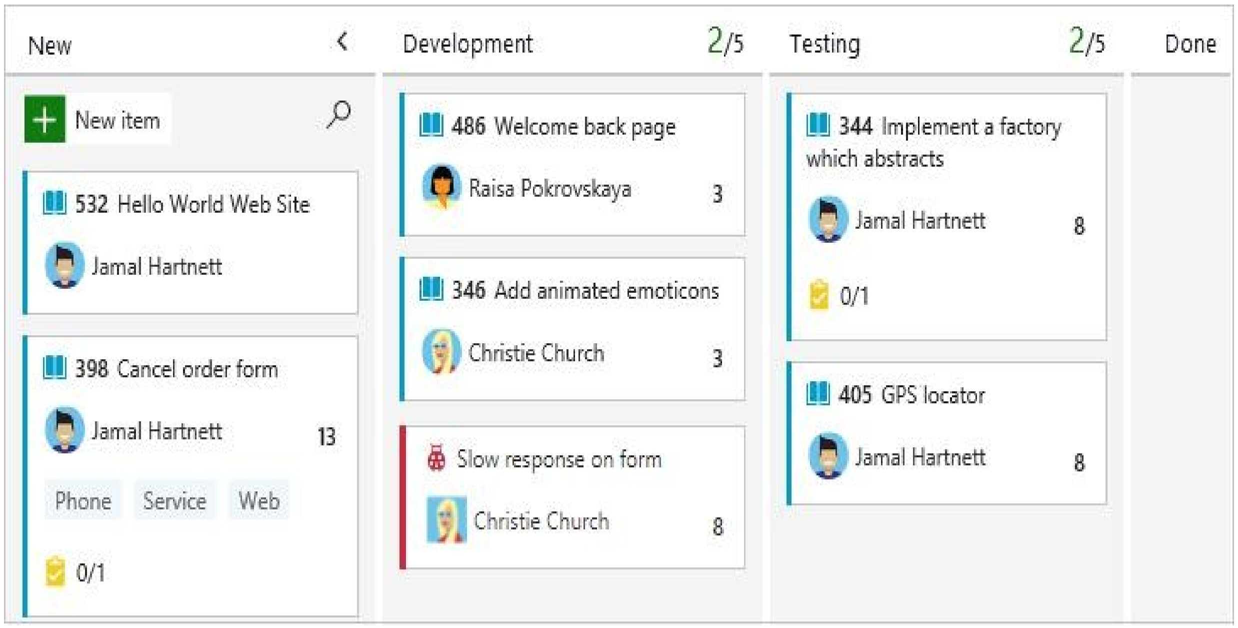
Measure continuous improvement

To practice continuous improvement, development teams need a way to measure effectiveness and throughput. Kanban boards provide a dynamic view of the states of work in a workflow, so teams can experiment with processes and more easily evaluate impact on workflows. Teams that embrace Kanban for continuous improvement use measurements like *lead time* and *cycle time*.

Kanban boards

The *Kanban board* is one of the tools teams use to implement Kanban practices. A Kanban board can be a physical board or a software application that shows cards arranged into columns. Typical column names are **To-do**, **Doing**, and **Done**, but teams can customize the names to match their workflow states. For example, a team might prefer to use **New**, **Development**, **Testing**, **UAT**, and **Done**.

Software development-based Kanban boards display cards that correspond to product backlog items. The cards include links to other items, such as tasks and test cases. Teams can customize the cards to include information relevant to their process.

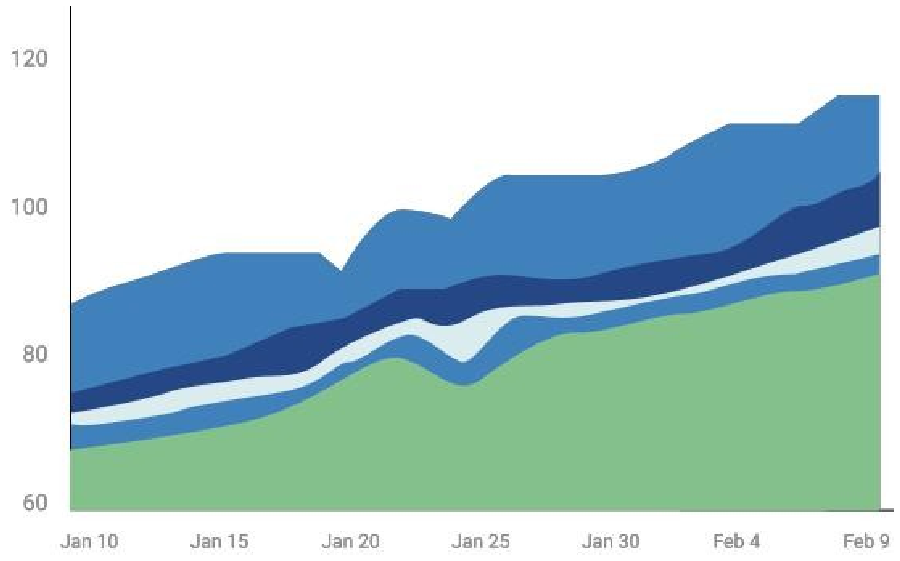


On a Kanban board, the WIP limit applies to all in-progress columns. WIP limits don't apply to the first and last columns, because those columns represent work that hasn't started or is completed. Kanban boards help teams stay within WIP limits by drawing attention to columns that exceed the limits. Teams can then determine a course of action to remove the bottleneck.

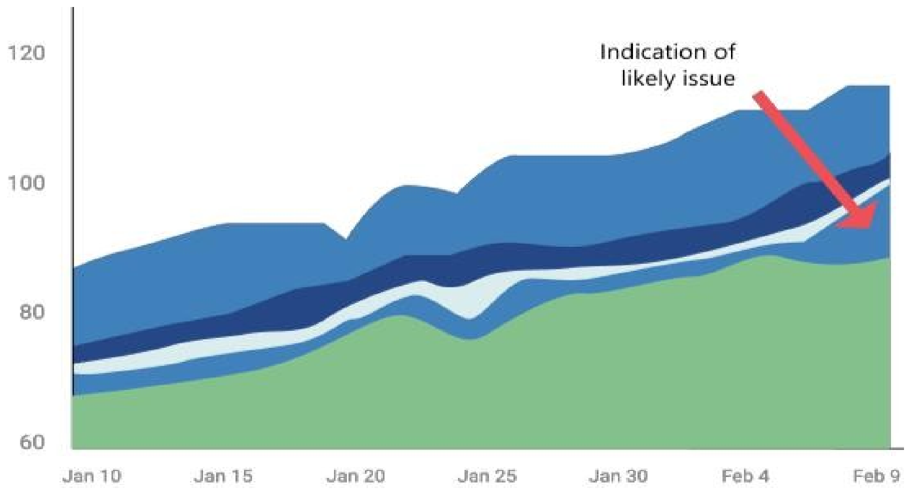
Cumulative flow diagrams

A common addition to software development-based Kanban boards is a chart called a *cumulative flow diagram (CFD)*. The CFD illustrates the number of items in each state over time, typically across several weeks. The horizontal axis shows the timeline, while the vertical axis shows the number of product backlog items. Colored areas indicate the states or columns the cards are currently in.

The CFD is particularly useful for identifying trends over time, including bottlenecks and other disruptions to progress velocity. A good CFD shows a consistent upward trend while a team is working on a project. The colored areas across the chart should be roughly parallel if the team is working within their WIP limits.



A bulge in one or more of the colored areas usually indicates a bottleneck or impediment in the team's flow. In the following CFD, the completed work in green is flat, while the testing state in blue is growing, probably due to a bottleneck.



Kanban and Scrum in Agile development

While broadly fitting under the umbrella of Agile development, Scrum and Kanban are quite different.

- Scrum focuses on fixed length sprints, while Kanban is a continuous flow model.
- Scrum has defined roles, while Kanban doesn't define any team roles.
- Scrum uses velocity as a key metric, while Kanban uses cycle time.

Teams commonly adopt aspects of both Scrum and Kanban to help them work most effectively. Regardless of which characteristics they choose, teams can always review and adapt until they find the best fit. Teams should start simple and not lose sight of the importance of delivering value regularly to users.

Kanban with GitHub

GitHub offers a Kanban experience through project boards (classic). These boards help you organize and prioritize work for specific feature development, comprehensive roadmaps, or release checklists. You can automate project boards (classic) to sync card status with associated issues and pull requests.

DevOps

Kanban with Azure Boards

Azure Boards provides a comprehensive Kanban solution for DevOps planning. Azure Boards has deep integration across Azure DevOps, and can also be part of Azure Boards-GitHub integration.

- For more information, see Reasons to use Azure Boards to plan and track your work.
- The Learn module Choose an Agile approach to software development provides hands-on Kanban experience in Azure Boards.

Delivery Pipeline

A DevOps pipeline is a set of automated processes and tools that allows both developers and operations professionals to work cohesively to build and deploy code to a production environment.

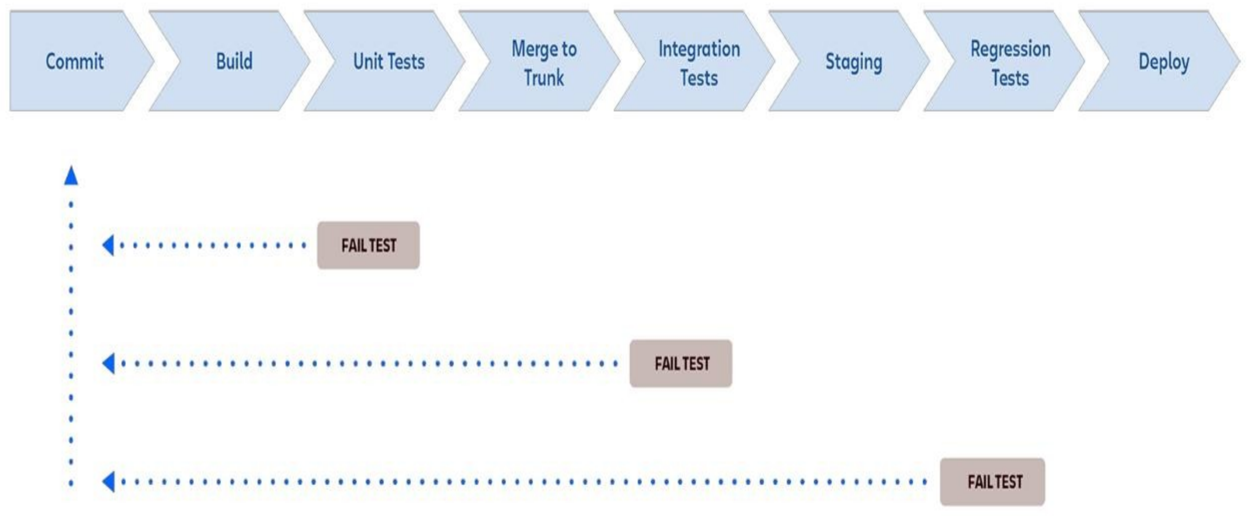
While a DevOps pipeline can differ by organization, it typically includes build automation/continuous integration, automation testing, validation, and reporting. It may also include one or more manual gates that require human intervention before code is allowed to proceed.

Continuous is a differentiated characteristic of a DevOps pipeline. This includes continuous integration, continuous delivery/deployment (CI/CD), continuous feedback, and continuous operations. Instead of one-off tests or scheduled deployments, each function occurs on an ongoing basis.

Considerations for building a DevOps pipeline

Since there isn't one standard DevOps pipeline, an organization's design and implementation of a DevOps pipeline depends on its technology stack, a DevOps engineer's level of experience, budget, and more. A DevOps engineer should have a wide-ranging knowledge of both development and operations, including coding, infrastructure management, system administration, and DevOps toolchains.

Plus, each organization has a different technology stack that can impact the process. For example, if your codebase is node.js, factors include whether you use a local proxy npm registry, whether you download the source code and run `npm install` at every stage in the pipeline, or do it once and generate an artifact that moves through the pipeline. Or, if an application is container-based, you need to decide to use a local or remote container registry, build the container once and move it through the pipeline, or rebuild it at every stage.



While every pipeline is unique, most organizations use similar fundamental components. Each step is evaluated for success before moving on to the next stage of the pipeline. In the event of a failure, the pipeline is stopped, and feedback is provided to the developer.

Components of a DevOps pipeline

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

Continuous integration is the practice of making frequent commits to a common source code repository. It's continuously integrating code changes into existing code base so that any conflicts between different developer's code changes are quickly identified and relatively easy to remediate. This practice is critically important to increasing deployment efficiency.
We believe that trunk-based development is a requirement of continuous integration. If you are not making frequent commits to a common branch in a shared source code repository, you are not doing continuous integration. If your build and test processes are automated but your developers are working on isolated, long-living feature branches that are infrequently integrated into a shared branch, you are also not doing continuous integration.

Continuous delivery ensures that the "main" or "trunk" branch of an application's source code is always in a releasable state. In other words, if management came to your desk at 4:30 PM on a Friday and said, "We need the latest version released right now," that version could be deployed with the push of a button and without fear of failure.
This means having a pre-production environment that is as close to identical to the production environment as possible and ensuring that automated tests are executed, so that every variable that might cause a failure is identified before code is merged into the main or trunk branch.

Continuous deployment entails having a level of continuous testing and operations that is so robust, new versions of software are validated and deployed into a production environment without requiring any human intervention.
This is rare and in most cases unnecessary. It is typically only the unicorn businesses who have hundreds or thousands of developers and have many releases each day that require, or even want to have, this level of automation.

To simplify the difference between continuous delivery and continuous deployment, think of delivery as the FedEx person handing you a box, and deployment as you opening that box and using what's inside. If a change to the product is required between the time you receive the box and when you open it, the manufacturer is in trouble!

3. **Continuous feedback**

4. The single biggest pain point of the old waterfall method of software development — and consequently why agile methodologies were designed — was the lack of timely feedback. When new features took months or years to go from idea to implementation, it was almost guaranteed that the end result would be something other than what the customer expected or wanted. Agile succeeded in ensuring that developers received faster feedback from stakeholders. Now with DevOps, developers receive continuous feedback not not only from stakeholders, but from systematic testing and monitoring of their code in the pipeline.

Continuous testing is a critical component of every DevOps pipeline and one of the primary enablers of continuous feedback. In a DevOps process, changes move continuously from development to testing to deployment, which leads not only to faster releases, but a higher quality product. This means having automated tests throughout your pipeline, including unit tests that run on every build change, smoke tests, functional tests, and end-to-end tests.
Continuous monitoring is another important component of continuous feedback. A DevOps approach entails using continuous monitoring in the staging, testing, and even development environments. It is sometimes useful to monitor pre-production environments for anomalous behavior, but in general this is an approach used to continuously assess the health and performance of applications in production.
Numerous tools and services exist to provide this functionality, and this may involve anything from monitoring your on-premise or cloud infrastructure such as server resources, networking, etc. or the performance of your application or its API interfaces.

3. Continuous operations

Continuous operations is a relatively new and less common term, and definitions vary. One way to interpret it is as "continuous uptime". For example in the case of a blue/green deployment

strategy in which you have two separate production environments, one that is “blue” (publicly accessible) and one that is “green” (not publicly accessible). In this situation, new code would be deployed to the green environment, and when it was confirmed to be functional then a switch would be flipped (usually on a load-balancer) and traffic would switch from the “blue” system to the “green” system. The result is no downtime for the end-users.
Another way to think of Continuous operations is as **continuous alerting**. This is the notion that engineering staff is on-call and notified if any performance anomalies in the application or infrastructure occur. In most cases, continuous alerting goes hand in hand with continuous monitoring.

One of the main goals of DevOps is to improve the overall workflow in the software development life cycle (SDLC). The flow of work is often described as WIP or work in progress. Improving WIP can be accomplished by a variety of means. In order to effectively remove bottlenecks that decrease the flow of WIP, one must first analyze the people, process, and technology aspects of the entire SDLC.
These are the 11 bottlenecks that have the biggest impact on the flow of work.

1. Inconsistent Environments

In almost every company I have worked for or consulted with, a huge amount of waste exists because the various environments (dev, test, stage, prod) are configured differently. I call this “environment hell”. How many times have you heard a developer say “it worked on my laptop”? As code moves from one environment to the next, software breaks because of the different configurations within each environment. I have seen teams waste days and even weeks fixing bugs that are due to environmental issues and are not due to errors within the code. Inconsistent environments are the number one killer of agility.

Create standard infrastructure blueprints and implement continuous delivery to ensure all environments are identical.

2. Manual Intervention

Manual intervention leads to human error and non-repeatable processes. Two areas where manual intervention can disrupt agility the most are in testing and deployments. If testing is performed manually, it is impossible to implement continuous integration and continuous delivery in an agile manner (if at all). Also, manual testing increases the chance of producing defects, creating unplanned work. When deployments are performed fully or partially manual, the risk of deployment failure increases significantly which lowers quality and reliability and increases unplanned work.

Automate the build and deployment processes and implement a test automation methodology like test driven development (TDD)

3. SDLC Maturity

The maturity of a team's software development lifecycle (SDLC) has a direct impact on their ability to deliver software. There is nothing new here; SDLC maturity has plagued IT for decades. In the age of DevOps, where we strive to deliver software in shorter increments with a high degree of reliability and quality, it is even more critical for a team to have a mature process.

Some companies I visit are still practicing waterfall methodologies. These companies struggle with DevOps because they don't have any experience with agile. But not all companies that practice agile do it well. Some are early in their agile journey, while others have implemented what I call "Wagile": waterfall tendencies with agile terminology sprinkled in. I have seen teams who have implemented Kanban but struggle with the prioritization and control of WIP. I have seen scrum teams struggle to complete the story points that they promised. It takes time to get really good at agile.

Invest in training and hold blameless post mortems to continously solicit feedback and improve.

4. Legacy Change Management Processes

Many companies have had their change management processes in place for years and are comfortable with it. The problem is that these processes were created back when companies were deploying and updating back office solutions or infrastructure changes that happened infrequently. Fast forward to today's environments where applications are made of many small components or micro services that can be changed and deployed quickly, now all of a sudden the process gets in the way.

Many large companies with well-established ITIL processes struggle with DevOps. In these environments I have seen development teams implement highly automated CI/CD processes only to stop and wait for weekly manual review gates. Sometimes these teams have to go through multiple reviews (security, operations, code, and change control). What is worse is that there is often a long line to wait in for reviews, causing a review process to slip another week. Many of these reviews are just rubber stamp approvals that could be entirely avoided with some minor modifications to the existing processes.

Companies with legacy processes need to look at how they can modernize processes to be more agile instead of being the reason why their company can't move fast enough.

5. Lack of Operational Maturity

Moving to a DevOps model often requires a different approach to operations. Some companies accustomed to supporting back office applications that change infrequently. It requires a different mindset to support software delivered as a service that is always on, and deployed frequently. With DevOps, operations is no longer just something Ops does. Developers now must have tools so they can support applications. Often I encounter companies that only monitor infrastructure. In the DevOps model developers need access to logging solutions, application performance monitoring (APM) tools, web and mobile analytics, advanced alerting and notification solutions. Processes like change management, problem management, request management, incident management, access management, and many others often need to be modernized to allow for more agility and transparency. With DevOps, operations is a team sport.

Assess your operational processes, tools, and organization and modernize to increase agility and transparency.

6. Outdated testing practices

Too often I see clients who have a separate QA department that is not fully integrated with the development team. The code is thrown over the wall and then testing begins. Bugs are detected and sent back to developers who then have to quickly fix, build, and redeploy. This process is repeated until there is no time remaining and teams are left to agree on what defects they can tolerate and promote to production. This is a death spiral in action. With every release, they introduce more technical debt into the system lowering its quality and reliability, and increasing unplanned work. There is a better way.

The better way is to block bugs from moving forward in the development process. This is accomplished by building automated test harnesses and by automatically failing the build if any of the tests fail. This is what continuous integration is designed for. Testing must be part of the development process, not a handoff that is performed after development. Developers need to play a bigger part in testing and testers need to play a bigger part in development. This strikes fear in some testers and not all testers can make the transition.

Quality is everyone's responsibility, not just the QA team.

7. Automating waste

A very common pattern I run into is the automation of waste. This occurs when a team declares itself a DevOps team or a person declares themselves a DevOps engineer and immediately starts writing hundreds or thousands of lines of Chef or Puppet scripts to automate their existing processes. The problem is that many of the existing processes are bottlenecks and need to be changed. Automating waste is like pouring concrete around unbalanced support beams. It makes bad design permanent.

Automate processes after the bottlenecks are removed.

8. Competing or Misaligned Incentives and Lack of Shared Ownership

This bottleneck has plagued IT for years but is more profound when attempting to be agile. In fact, this issue is at the heart of why DevOps came to be in the first place. Developers are incented for speed to market and operations is incented to ensure security, reliability, availability, and governance. The incentives are conflicting. Instead, everyone should be incented for customer satisfaction, with a high degree of agility, reliability, and quality (which is what DevOps is all about). If every team is not marching towards the same goals, then there will be a never-ending battle of priorities and resources. If all teams' goals are in support of the goals I mentioned above, and everyone is measured in a way that enforces those incentives, then everyone wins --- especially the customer.

Work with HR to help change the reward and incentives to foster the desired behaviors.

9. Dependence on Heroic Efforts

When heroic efforts are necessary to succeed, then a team is in a dark place. This often means working insane hours, being reactive instead of proactive, and being highly reliant on luck and chance. The biggest causes of this are a lack of automation, too much tribal knowledge, immature operational processes, and even poor management. The culture of heroism often leads to burnout, high turnover, and poor customer satisfaction.

If your organization relies on heroes, find out what the root causes are that creates these dependencies and fix them fast.

10. Governance as an Afterthought

When DevOps starts as a grassroots initiative there is typically little attention paid to the question "how does this scale?" It is much easier to show some success in a small isolated team and for an initial project. But once the DevOps initiative starts scaling to larger projects running on way more infrastructures or once it starts spreading to other teams, it can come crashing down without proper governance in place. This is very similar to building software in the cloud. How many times have you seen a small team whip out their credit card and build an amazing solution on AWS? Easy to do, right? Then a year later the costs are spiraling out of control as they lose sight of how many servers are in use and what is running on them. They all have different versions of third party products and libraries on them. Suddenly, it is not so easy anymore.

With DevOps, the same thing can happen without the appropriate controls in place. Many companies start their DevOps journey with a team of innovators and are able to score some major wins. But when they take that model to other teams it all falls down. There are numerous reasons that this happens. Is the organization ready to manage infrastructure and operations across multiple teams? Are there common shared services available like central logging and monitoring solutions or is each team rolling their own? Is there a common security architecture that everyone can adhere to? Can the teams provision their own infrastructure from a self-service portal or are they all dependent on a single queue ticketing system? I could go on but you get the point. It is easier to cut some corners when there is one team to manage but to scale we must look at the entire service catalog. DevOps will not scale without the appropriate level of governance in place.

Assign an owner and start building a plan for scaling DevOps across the organization.

11. Limited to No Executive Sponsorship

The most successful companies have top level support for their DevOps initiative. One of my clients is making a heavy investment in DevOps training and it will run a large number of employees through the program. Companies with top level support make DevOps a priority. They break down barriers, drive organizational change, improve incentive plans, communicate “Why” they are doing Devops, and fund the initiative. When there is no top level support, DevOps becomes much more challenging and often becomes a new silo. Don’t let this stop you from starting a grass roots initiative. Many sponsored initiatives started as grassroots initiatives. These grassroots teams measured their success and pitched their executives. Sometimes when executives see the results and the ROI they become the champions for furthering the cause. My point is, it is hard to get dev and ops to work together with common goals when it is not supported at the highest levels. It is difficult to transform a company to DevOps if it is not supported at the highest levels.

If running a grassroots effort, gather before and after metrics and be prepared to sell and evangelize DevOps upward.