**UNIT-V**

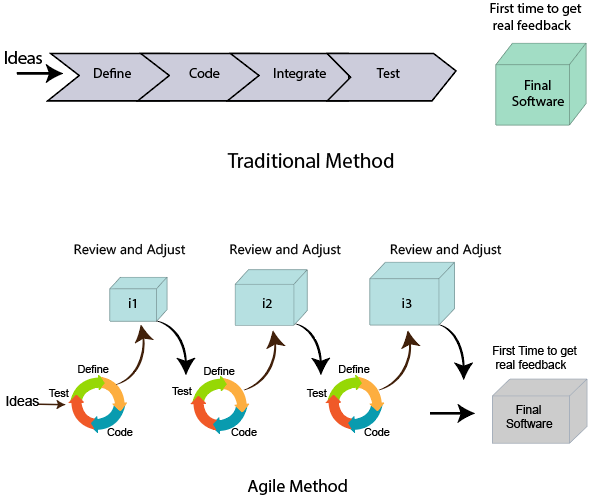
**Agile Methodology and DevOps**

## **Agile Methodology?**

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.



## **Roles in Agile**

There are two different roles in a Agile methodology. These are the Scrum Master and Product Owner.

### 1. Scrum Master

The Scrum Master is a team leader and facility provider who helps the team member to follow agile practices, so that the team member meets their commitments and customers requirements. The scrum master plays the following responsibilities:

* They enable the close co-operation between all the roles and functions.
* They remove all the blocks which occur.
* They safeguard the team from any disturbances.
* They work with the organization to track the progress and processes of the company.
* They ensure that Agile Inspect & Adapt processes are leveraged correctly which includes
  + Planned meetings
  + Daily stand-ups
  + Demo
  + Review
  + Retrospective meetings, and
  + Facilitate team meetings and decision-making process.

### 2. Product Owner

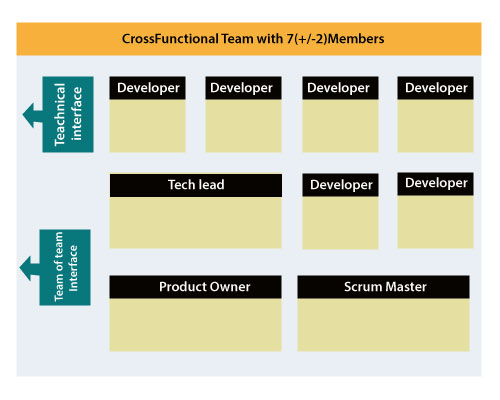
The Product Owner is one who runs the product from a business perspective. The Product Owner plays the following responsibilities:

* He defines the requirements and prioritizes their values.
* He sets the release date and contents.
* He takes an active role in iteration and releasing planning meetings.
* He ensures that the team is working on the most valued requirement.
* He represents the voice of the customer.
* He accepts the user stories that meet the definition of done and defined acceptance criteria.

## **Cross-functional team**

Every agile team contains self-sufficient team with 5 to 9 team members. The average experience of each member ranges from 6 to 10 years. The agile team contains 3 to 4 developers, 1 tester, 1 technical lead, 1 scrum master and 1 product owner.

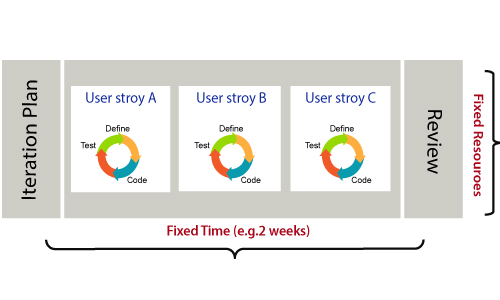
The Scrum master and Product owner are considered as a part of Team Interface, on the other hand remaining members are the part of Technical Interface.



## **How an Agile Team plan their work?**

An Agile methodology is not a specific set of ceremonies or specific development techniques. Rather, it is a group of methodologies that demonstrate a commitment to tight feedback cycles and continuous improvement. An Agile team works in iterations to deliver the customer requirement, and each iteration takes 10 to 15 days. However, the original Agile Manifesto didn't set the time period of two-week iterations or an ideal team size.

Each user requirement is a planned based and their backlog prioritization and size. The team decides, how much scope they have and how many hours available with each team to perform their planed task.



## **What is a user requirement?**

The user requirement defines the requirements of the user in terms of functionalities. There may be of two type of functionality.

* As a <User Role> I want <Functionality> so that <Business Value>
* In order to <Business value> as a <User Role> I want <Functionality>.

During software release planning, a rough estimate is given to a user requirement using relative scale points. During iteration planning, the requirement is broken down into tasks.

# Scrum?

**Scrum is a framework** that helps agile teams to work together. Using it, the team members can deliver and sustain the complex product. It encourages the team to learn through practice, self-organize while working on the problem. Scrum is a work done through the framework and continuously shipping values to customers.

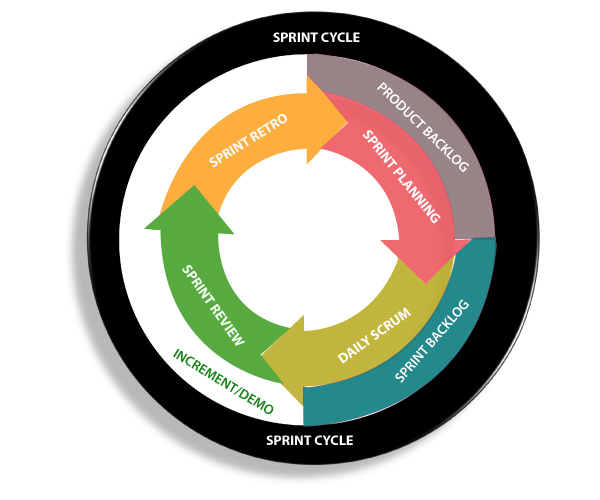
## **The framework**

Scrum and agile are not the same thing because Scrum focused on continuous improvement, which is a core foundation of agile. Scrum framework focuses on ongoing getting work done.

## **What are sprints?**

With scrum, a product is built in a series of repetition called **sprints**. It breaks down big complex projects into bite-size pieces. It makes projects more manageable, allows teams to ship high quality, work faster, and more frequently. The sprints give them more flexibility to adapt to the changes.

Sprints are a short, time-boxed period for Scrum team that works to complete a set amount of work. Sprints are the core component of Scrum and agile methodology. The right sprints will help our agile team to ship better software.



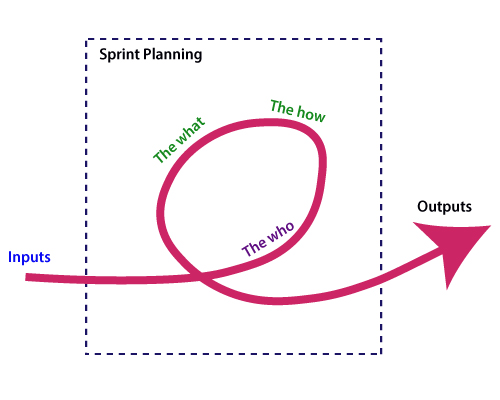
## **What is sprint plan?**

Sprint plan is an action in Scrum that kicks off the sprint. The primary purpose of sprint plan is to define what can deliver in the sprint. It also focuses on how the work will be achieved. It is done in combination with the whole Scrum team members.

The sprint is a set of the period where all the work to be done. Before we start the development, we have to set up the sprint. We need to describe how long time is required to achieve the sprint goal and where we are going to start.

## **Factors affecting Sprint planning**

* **The What:** The product owner describes the goal of the sprint and the backlog items which contribute to achieve that goal.
* **The How:** Agile development team plans its necessary work on how to achieve and deliver the sprint goal.
* **The Who:** The product owner defines the goal based on the value that the customers seek. And the developer needs to understand how they can or cannot deliver that goal.
* **The Inputs:** The product backlog provides the list of input stuff that could potentially be part of the current sprint. The team looks over the existing work done in incremental ways.
* **The Outputs:** The critical outcome of sprint planning is to meet described team goal. The product set the goal of sprint and how they will start working towards the goal.



## **What is the product backlog?**

A product backlog is a registered list of work for the development team. It is driven from the roadmap and its requirements. The essential task is represented at the top of the product backlog so that the team member knows what to deliver first. The developer team doesn't work through the backlog from the product owner's side and product owner doesn't push the work to the developer team. The developer team pulls work from the product backlog.

## **Backlog starts with the two "R"s**

The fundamental product backlog is provided by a team's **roadmap** and **requirements**. Roadmap repetition breaks down into several epics, and each epic will have several requirements and user stories.

The product owner organizes each of the customer stories into a single list. This story is organized for the development team. The product owner chooses to deliver first complete epic.

## **The factors that influence a product owner's prioritization**

* Priority of customer
* Importance of getting feedback
* Relative implementation difficulty
* Symbiotic relationships between work items

When we talk about adopting any new framework or methodology, we think about how we can incorporate these changes into our organization. We simply cannot impose any change in our organization and get started with it, there has to be some process or ways to incorporate that. Also, there are some ways to incorporate Scrum changes within our organization. There include five activities in adopting the Scrum successfully:

1. Awareness
2. Desire
3. Ability
4. Promotion
5. Transfer

So to remember, we call it by the acronym ADAPT.

**Patterns for Adopting Scrum**

### Start Small or Go All In

### Organizations go ahead with it like a Pilot project, like selecting few team members and implementing Scrum with them, It's a ‘Start Small’ pattern. The other approach can be Go All In, which is like the executives are convinced and want the whole organization to implement in one go.

**Reasons to prefer starting small**

* It’s less expensive
* Early success guaranteed
* Avoids risks of going all in
* Less stressful
* Can be done without much change

**Reasons to prefer going all in**

* Reduces resistance
* Avoids problems within different teams
* The All-in transition is Quick!

**Choosing between the two**

As recommended by Mike Cohn, one should always Start Small! It involves less cost, and guaranteed early success. Going all in should be in limited cases, only when it’s a quick need. Also, it involves more cost/money as there are a lot of changes in different departments if required.

### Public Display of Agility or Stealth

The next pattern that comes into the picture is whether to Publicize it or not. We can do the Public Display of Agility. In this approach, the organization announces that it is adopting Scrum. This can vary from announcing it in a meeting room to announcing it through the press release.

The other approach is Stealth transition. In this, only team members know they are using Scrum until the project is complete.

**Reasons for Public Display of Agility**

* Everyone knows that team is doing it and they are more likely to be focussed
* Operating publicly is a firm statement of commitment
* You can solicit organizational support
* It sends a powerful message

**Reasons for Stealth Transition**

* A chance to make progress before resistance starts
* It keeps pressure off
* No one knows until you tell them
* If no one knows, no one can tell you to stop?

**What is the agile iterative approach?**

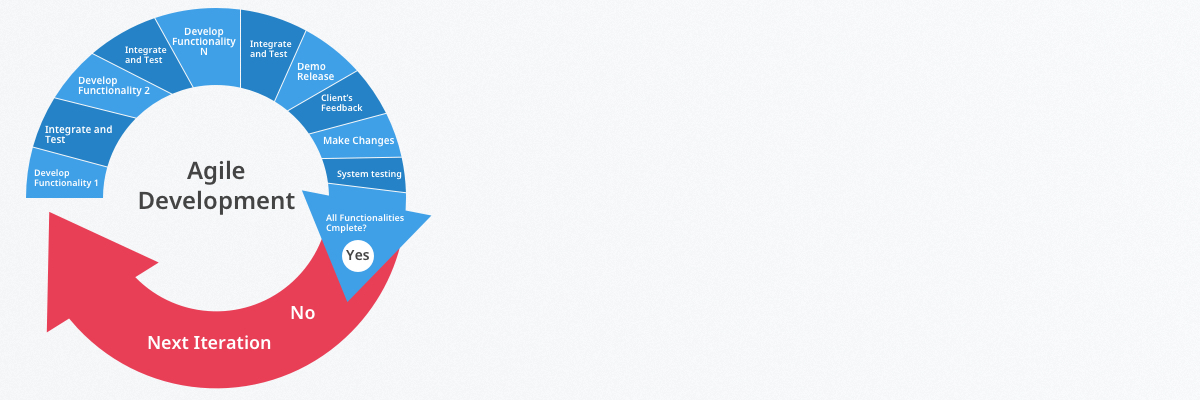
The agile iterative approach focuses on delivering value as fast as possible in increments, rather than all at once. This approach is especially useful in software development and product development. An iterative approach means the software or product development process is split into multiple explicit iterations or versions, each delivering some valuable improvements or additional features.

Iterative methodology allows software developers to adjust, refine, and review software development processes constantly to improve their performance incrementally. The agile iterative approach creates opportunities for constant evaluation and improvement in development processes. The design of an iterative approach is simple and easy to implement, regardless of the context.

Agile iterative development

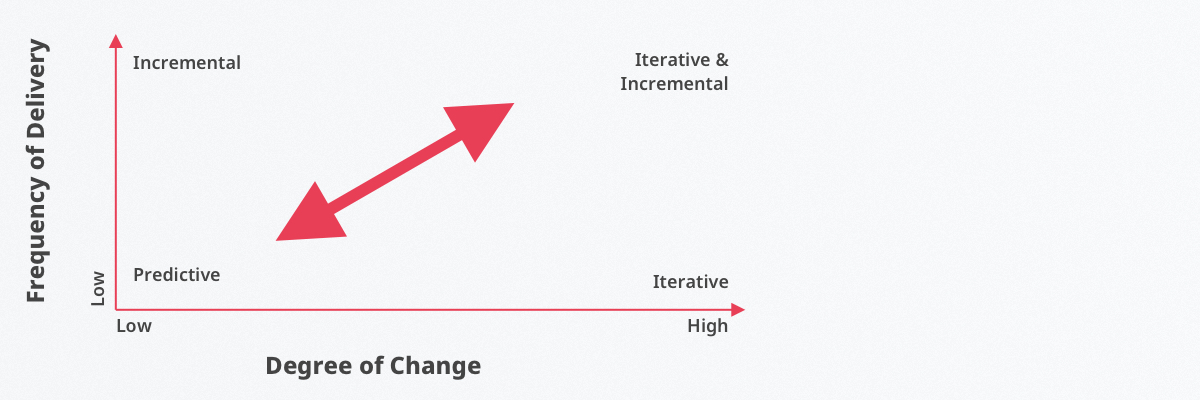
Iterative development breaks down large tasks into smaller pieces that can be repeated, refined, and researched throughout the software development cycle.

Project developers use the most recent development, or iteration, as a base from which to design further products or processes. The iterative development model seeks to advance and refine a product or process in every iteration.



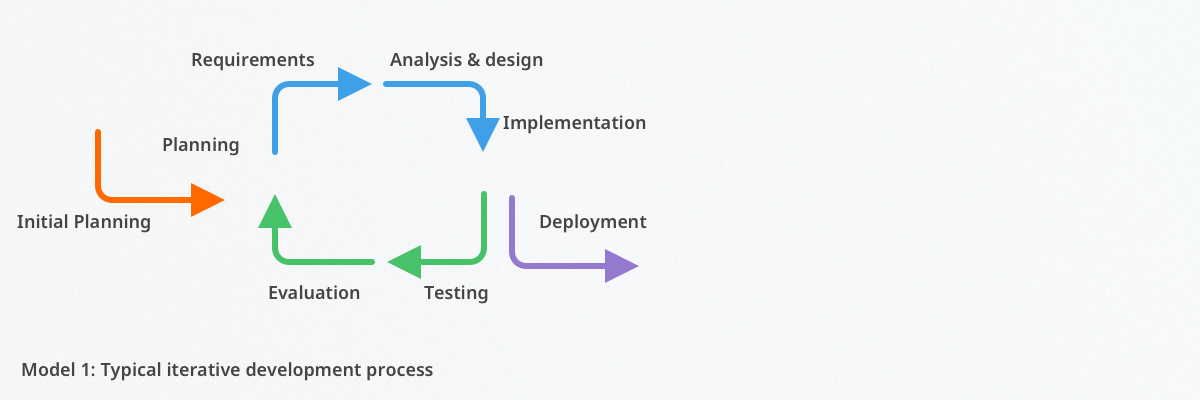
Incremental development is an approach that uses a set number of steps, or increments, that follow a linear path of progression. The steps include conception, analysis, design, testing, release, maintenance, and other increments. The waterfall model is an example of incremental development.

Every sequential increment responds to changes or developments that have already been made. This is essentially the software development equivalent of the factory assembly line. While incremental approaches are less flexible compared to iterative approaches, the two methods are frequently used together in software development projects.



The agile iterative model, or PDCA cycle, is the centrepiece of agile iterative development. Agile iterative software development projects are executed according to the following four step pattern

* P (Plan) – Iteration planning focuses on the planning and discussion of requirements and objectives of a project. During the planning phase, software developers recap finished iterations and discuss anticipated needs moving forward.
* D (Design) – Iteration implementation is concerned with the analysis, design, and implementation of projects. The team develops software during this phase of the cycle. Developers can also test the functionality of the product in the Design phase as well.
* C (Check) – Iteration testing is concerned with ensuring the deliverable meets project requirements. If certain criteria are not met, the team can move backward to the other phases for further improvements.
* A (Adjust) – Iteration evaluation means comprehensively reviewing the work of the iteration or cycle. The software development team will also refine its backlog to prepare for future iterations.



Where is the agile iterative approach employed?

The approach suits constantly evolving software projects for the following reasons:

* Some requirements may evolve during the development process. The agile iterative approach allows teams to modify and enhance different aspects of a deliverable throughout the cycle, especially in the Check and Adjust phases. The cyclical and modifiable nature of the methodology gives development teams a large advantage over more rigid processes.
* While working on a project, a development team can quickly and ‘productively’ adapt to change. The PDCA model of the iterative approach allows for far more flexibility, adaptability, and speed compared to sequential approaches.
* The urgency or risk associated with project components may change. The approach provides a time and space for urgency and risk recognition and alleviation in the early stages of the process.

**Benefits of agile iterative development**

The agile iterative approach provides the following benefits to software development teams:

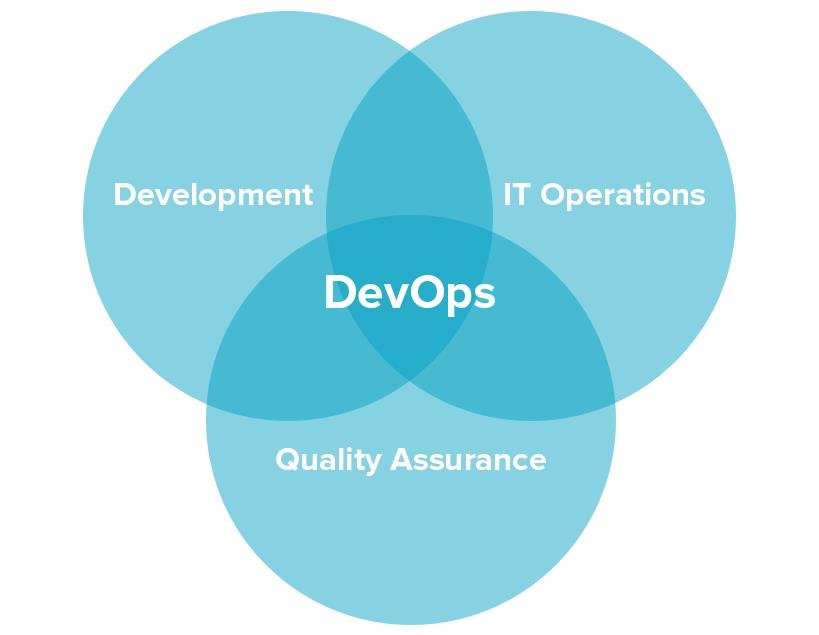
* Flexibility for making changes. The methodology allows for modification throughout the development process.
* Customer involvement. Development teams focus on customer feedback during the Design and Adjust periods of the PDCA iterative cycle.
* Early risk identification and response. Managing each iteration is simpler than managing the whole project at one time. The iterative approach allows development teams to tackle issues early on without requiring the team to backtrack.
* Rapid delivery. The iterative approach requires less time spent on documenting and thus allows development teams to spend more time designing and implementing projects.
* Testing during iteration is easier than testing at the end of the development process. By testing early on, teams can analyze risks and modify their deliverables.
* Enables cognitively-diverse teams to experiment and innovate. The modifiable and cyclical nature of the iterative approach allows teams to test new ideas for their products.

The agile iterative approach allows software development teams to plan, design, check, and adjust iterations. This software development model is a comprehensive and accessible methodology that can benefit agile organizations immensely. Compared to other approaches, the agile iterative approach is more flexible, innovative, quicker, and more modifiable. The approach provides a space for customers to be involved during the development process, so no retroactive changes should need to be made after a deliverable is released.

## **DevOps Methodology?**

DevOps is an information technology (IT) methodology that facilitates collaboration, communication, and integration between software developers and IT operations staff. The primary purpose of DevOps is to improve the quality and speed of software delivery, enabling continuous, frequent updates that deliver value to customers.

The DevOps team works together to create a consistent development, testing and production environment, and automates the development pipeline, to make software delivery efficient, predictable, sustainable and secure.



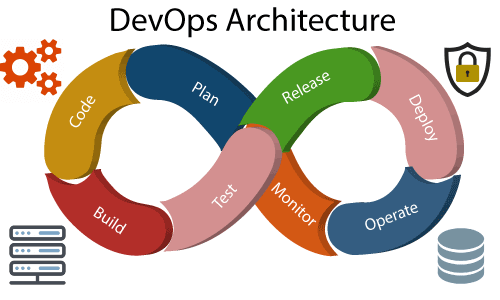
By adopting DevOps companies gain three core advantages that cover technical, business, and cultural aspects of development.

**Higher speed and quality of product releases.** DevOps speeds up product release by introducing [continuous delivery](https://www.altexsoft.com/blog/business/continuous-delivery-and-integration-rapid-updates-by-automating-quality-assurance/), encouraging faster feedback, and allowing developers to fix bugs in the system in the early stages. Practicing DevOps, the team can focus on the quality of the product and automate a number of processes.

**Faster responsiveness to customer needs.**With DevOps, a team can react to change requests from customers faster, adding new and updating existing features. As a result, the time-to-market and value-delivery rates increase.

**Better working environment.** DevOps principles and practices lead to better communication between team members, and increased productivity and agility. Teams that practice DevOps are considered to be more productive and cross-skilled. Members of a DevOps team, both those who develop and those who operate, act in concert

**DevOps Architecture**

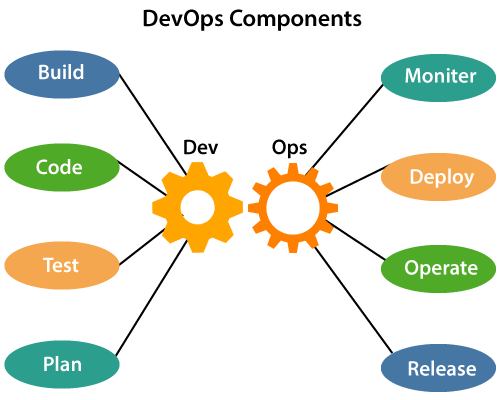


Development and operations both play essential roles in order to deliver applications. The deployment comprises analyzing the **requirements, designing, developing**, and **testing** of the software components or frameworks.

The operation consists of the administrative processes, services, and support for the software. When both the development and operations are combined with collaborating, then the DevOps architecture is the solution to fix the gap between deployment and operation terms; therefore, delivery can be faster.

DevOps architecture is used for the applications hosted on the cloud platform and large distributed applications. Agile Development is used in the DevOps architecture so that integration and delivery can be contiguous. When the development and operations team works separately from each other, then it is time-consuming to **design, test**, and **deploy**. And if the terms are not in sync with each other, then it may cause a delay in the delivery. So DevOps enables the teams to change their shortcomings and increases productivity.

Below are the various components that are used in the DevOps architecture:



1) Build

Without DevOps, the cost of the consumption of the resources was evaluated based on the pre-defined individual usage with fixed hardware allocation. And with DevOps, the usage of cloud, sharing of resources comes into the picture, and the build is dependent upon the user's need, which is a mechanism to control the usage of resources or capacity.

2) Code

Many good practices such as Git enables the code to be used, which ensures writing the code for business, helps to track changes, getting notified about the reason behind the difference in the actual and the expected output, and if necessary reverting to the original code developed. The code can be appropriately arranged in **files, folders**, etc. And they can be reused.

3) Test

The application will be ready for production after testing. In the case of manual testing, it consumes more time in testing and moving the code to the output. The testing can be automated, which decreases the time for testing so that the time to deploy the code to production can be reduced as automating the running of the scripts will remove many manual steps.

4) Plan

DevOps use Agile methodology to plan the development. With the operations and development team in sync, it helps in organizing the work to plan accordingly to increase productivity.

5) Monitor

Continuous monitoring is used to identify any risk of failure. Also, it helps in tracking the system accurately so that the health of the application can be checked. The monitoring becomes more comfortable with services where the log data may get monitored through many third-party tools such as **Splunk**.

6) Deploy

Many systems can support the scheduler for automated deployment. The cloud management platform enables users to capture accurate insights and view the optimization scenario, analytics on trends by the deployment of dashboards.

7) Operate

DevOps changes the way traditional approach of developing and testing separately. The teams operate in a collaborative way where both the teams actively participate throughout the service lifecycle. The operation team interacts with developers, and they come up with a monitoring plan which serves the IT and business requirements.

8) Release

Deployment to an environment can be done by automation. But when the deployment is made to the production environment, it is done by manual triggering. Many processes involved in release management commonly used to do the deployment in the production environment manually to lessen the impact on the customers.

**Agile Vs DevOps**

|  |  |  |
| --- | --- | --- |
|  | **Agile** | **DevOps** |
| **Process** | Communication between the team and customers is continuous, and frequent changes are made to the software to ensure quality. Better suited for complex projects. | Focuses on frequent testing and delivery, but communication is primarily between developers and IT operations. Better suited for end-to-end processes. |
| **Teams** | Allows small teams to complete tasks faster. Agile methods encourage all members to share responsibilities equally instead of assigning specific responsibilities to team members. So, every agile team member should be able to handle or assign any part of the project at any time. | Suitable for large teams. Skill sets are distributed among operations and development team members: Each team member has a specific set of tasks they need to do at each stage of the SDLC. |
| **Focus and feedback** | Typically works in sprints, each sprint lasting less than a month. The idea of a sprint is to complete the project step by step, starting each sprint right after delivery of the previous sprint’s deliverables. | Focuses on operations and business readiness. Most of the feedback comes from internal team members and metrics collected from production environments. Deadlines and goals may recur on a daily basis. |

**Deployment:**

To learn more, read our DevOps resources. **Deployment automation is what enables you to deploy your software to testing and production environments with the push of a button**. Automation is essential to reduce the risk of production deployments.

It's also essential for providing fast feedback on the quality of your software by allowing teams to do comprehensive testing as soon as possible after changes.

An automated deployment process has the following inputs:

* Packages created by the continuous integration (CI) process (these packages should be deployable to any environment, including production).
* Scripts to configure the environment, deploy the packages, and perform a deployment test (sometimes known as a *smoke test*).
* Environment-specific configuration information.

We recommend that you store the scripts and configuration information in version control. Your deployment process should download the packages from an artifact repository (for example, [Artifact Registry](https://cloud.google.com/artifact-registry), [Nexus](https://www.sonatype.com/nexus-repository-sonatype), [Artifactory](https://jfrog.com/artifactory/" \t "external), or your CI tool's built-in repository).

The scripts usually perform the following tasks:

1. Prepare the target environment, perhaps by installing and configuring any necessary software, or by starting up a virtual host from a pre-prepared image in a cloud provider such as Google Cloud.
2. Deploy the packages.
3. Perform any deployment-related tasks such as running database migration scripts.
4. Perform any required configuration.
5. Perform a deployment test to make sure that any necessary external services are reachable, and that the system is functioning.

**Instance of Applications:**

Application of DevOps in the Online Financial Trading Company

The methodology in the process of testing, building, and development was automated in the financial trading company. Using the DevOps, deployment was being done within 45 seconds. These deployments used to take long nights and weekends for the employees. The time of the overall process reduced and the interest of clients increased.

### 2. Use of DevOps in Network cycling

Deployment, testing and rapid designing became ten times faster. It became effortless for the telco service provider to add patches of security every day, which used to be done only every three months. Through deployment and design, the new version of network cycling was being rolled out.

### . Application in Car Manufacturing Industries

Using DevOps, employees helped car manufacturers to catch the error while scaling the production, which was not possible before.

### . Benefits to Airlines Industries

With the benefit of DevOps, United Airlines saved $500,000 by changing to continuous testing standards. It also increased its coverage of code by 85%.

### 5. Application to GM Financial

Regression testing time was reduced by 93%, which in turn reduced the funding period of load by five times.

### 6. Bug Reduction Benefit of DevOps

DevOps has reduced the bugs by up to 35% and in many cases of pre-production bugs up to 40%. By using DevOps, Rabobank was able to provide better quality applications for their clients within less time because it massively reduced the time taken for regression testing.

### 7. Less Time for Integration

Key Bank used DevOps to reduce the time taken for the integration of security and compliance into the process from 3 months to 1 week.

### 8.Decreased Computation Cost and Operation Time

By the use of DevOps, Computation time has been dramatically reduced. In many cases, it has reduced the computing time from up to 60%. When the time taken to complete a task is decreased, then the cost involved the process also decreases.

### 9. Faster Development of Software

The DevOps helps in the faster delivery of apps because it ensures speedier delivery.

### 10. Improvement in Team Collaboration

Transparency is required for better decision-making and works better efficiency of resources. By using DevOps, teams can be more transparent in their work of developing applications and software. There are many big tasks of a project which are broken down into many small tasks that are allotted to different teams or people in the organisation.

# DevOps Pipeline

A pipeline in software engineering team is a set of automated processes which allows DevOps professionals and developer to reliably and efficiently compile, build, and deploy their code to their production compute platforms.

The most common components of a pipeline in DevOps are build automation or continuous integration, test automation, and deployment automation.

A pipeline consists of a set of tools which are classified into the following categories such as:

* Source control
* Build tools
* Containerization
* Configuration management
* Monitoring

### Continuous Integration Pipeline

Continuous integration (CI) is a practice in which developers can check their code into a version-controlled repository several times per day. Automated build pipelines are triggered by these checks which allows fast and easy to locate error detection.

**Some significant benefits of CI are:**

* Small changes are easy to integrate into large codebases.
* More comfortable for other team members to see what you have been working.
* Fewer integration issues allowing rapid code delivery.
* Bugs are identified early, making them easier to fix, resulting in less debugging work.

### Continuous Delivery Pipeline

Continuous delivery (CD) is the process that allows operation engineers and developers to deliver bug fixes, features, and configuration change into production reliably, quickly, and sustainably. Continuous delivery offers the benefits of code delivery pipelines, which are carried out that can be performed on demand.

**Some significant benefits of the CD are:**

* Faster bug fixes and features delivery.
* CD allows the team to work on features and bug fixes in small batches, which means user feedback received much quicker. It reduces the overall time and cost of the project.

## **DevOps Methodology**

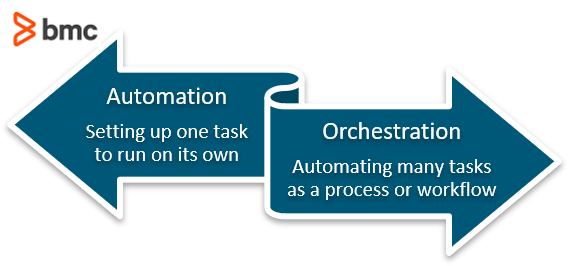
We have a demonstrated methodology that takes an approach to cloud adoption. It accounts for all the factors required for successful approval such as people, process, and technology, resulting in a focus on the following critical consideration:

* **The Teams:** Mission or project and cloud management.
* **Connectivity:** Public, on-premise, and hybrid cloud network access.
* **Automation:** Infrastructure as code, scripting the orchestration and deployment of resources.
* **On-boarding Process:** How the project gets started in the cloud.
* **Project Environment:** TEST, DEV, PROD (identical deployment, testing, and production).
* **Shared Services:** Common capabilities provided by the enterprise.
* **Naming Conventions:** Vital aspect to track resource utilization and billing.
* **Defining Standards Role across the Teams:** Permissions to access resources by job function.

**DevOps orchestration?**

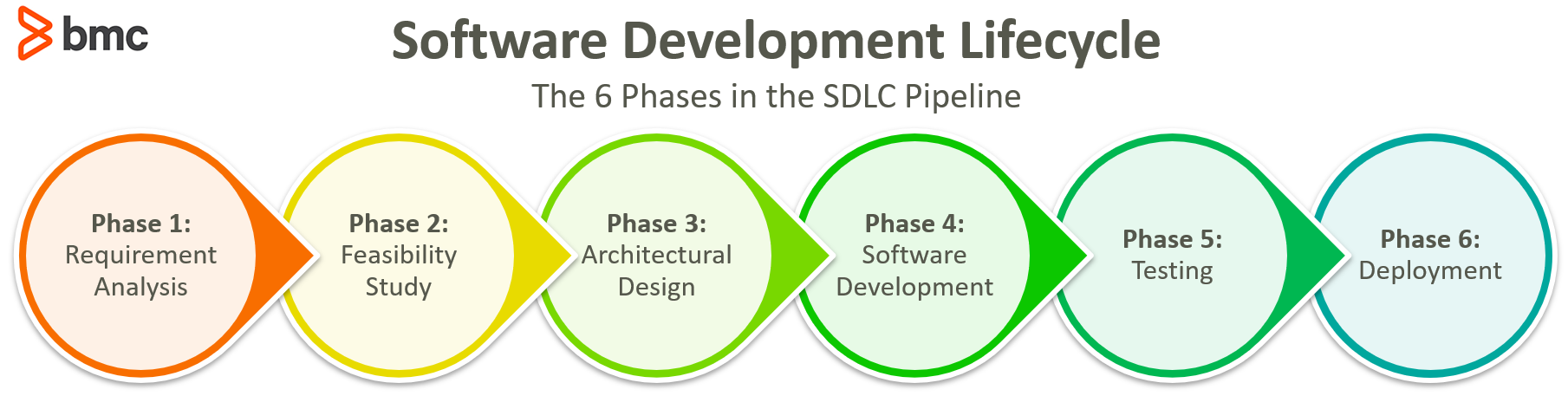
[Orchestration](https://www.bmc.com/blogs/it-orchestration-vs-automation-whats-the-difference/) is the automated coordination of automated activities. It’s easiest to think of it like this:

* Automation covers individual, basic tasks that are programmatically performed through micro-level scripts.
* Orchestration is the large-scale coordination of these automated tasks.



Cloud orchestration involves automating the workflow processes that occur to deliver resources [as a service](https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/). DevOps orchestration, on the other hand, is the coordination of your entire organization’s DevOps practices and the automation tools you employ in pursuit of your goals.

It’s difficult to overstate the importance that automation plays in the successful implementation of DevOps practices. [Software development lifecycle (SDLC)](https://www.bmc.com/blogs/sdlc-software-development-lifecycle/) is a relatively self-explanatory term that simply refers to the loop of software development in the software as a service (SaaS) industry. The SDLC generally revolves around six stages:



Automation drastically speeds up your SDLC, without dimming the quality. The next step, orchestrating your automated tasks, sees more benefits:

* Helps maximize the potential (and ROI) of your automation tools
* Increases the return on your DevOps investment

**Impact of automation & orchestration**

Not every aspect of development can be automated. But heaps of activities can—and should—be automated in search of increasing DevOps optimization and overall speed and reliability of development.

Here are some areas where automation is practically a no-brainer:

**Data centers**

Data centers are a prime location to begin automating routine due to how many recurring tasks tend to take place there. Servers, networks, and databases all require regular maintenance and [security](https://www.bmc.com/blogs/security-vulnerability-vs-threat-vs-risk-whats-difference/) tasks that can be automated to save man hours in the long run while also ensuring fewer things slip through the gaps. Automating your data centers can:

* Improve IT efficiency
* Reduce deployment failures
* Offer more manageable complexity across environments

**Job scheduling & workload automation**

[Job scheduling](https://www.bmc.com/blogs/job-scheduling-vs-workload-automation-whats-difference/) and [workload automation](https://www.bmc.com/blogs/what-is-workload-automation/)can also drastically improve efficiency by simplifying application delivery and managing the data required by batch jobs. Automating application workflows helps to:

* Reduce downtime
* Curb expenses caused by business interruptions
* Enhance scalability thanks to its flexibility that simplifies even the most complex of systems

Overwhelming tasks can be broken into smaller, automatable processes that are tackled programmatically and without the need to keep constant tabs on them.

**IT processes**

The automation of [IT processes](https://www.bmc.com/blogs/practice-vs-process/) improves collaboration by supporting built-in and audited annotation for crystal clear communication. Automation tools also give you the ability to handle Big Data with your existing enterprise skills and best practices. The customization available through automation platforms enables you to setup orchestration across your organization that works in the way you need it to. The deep functionality and standardization available through automation tools and their enterprise-wide user interface makes for more simplified management of even the most complex systems.

These tools also support self-servicing which allows business users the freedom they need to manage workloads and submit queries at any time with a user-friendly and web-based mobile tool that doesn’t require a workload automation expert to leverage.

**Orchestration compounds the benefits**

DevOps orchestration coordinates all of these automated tasks to provide enhanced optimization and oversight. Orchestrating automated IT processes can:

* Accelerate your service delivery speeds
* Lower the cost of service delivery
* Increase the reliability of deployments

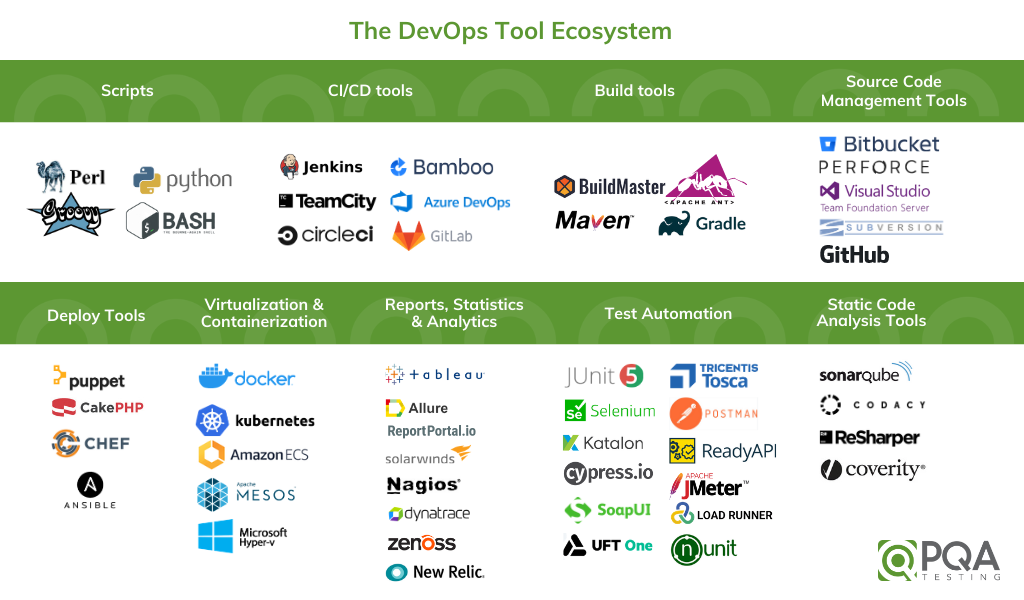
Orchestration tools customized for your organization’s needs can help you make the most of your automation efforts and DevOps implementation.

There is no silver bullet solution or a blueprint that can be followed verbatim that will enhance your enterprise overnight, but DevOps is a mentality that encourages a culture of collaboration as well as a suite of tools that aid collaboration by enhancing communication across the organization.

**DevOps ECO System or Tools in DevOps Implementation**

At its core, the DevOps ecosystem is the idea that tools should be helping you in your journey from requirements to production. In order to help you along your DevOps path, we’ve categorized the different classes of tools out there. With each class, we have included a list of examples of popular solutions. This list is intended to be a starting point to help guide you, as opposed to an exhaustive list of everything available. Feel free to save this graphic as a reminder, and share it with your networks!

Below the graphic, you’ll find helpful descriptions of each of the classes.



It’s important to remember that many tools attempt to provide more than one piece of the pie and there is definitely overlap in the defined classes – in these cases, we have tried to categorize the best fits. These tools play well with others, so combining them can help you produce a streamlined production focused ecosystem.

**Scripts**

Scripts are the connector pieces that bring the disparate tools and classes together into a more coherent whole. Scripts also help with automated and complicated build, deployment, and automation tasks.  Scripting is incredibly flexible and with it, you can do almost anything.



**CI/CD Tools**

Continuous Integration/Delivery Tools are almost the enterprise level package in the DevOps world.  They try to do all of the things. They schedule, gather results, kick-off scripts/processes/tools, integrate with other tools, and report on success. CI/CD tools can often do much of what you need, but as with most generalists, sometimes you have to add another tool that provides specialized capabilities.



**Build Tools**

Build tools are an essential foundational piece of DevOps. With more developers working on a code-base, checking in code, and moving your product/project forward, it’s essential there be a tool that can automate bringing together the right dependencies and generating an installable compiled package.



**Source Code Management Tools (Code Repositories)**

With the distributed nature of teams in the DevOps world, powerful, enabling source code management tools are essential. These tools help to manage multiple simultaneous update streams for the codebase and ensure code coherency through branching and merging mechanisms.



**Deploy Tools /Configuration as Code**

With the levels of automation that are becoming standard in the DevOps ecosystem, it makes sense that there be an automated capability for code to be deployed into your desired environments. It’s necessary for automation to happen, but may also be desired for manual testing or UAT steps. Deployment tools need to be able to integrate with your virtualization solutions as well.



**Virtualization/Containerization**

Gone are the days of setting up a single server to run multiple applications/services. With the concept that each application or service you wish to deploy having a different optimal server configuration grew the idea that every application could run in its own independent container. As this concept grew, it greatly empowered testers and developers who worked best in their own walled environment. Being able to tear down, alter, and redeploy these environments at will has added significant flexibility to DevOps.



**Reports, Statistics, and Analytics**

Many of the tools in these lists have their own integrated reports and reporting capability, but you can add functionality and flexibility by adding your own report engine or plugin. Without a strong capability to monitor the health of your code, servers, interactions, etc the speed and safety that is attainable with DevOps practices will be lost. An integrated monitoring system that can bring you real-time statistics is essential.



**Test Automation**

Test automation is one of the biggest topics in DevOps with numerous threads, intents, and tool systems. The number of tools and their target is large and deserving of its own [article](https://www.pqatesting.com/white-papers/a-quick-matrix-guide-to-popular-automation-frameworks/).).  Any automation solutions/tools that you decide to include in your solution should be capable of integrating with the other tools in your DevOps ecosphere.



**Static Code Analysis Tools**

One of the tools that can help you determine code quality and health is static code analysis. By choosing standards for the way your code should be written and then using a tool to assess this adherence, you can automatically remove a plethora of issues before they ever get put in front of a tester or user.



# 6 Steps to a Successful DevOps Adoption

If you’re considering a move to a DevOps delivery model, here are six approaches I’ve found to be critical for ensuring a successful DevOps adoption within an organization.

**1. Embrace a DevOps Mindset**

DevOps doesn’t begin by just stating, “Let’s do DevOps,” and jumping in with tools. Your entire organization needs to have a clear understanding of what DevOps is and what specific business needs it can address, and, most importantly, everyone needs to be willing to change the way things have always been done.

One way to begin this process is to identify your current application value streams—the series of activities necessary for moving your products from development all the way into production. Understanding where in this delivery process there are constraints, bottlenecks, and wait queues will allow you to which activities you should concentrate on improving.

Organizations often equate D evOps with automation. While automation can help accelerate manual processes, DevOps is fundamentally about collaboration and communication. If you don’t embrace strong communication and collaborative practices among everyone in the software development, testing, delivery, and operational process, automating your processes will not yield the business benefits you desire.

**2. Make the Most of Metrics**

Production failure rate: how often the software fails in production during a fixed period of time

Mean time to recover: how long it takes an application in production to recover from a failure

Average lead time: how long it takes for a new requirement to be built, tested, delivered, and deployed into production

Deployment speed: how fast you can deploy a new version of an application to a particular environment (integration, test, staging, preproduction, or production environments)

Deployment frequency: how often you deploy new release candidates to test, staging, preproduction, and production environments

Mean time to production: how long it takes when new code is committed into a code repository for it to be deployed to production

Once you’ve determined the metrics you wish to collect and have a baseline of where you currently stand, set goals for each metric so the team knows what to strive for.

**3. Understand and Address Your Unique Needs**

Contrary to what those selling DevOps products will tell you, there is no “one size fits all” solution for DevOps. You cannot just drop in an automated tool or hire a self-proclaimed “DevOps engineer” and expect success. Every organization will have a unique DevOps journey tied to its specific business and culture, and that journey will be far more about changing people’s habits and communication patterns than what tools help you enable automation.

**4. Adopt Iteratively**

Generally, you should look at tackling your biggest value stream constraints first, as doing so will have the largest business impact. Some of these constraints will be easy to resolve, while others will take a significant amount of time to correct—and often a whole lot of convincing others to change.

If you’re beginning your DevOps journey from inside a software development organization, consider starting from the beginning of your delivery process and moving toward production. Properly implementing branch management strategies and build automation is key to fast feedback that will enable efficient downstream processes in the future.

**5. Emphasize Quality Assurance Early**

Based on my observations in most organizations, testers get the least amount of time to do quality assurance, and eventually, the quality of the product suffers. Organizations that struggle with DevOps often focus their efforts on automating deployments, overlooking the needs of QA.

While it is impossible to automate all your testing in DevOps, it is critical to automate all tests run as part of your continuous integration process (unit tests, static code analysis, etc.), as well as regression testing and smoke testing performed on each environment within your delivery process. Automating at least some functional testing and nonfunctional tests associated with security, performance, and other quality characteristics can often be achieved to speed up these activities.

**6. Take a Smart Approach to Automation**

Automation is the cornerstone of accelerating your delivery processes, and everything—infrastructure, environment, configuration, platform, build, test, process, etc.—should be defined and written in code. If something is time-intensive, broken, or prone to error, start automating there first. This will quickly benefit your team by reducing delivery times, increasing repeatability, and eliminating configuration drift.

**Agile Capability**

In a world of challenges, constant innovation, turbulence and uncertainty Agile Capability helps organizations of all sizes to deliver customer centric value sooner in a way which is more certain and predictable than traditional ways of working.

Agile Capability is an associate network of Agile experts who operate as an Agile transformation service. We bring Reduced Time to Change through proven real world experience of delivering multiple Agile transformations around the planet. Whereas we tend to serve fortune 500 organizations in the banking and wealth sector we’ve also delivered for manufacturers, telecoms and industrial Iot clients.

Agile Capability is also the creator of the ACT tool, a platform that helps organizations visualize their Agile progression, create organizational coaching and transformation plans, while providing tools that can improve their practices and augment their transformation strategy.