

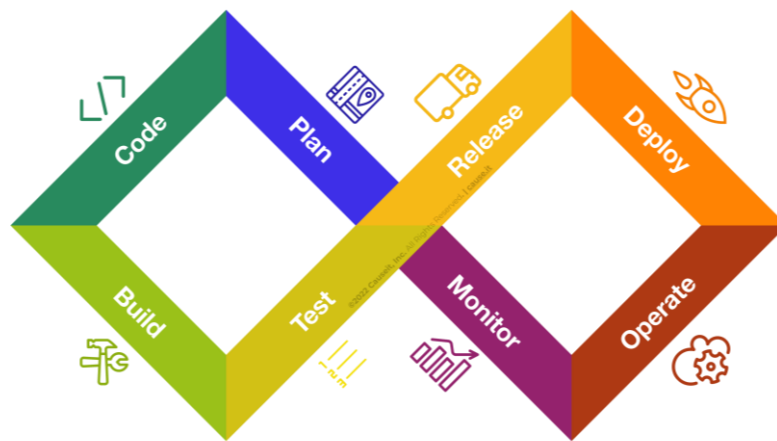
Unit 1 & 2: Web Development Framework

What is DevOps?

If you want to build better software faster, DevOps is the answer. Here's how this software development methodology brings everyone to the table to create secure code quickly.

Overview

DevOps combines development and operations to increase the efficiency, speed, and security of software development and delivery compared to traditional processes. A more nimble software development lifecycle results in a competitive advantage for businesses and their customers.



DevOps is a continuous cycle, often depicted as an infinity symbol, where the 'final' stage connects to the 'first', to emphasize the continuing nature of the process.

The stages in DevOps are:

- Plan
- Code & Assemble
- Build
- Test
- Release
- Deploy
- Operate
- Monitor

There are variations on this approach, sometimes naming 'release' and 'deploy' continuous integration & continuous deployment (CI/CD). At its core, the cycle is intended to manage a series of handoffs between different responsibilities in a more fluid, iterative way.

DevOps explained

DevOps can be best explained as people working together to conceive, build and deliver secure software at top speed. DevOps practices enable software development (dev) and operations (ops) teams to accelerate delivery through automation, collaboration, fast feedback, and iterative improvement. Stemming from an Agile approach to software development, a DevOps process expands on the cross-functional approach of building and shipping applications in a faster and more iterative manner. In adopting a DevOps development process, you are making a decision to improve the flow and value delivery of your application by encouraging a more collaborative environment at all stages of the development cycle. DevOps represents a change in mindset for IT culture. In building on top of Agile, lean practices, and systems theory, DevOps focuses on incremental development and rapid delivery of software. Success relies on the ability to create a culture of accountability, improved collaboration, empathy, and joint responsibility for business outcomes.

DevOps is a combination of software development (dev) and operations (ops). It is defined as a software engineering methodology which aims to integrate the work of development teams and operations teams by facilitating a culture of collaboration and shared responsibility.

Core DevOps principles

The DevOps methodology comprises four key principles that guide the effectiveness and efficiency of application development and deployment. These principles, listed below, center on the best aspects of modern software development.

- **Automation of the software development lifecycle.** This includes automating testing, builds, releases, the provisioning of development environments, and other manual tasks that can slow down or introduce human error into the software delivery process.
- **Collaboration and communication.** A good DevOps team has automation, but a great DevOps team also has effective collaboration and communication.
- **Continuous improvement and minimization of waste.** From automating repetitive tasks to watching performance metrics for ways to reduce release times or mean-time-to-recovery, high performing DevOps teams are regularly looking for areas that could be improved.
- **Hyperfocus on user needs with short feedback loops.** Through automation, improved communication and collaboration, and continuous improvement, DevOps teams can take a moment and focus on what real users really want, and how to give it to them.

By adopting these principles, organizations can improve code quality, achieve a faster time to market, and engage in better application planning.

The four phases of DevOps

As DevOps has evolved, so has its complexity. This complexity is driven by two factors:

- Organizations are moving from monolithic architectures to microservices architectures. As DevOps matures, organizations need more and more DevOps tools per project.
- The result of more projects and more tools per project has been an exponential increase in the number of project-tool integrations. This necessitated a change in the way organizations adopted DevOps tools.

This evolution took place in following four phases:

Phase 1: Bring Your Own DevOps

In the Bring Your Own DevOps phase, each team selected its own tools. This approach caused problems when teams attempted to work together because they were not familiar with the tools of other teams.

Phase 2: Best-in-class DevOps

To address the challenges of using disparate tools, organizations moved to the second phase, Best-in-class DevOps. In this phase, organizations standardized on the same set of tools, with one preferred tool for each stage of the DevOps lifecycle. It helped teams collaborate with one another, but the problem then became moving software changes through the tools for each stage.

Phase 3: Do-it-yourself DevOps

To remedy this problem, organizations adopted do-it-yourself (DIY) DevOps, building on top of and between their tools. They performed a lot of custom work to integrate their DevOps point solutions together. However, since these tools were developed independently without integration in mind, they never fit quite right. For many organizations, maintaining DIY DevOps was a significant effort and resulted in higher costs, with engineers maintaining tooling integration rather than working on their core software product.

Phase 4: DevOps Platform

A single-application platform approach improves the team experience and business efficiency. A DevOps platform replaces DIY DevOps, allowing visibility throughout and control over all stages of the DevOps lifecycle.

By empowering all teams – Development, Operations, IT, Security, and Business – to collaboratively plan, build, secure, and deploy software across an end-to-end unified system, a DevOps platform represents a fundamental step-change in realizing the full potential of DevOps.

What is the goal of DevOps?

DevOps represents a change in mindset for IT culture. In building on top of Agile practices, DevOps focuses on incremental development and rapid delivery of software. Success relies on the ability to create a culture of accountability, improved collaboration, empathy, and joint responsibility for business outcomes. Adopting a DevOps strategy enables businesses to increase operational efficiencies, deliver better products faster, and reduce security and compliance risk.

The DevOps lifecycle and how DevOps works

The DevOps lifecycle stretches from the beginning of software development through to delivery, maintenance, and security. The stages of the DevOps lifecycle are:

- **Plan:** Organize the work that needs to be done, prioritize it, and track its completion.
- **Create:** Write, design, develop and securely manage code and project data with your team.
- **Verify:** Ensure that your code works correctly and adheres to your quality standards — ideally with automated testing.
- **Package:** Package your applications and dependencies, manage containers, and build artifacts.
- **Secure:** Check for vulnerabilities through static and dynamic tests, fuzz testing, and dependency scanning.
- **Release:** Deploy the software to end users.
- **Configure:** Manage and configure the infrastructure required to support your applications.
- **Monitor:** Track performance metrics and errors to help reduce the severity and frequency of incidents.
- **Govern:** Manage security vulnerabilities, policies, and compliance across your organization.

What is a DevOps engineer?

A DevOps engineer is responsible for all aspects of the software development lifecycle, including communicating critical information to the business and customers. Adhering to DevOps methodologies and principles, they efficiently integrate development processes into workflows, introduce automation where possible, and test and analyze code. They build, evaluate, deploy, and update tools and platforms (including IT infrastructure if necessary). DevOps engineers manage releases, as well as identify and help resolve technical issues for software users.

DevOps tools, concepts and fundamentals

DevOps covers a wide range of practices across the application lifecycle. Teams often start with one or more of these practices in their journey to DevOps success.

Topic	Description
Version control	The fundamental practice of tracking and managing every change made to source code and other files. Version control is closely related to source code management.
Agile	Agile development means taking iterative, incremental, and lean approaches to streamline and accelerate the delivery of projects.
Continuous Integration (CI)	The practice of regularly integrating all code changes into the main branch, automatically testing each change, and automatically kicking off a build.
Continuous Delivery (CD)	Continuous delivery works in conjunction with continuous integration to automate the infrastructure provisioning and application release process. They are commonly referred to together as CI/CD .
Shift left	A term for shifting security and testing much earlier in the development process. Doing this can help speed up development while simultaneously improving code quality.

How does DevSecOps relate to DevOps?

Security has become an integral part of the software development lifecycle, with much of the security shifting left in the development process. DevSecOps ensures that DevOps teams understand the security and compliance requirements from the very beginning of application creation and can properly protect the integrity of the software.

By integrating security seamlessly into DevOps workflows, organizations gain the visibility and control necessary to meet complex security demands, including vulnerability reporting and auditing. Security teams can ensure that policies are being enforced throughout development and deployment, including critical testing phases. DevSecOps can be implemented across an array of environments such as on-

premises, cloud-native, and hybrid, ensuring maximum control over the entire software development lifecycle.

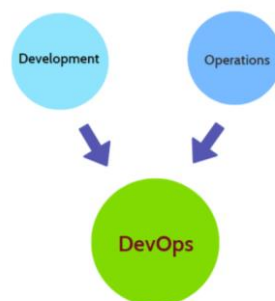
How are DevOps and CI/CD related?

CI/CD — the combination of continuous integration and continuous delivery — is an essential part of DevOps and any modern software development practice. A purpose-built CI/CD platform can maximize development time by improving an organization's productivity, increasing efficiency, and streamlining workflows through built-in automation, continuous testing, and collaboration. As applications grow larger, the features of CI/CD can help decrease development complexity. Adopting other DevOps practices — like shifting left on security and creating tighter feedback loops — helps break down development silos, scale safely, and get the most out of CI/CD.

DevOps Architecture

In Software Engineering, Development and Operations are vital in delivering applications. The development comprises analyzing the requirements, designing, developing, and testing software components or frameworks. The operation consists of administrative processes, services, and support for the software. When both Development and Operation collaborate, they combine to form the picture of DevOps architecture. Moreover, it can be gathered that DevOps architecture is the solution to mend the gap between Development and Operations teams so that the delivery can be faster with fewer issues.

DevOps Architecture and Components



Teams use DevOps architecture for hosting applications on cloud platforms and developing large distributed applications. Agile Development is used here so that integration and delivery can be continuous. When the Development and Operations team work separately from each other, it is time-consuming to design, test and deploy. Also, if the teams are not in sync with each other, it may cause a delay in delivery. So DevOps enables the teams to amend their shortcomings and increase productivity.

Below are the various DevOps components

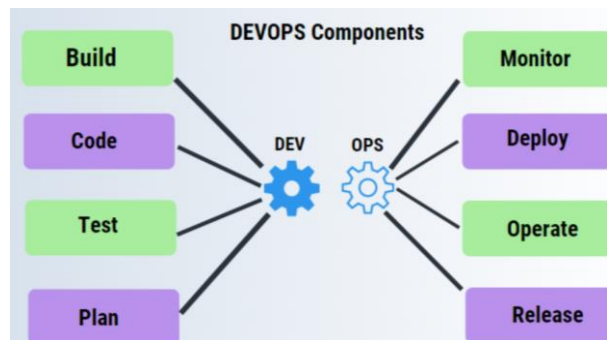
1. Build

The evaluation of the cost of resource consumption was based on pre-defined individual usage with fixed hardware allocation before the adoption of DevOps. But with DevOps, the use of the cloud and sharing of resources comes into the picture,

and the build depends upon the user's need which is a mechanism to control the usage of resources or capacity.

2. Code

Many good practices like widely used git enable the code to be used, which ensures not only writing the code for business but also helps to track changes, getting notified about the reason behind the difference, and, if necessary, reverting to the original code developed. The code can be appropriately arranged in files and folders etc., and they can be reused.



3. Test

The application will move to production after it is tested. In the case of Manual Testing, it consumes more time in testing and pushing the code to display. Automating the testing process reduces the time required for testing, thereby decreasing the time to deploy the code to production. Running automated scripts eliminates many manual steps, making the testing process more efficient.

4. Plan

DevOps use agile methodology to plan development. Unplanned work always reduces productivity. With the Development and Operations teams in sync, it helps organize the work to plan accordingly to increase productivity.

5. Monitor

Continuous monitoring is used to identify any risks of failure. It is also helpful in tracking the system accurately so that the application's health can be checked. The monitoring becomes easier with services where the log data may be monitored through third-party tools like Splunk.

6. Deploy

Most systems can support the scheduler for automated deployment. A cloud management platform enables users to capture accurate insights and view the optimization scenario and analytics on trends through the deployment of dashboards.

7. Operate

DevOps changes the way the traditional approach of developing and testing separately. The teams operate collaboratively, participating actively throughout the

service lifecycle. The operations team interacts with developers and devises a monitoring plan for IT and business requirements.

8. Release

Generally, deployment to an environment can be done by automation. However, the deployment to the production environment is triggered manually. Most of the processes involved in release management commonly specify manually deploying in the production environment to lessen the impact on the customers.

Features of DevOps Architecture

Below are the key features of DevOps Architecture.

1. Automation

Automation most effectively reduces time consumption, specifically during the testing and deployment phase. Since automated tests are executed more rigorously, they increase productivity, accelerate releases, and reduce issues. This will lead to catching bugs sooner so they can be fixed more quickly. The team uses automated tests, cloud-based services, and builds for continuous delivery to implement each code change. This promotes production using automated deploys.

2. Collaboration

The Development and Operations team collaborates as a DevOps team, improving the cultural model as the teams become more effective with their productivity, strengthening accountability and ownership. The teams share their responsibilities and work closely in sync, which in turn makes the deployment to production faster.

3. Integration

To integrate applications with other components in the environment, the integration phase involves integrating the existing code with new functionality, followed by testing. Continuous integration and testing enable continuous development. The frequency of the releases and micro-services leads to significant operational challenges. To overcome such challenges, continuous integration and delivery are implemented to deliver quicker, safer, and more reliable.

4. Configuration Management

This ensures that the application only interacts with the resources concerned with the environment in which it runs. The configuration files separate the external configuration from the source code by creating them outside the application. The configuration file can be written during deployment or loaded at the run time, depending on the environment it is running.

Waterfall Model

The waterfall methodology is a project management approach that emphasizes a linear progression from beginning to end of a project. This methodology, often used by engineers, is front-loaded to rely on careful planning, detailed documentation, and consecutive execution. The Waterfall methodology — also known as the Waterfall

model — is a sequential development process that flows like a waterfall through all phases of a project (analysis, design, development, and testing, for example), with each phase completely wrapping up before the next phase begins. It is said that the Waterfall methodology follows the adage to “measure twice, cut once.” The success of the Waterfall method depends on the amount and quality of the work done on the front end, documenting everything in advance, including the user interface, user stories, and all the features’ variations and outcomes.

With the majority of the research done upfront, estimates of the time needed for each requirement are more accurate, and this can provide a more predictable release date. With a Waterfall project, if parameters change along the way, it’s harder to change course than it is with Agile methodology.

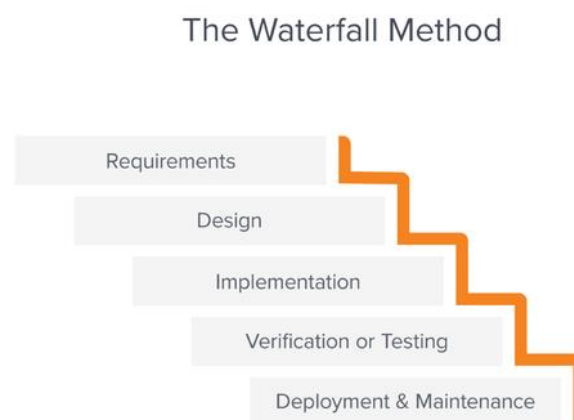
What is Waterfall software?

Waterfall software helps project managers handle the task. As Waterfalls are a relatively complex, phased approach, they require close attention and coordination. Waterfall software can be desktop or cloud-based. It helps you:

- Structure your processes
- Organize tasks
- Set up Gantt charts and schedules
- Monitor project progress

5 common stages in a Waterfall process.

The Waterfall methodology follows a chronological process and works based on fixed dates, requirements, and outcomes. With this method, the individual execution teams aren’t required to be in constant communication and, unless specific integrations are required, are usually self-contained. Team members also tend to work independently and aren’t expected to provide status reports as often as with the Agile approach. Usually, one phase doesn’t begin until the previous one is finished.



Using a software development project as an example, the Waterfall process usually includes stages that look like this:

- **Requirements:** The Waterfall methodology depends on the belief that all project requirements can be gathered and understood upfront. The project

manager does their best to get a detailed understanding of the project sponsor's requirements. Written requirements, usually contained in a single document, are used to describe each stage of the project, including the costs, assumptions, risks, dependencies, success metrics, and timelines for completion.

- **Design:** Here, software developers design a technical solution to the problems set out by the product requirements, including scenarios, layouts, and data models. First, a higher-level or logical design is created that describes the purpose and scope of the project, the general traffic flow of each component, and the integration points. Once this is complete, it is transformed into a physical design using specific hardware and software technologies.
- **Implementation:** Once the design is complete, technical implementation starts. This might be the shortest phase of the Waterfall process because painstaking research and design have already been done. In this phase, programmers code applications based on project requirements and specifications, with some testing and implementation taking place as well. If significant changes are required during this stage, this may mean going back to the design phase.
- **Verification or testing:** Before a product can be released to customers, testing needs to be done to ensure the product has no errors and all of the requirements have been completed, ensuring a good user experience with the software. The testing team will turn to the design documents, personas, and user case scenarios supplied by the product manager to create their test cases.
- **Deployment and maintenance:** Once the software has been deployed in the market or released to customers, the maintenance phase begins. As defects are found and change requests come in from users, a team will be assigned to take care of updates and release new versions of the software.

Advantages of the Waterfall methodology.

The Waterfall methodology is a straightforward, well-defined project management methodology with a proven track record. Since the requirements are clearly laid out from the beginning, each contributor knows what must be done when, and they can effectively plan their time for the duration of the project.

Other benefits of the Waterfall method include:

- Developers can catch design errors during the analysis and design stages, helping them to avoid writing faulty code during the implementation phase.
- The total cost of the project can be accurately estimated, as can the timeline, after the requirements have been defined.
- With the structured approach, it is easier to measure progress according to clearly defined milestones.
- Developers who join the project in progress can easily get up to speed because everything they need to know should be in the requirements document.
- Customers aren't always adding new requirements to the project, delaying production.

Disadvantages of the Waterfall methodology.

Like any development process, the strengths in one area might mean weaknesses in the other. The Waterfall methodology's insistence on upfront project planning and commitment to a certain defined progress means that it is less flexible, or agile, later in the game. Changes that come further in the process can be time-consuming, painful, and costly.

Other reasons the Waterfall methodology may not work include:

- Projects can take longer to deliver with this chronological approach than with an iterative one, such as the Agile method.
- Clients often don't fully know what they want at the front end, opening the door to requests for changes and new features later in the process when they're harder to accommodate.
- Clients are not involved in the design and implementation stages.
- Deadline creep — when one phase in the process is delayed, all the other phases are delayed.

Who uses the Waterfall model?

The Waterfall process is adopted by project managers who are faced with development projects that:

- Don't have ambiguous requirements.
- Offer a clear picture of how things will proceed from the outset.
- Have clients who seem unlikely to change the scope of the project once it is underway.
- If a project manager prefers clearly defined processes, where cost, design, and time requirements are known upfront, then the Waterfall method is the way to go, as long as the project itself is conducive to those constraints.

Agile Software Development

Agile is defined as an iterative software development approach where value is provided to users in small increments rather than through a single large launch. Agile teams evaluate requirements and results continuously, which leads to the efficient implementation of change. Agile software development methodologies often called Agile, preach flexibility and pragmatism in the application delivery process. This iterative software development approach delivers value to users in small increments rather than through a single large launch. Agile teams evaluate requirements and results continuously, which leads to the efficient implementation of change.

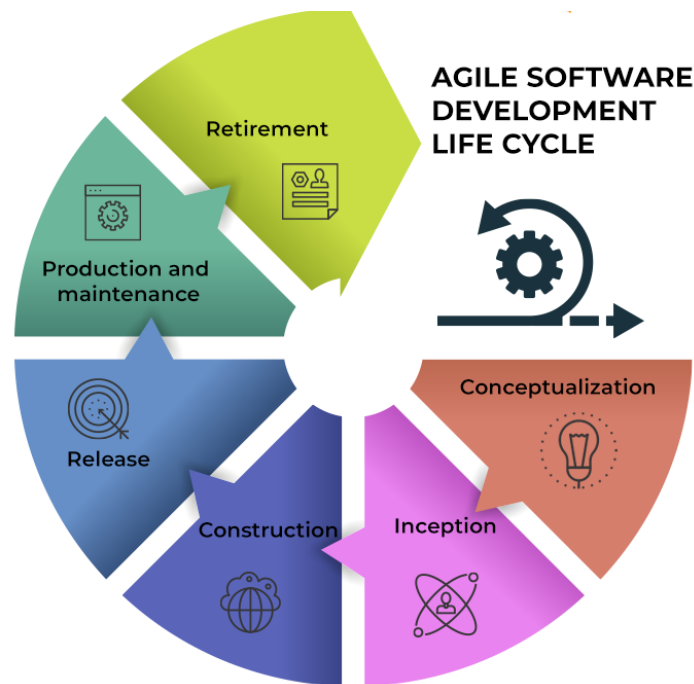
Agile also mandates testing throughout the development cycle. This allows teams to make changes whenever required, alert each other of potential problems, and consequently gives them the confidence to create and release high-quality applications. The core values of Agile are embodied in the Agile Manifesto, which was created by a group of software development personnel in 2001. This manifesto outlines four key concepts that encourage lightweight development, outlined below:

- **Prioritize people over tools & processes.** While the latter is undoubtedly essential, meaningful individual interactions are a vital driver of the software development process and help create an effective response to business needs.
- **Well-built application comes before in-depth documentation.** Agile doesn't completely do away with documentation but focuses on giving the development team only the information they need to meet their goal, such as user stories.
- **Replace contract negotiations between the project manager and client with frequent collaboration.** The product can take shape according to the customer's vision more effectively if they're involved throughout the development process rather than just at the start and end.
- **Respond to change swiftly & effectively.** Agile dismisses the notion of change as an undesired expense. Instead, it values change and encourages short iterations to allow for modifications to be made quickly and easily.

Apart from these core values, **the Agile Manifesto** outlines **12 principles** for development teams to improve their functioning:

1. Divide large tasks into smaller pieces for quick completion
2. Focus on customer satisfaction through the speedy and continuous delivery of value
3. Ensure the creation of processes that drive sustainable efforts
4. Accept changing requirements, even if introduced at a later phase in project
5. Welcome change as a means to achieve a competitive advantage
6. Give motivated team members the work environment and trust required to complete requirements quickly.
7. Acknowledge that self-organized teams deliver the best work
8. Measure progress based on work completed
9. Complete work at a consistent pace
10. Ensure regular collaboration between project and business teams throughout the project duration
11. Regularly reflect on how one can adjust team behavior to enhance effectiveness
12. Finally, constantly strive for excellence.

The Agile life cycle sees developers strategically move the application from conceptualization to retirement.



1. Conceptualization

In the first step of the Agile life cycle, the product owner defines the project scope. In the case of multiple projects, the critical ones are prioritized. Depending on the organization's structure, personnel may be assigned to more than one project at once.

This stage sees the product owner and the client discuss essential requirements and formulate basic documentation based on the finalized project goals. This documentation, perhaps in the form of a product requirements document (PRD), will include the proposed aim of the project and supported features. The time and cost of the project are also estimated at this stage.

The in-depth analysis carried out during conceptualization helps determine feasibility before work starts. Developers can aim to complete only the most critical requirements as one can add more in later stages.

2. Inception

Once the project is conceptualized, the next step is building the software development team. In this stage, the product owner checks the availability of team members and assigns the best available ones to the project. The product owner is responsible for giving these team members the required resources.

Once the team is set, it will begin the design process by creating a mock-up of the user interface and, perhaps, a few user flow and UML diagrams. The project architecture is also built at this stage. The designed elements are then shown to the stakeholders for further input.

All this lets the team fully establish the requirements in the design and figure out application functionality and how it will all fit into the existing system. Frequent check-ins by the business team will ensure that inception stays on track.

3. Construction

The construction phase, known as the iteration phase, is where most work happens. This is usually the longest phase, with the dev team and the UX designers collaborating closely to bring together the requirements and feedback and interpret the design into code.

The construction goal is to create the application's basic functionality before the first iteration (or 'sprint', as described below) ends. Additional secondary features and minor modifications can occur in future iterations. The main goal is to swiftly create a working application and implement improvements for client satisfaction.

4. Release

When the team enters this stage, the product should be nearly ready to release. However, before this can happen, the QA team must test the application and make sure it is fully functional according to the decided project goals. Testing also takes place to ensure that no bugs and defects exist in the code; if any are found, they must be reported swiftly and fixed by the dev team. Clean code is a cornerstone of this stage.

This phase also includes user training, the creation of the system, and user documentation to support it. Visualizing the code is helpful here. Once all the defects are ironed out, and user training is completed, the final iteration of the product can be taken live and released into production.

5. Production and maintenance

Once the application is released successfully and made available to end users, the team moves into maintenance mode. This phase sees the dev team providing continuous support to ensure smooth system operations and quash any newly found bugs.

The team will also be on call to offer additional training to customers and resolve post-live queries to ensure that the product is used as intended. Developers can also use the feedback collected during this stage to plan the features and upgrades for the next iterations.

6. Retirement

The application may be slated for retirement for two reasons: replacement with a new version or the lack of a use case due to redundancy or obsolescence.

If an application enters this phase, the first step is to notify users of the impending retirement of the software. Next, one must ensure a smooth migration to the new system. Finally, the dev team must complete all the pending end-of-life activities and cease the support provided to the existing application.

Sprint planning in Agile

Each Agile phase outlined above leads to the creation of numerous software iterations. These iterations are created as the dev team repeats its processes to refine the

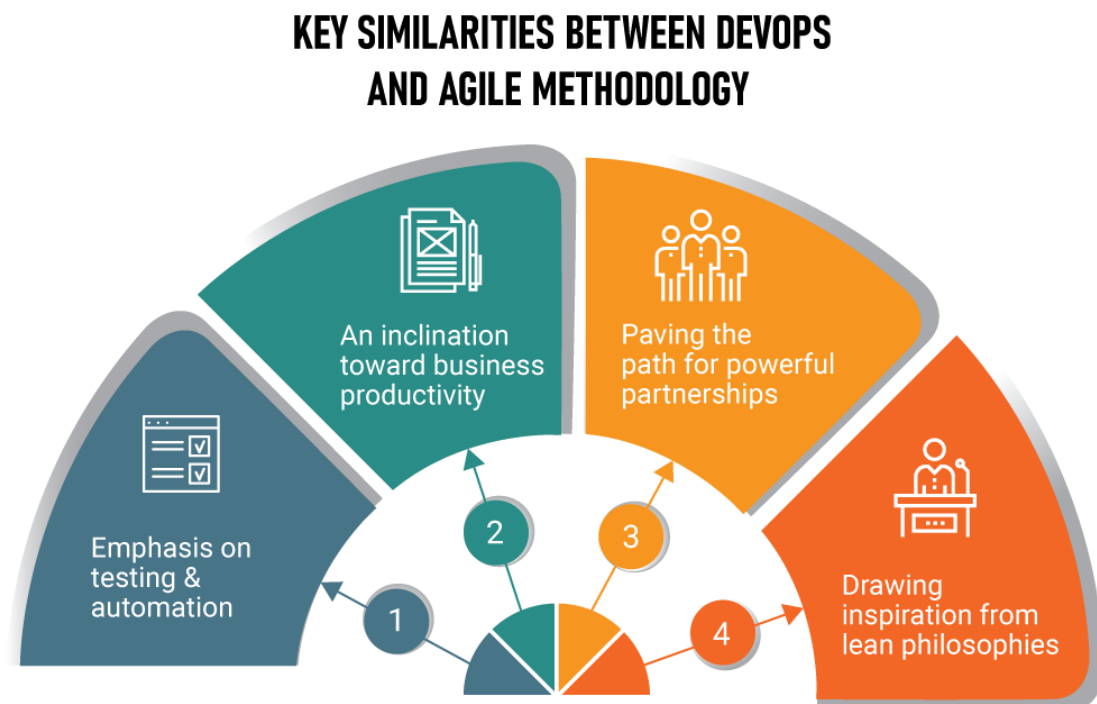
application and create the best possible version according to the determined project requirements. These iterations are 'sub-cycles' contained within the larger Agile software development life cycle.

The Agile life cycle divides work into 'sprints' to complete these iterations. The goal of each sprint is to produce a working application. A typical sprint should last for 10 business days (2 weeks). Outlined below is the typical sprint workflow:










- **Planning:** Each sprint starts with a 'sprint planning meeting', in which team members get together and decide which objectives will be addressed through the upcoming work round. During this meeting, the product manager must prioritize work from the task backlog and assign tasks to specific members.
- **Development:** Once the plan is in place, the team works to design and develop the application according to existing guidelines.
- **QA testing:** After the application is developed, the QA team tests it thoroughly, drives the correction of any errors or shortcomings, and documents the results.
- **Delivery:** Post testing, the application is ready for deployment and is presented to all relevant stakeholders and customers.
- **Assessment:** After delivery, feedback is collected from customers and combined with other relevant information for implementation during the next sprint.

More Detail about Agile: - <https://www.spiceworks.com/tech/devops/articles/what-is-agile-software-development/>

Key Similarities between DevOps and Agile



Key Differences between DevOps and Agile Methodology (Agile VS DevOps)

 DevOps	Parameter	 Agile Methodology
Emphasis on collaboration & productivity	Philosophy 	Emphasis on incremental changes through iterative development & testing
Continuous testing & integration, providing end-to-end business solutions	Focal Point & Purpose 	Incremental deployments in complex projects
Looks after secure deployment	Delivery & Deployment 	Looks after developing & launching software
Large team with different skill-set	Team Size & Skills 	Smaller team with advanced skill-sets
Extensive documentation	Documentation 	Light on documentation
Received via customers	Feedback 	Received internally
Specs & design documents	Communication 	Daily scrum meetings

More Details: - <https://www.spiceworks.com/tech/devops/articles/devops-vs-agile/>

DevOps Automation tools and their purposes

DevOps is a technique that merges IT operations and software development teams to establish a process that enhances collaboration and production. It is based on Agile methodologies. Despite the numerous definitions of DevOps, there is universal agreement that automation and continuity are crucial parts of DevOps. DevOps solutions are widely available and can help team members adopt automation and continuity most effectively. DevOps automation tools are used to automate software development processes while also focusing on lifecycle, distribution, and monitoring equipment, among other things.

What are DevOps Automation Tools?

What is DevOps automation? DevOps automation uses technology to augment processes that enable feedback loops among operations and development teams to hasten the distribution of iterative changes to applications in production. Automation tools in DevOps improve speed, steadiness, correctness, dependency, and delivery rate. DevOps automation covers all aspects of development, deployment, and monitoring.

Why Use DevOps Automation Tools?

DevOps automation tools make it simpler by introducing a new flow throughout SDLC and addressing essential perspectives of your DevOps by automating the entire system chain with Create, Check, Publish, and Release capabilities.

- Speeded Up Development
- improved operational effectiveness.
- More rapid release
- Continuous Supply
- Continuous Deployment
- Greater Collaboration and Faster Recover Time
- Greater Innovation Speed
- Unbroken Flow Across the Value Chain

Best Automation Tools for DevOps

1. **Splunk:** Real-time DevOps monitoring can improve the delivery of applications. Thanks to Splunk software, you can deliver better apps faster with more business impact. Splunk provides real insights across all phases of the delivery life cycle, as opposed to competing solutions focusing on single-release components.
2. **Maven:** Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation. It simplifies the build process like ANT. But it is too much advanced than ANT.
3. **Jira:** Jira Software provides planning and tracking tools so teams can manage dependencies, feature requirements, and stakeholders from day one. CI/CD integrations facilitate transparency throughout the software development life cycle. Jira gives users the option to create a project roadmap or a project plan

for continuous, iterative projects like software development. Start by establishing project tasks in Jira before constructing your strategy. Every project's tasks form its core.

4. **Ansible:** Ansible is a cross-platform, open-source resource provisioning automation tool that DevOps professionals frequently utilize to accelerate the delivery of software development using an "infrastructure as code" strategy. The build automation tools in DevOps are widely used, and Ansible is one of them. Ansible makes DevOps simpler by automating the integration of internally created applications into your production systems. Ansible is the most well-liked DevOps tool for orchestrating, configuring, automating and managing IT infrastructure.
5. **Docker:** The continuous phases of the DevOps environment include development, integration, testing, deployment, and monitoring. The Continuous Deployment step of the DevOps ecosystem uses Docker, which is essential to the ecosystem. Developers control what is inside a container with Docker (application, service, and dependencies to frameworks and components) and how the containers and services interact to form an application composed of several services.
6. **Jenkins:** The name of a DevOps integration tool is Jenkins. Jenkins is different from continuous integration as it is designed for internal use along with plugin additions. Jenkins is a free and open-source Java-based CI server that may run on Windows, mac, and different Unix-based OS. Jenkins can be established on cloud-based platforms as well. Jenkins is a crucial tool as it supports CI and CD, two essential DevOps concepts. The roughly 1,500 plugins available to enable integration points for providing custom functionality during software development make Jenkins compatible with most CI/CD integration tools and services.
7. **Terraform:** HashiCorp's Terraform is an open-source DevOps solution that enables you to construct, maintain, and describe infrastructure using language understandable to humans. Declarative programming and Terraform simplify automating and administering your organization's platform's infrastructure and services. Terraform offers developers a safe, effective environment to build and modify infrastructure. Terraform provides a single procedure for all clouds, which makes it ideal for multi-cloud deployments.
8. **Git:** Git is a DevOps tool for managing source code. It is a version of a control system that can efficiently manage little to massive projects. Git is a tool in DevOps that is used to make some changes to the source code, allowing many engineers to engage in inconsistent development. Since it is interoperable with most engagements, along with SSH, HTTP, and FTP, the Git DevOps tool is easy to use. In contrast to the majority of centered version control systems, it offers the best benefit for inconsistent shared-repository development projects. This makes it a good value for software that is mission critical.
9. **Kubernetes:** The most well-known open-source container orchestration technology, Kubernetes, automates containerized applications' management, deployment, scaling, networking, and reliability. Kubernetes divides the compartments that make up an application into units for simple sorting and

discovery. Building, implementing, and growing enterprise-grade DevOps pipelines is made possible by Kubernetes' broad range of capabilities and capabilities. Teams can also use it to automate the manual processes related to orchestration. This kind of automation would benefit any organization wanting to boost performance and productivity.

10. **Gradle:** An automation tool for building that supports multi-language development is called Gradle. It is a handy DevOps tool for those wishing to develop, test, and deploy software across several platforms. Gradle provides a versatile build architecture that may help developers at every phase of the development process, from code generation and packaging to online publication of the finished product. With Java, C/C++, and Groovy, Gradle is compatible. Furthermore, Google favors using it to create Android applications.

DevOps Automation Tools Key Features

DevOps automation tools have some key features which make them very useful. Here we will discuss the key elements of automation tools used in DevOps.

1. CI/CD

CI/CD is a technique for frequently providing applications to clients by creating automation for the different app development steps. Continuous integration, delivery, and deployment are the three core CI/CD concepts. CI/CD solves the issues various teams may experience when combining new code.

In particular, CI/CD brings continuous monitoring and continual automation throughout the whole lifespan of an app, from the testing and integration phases to the delivery and deployment stages. The different teams collaborate agilely using a DevOps or site reliability engineering (SRE) strategy. These interconnected processes are called a "CI/CD pipeline."

2. Performance Monitoring

The technique of process monitoring everything, from strategy to development, integration to testing, and deployment to operations, is known as DevOps. It provides a complete, up-to-the-minute view of the condition of the infrastructure, services, and applications in use. The most critical application and service monitoring elements are real-time streaming, historical replay, and visualization.

Teams can react promptly and automatically to changes in the client experience thanks to DevOps monitoring. Additionally, it allows developers to switch back to earlier stages of development, lowering the number of failed production improvements. More excellent software instrumentation can identify and resolve issues manually or automatically, depending on the situation.

3. Reporting System

DevOps automation solutions enable software developers to streamline their development lifecycle by automating the application deployment to the server. The automating DevOps automation testing tools also serve as a sound reporting system making it easier for engineers to analyse their work.

Purpose and Introduction of Maven

Maven is a powerful build automation tool primarily used for Java projects, though it's capable of managing projects in various programming languages. Its purpose revolves around simplifying and streamlining the build process, managing dependencies, and providing uniform project structures.

Purpose:

- **Dependency Management:** Maven handles project dependencies, automatically fetching libraries and external dependencies required for a project, making it easier to manage and update them.
- **Build Automation:** It automates the build process, allowing developers to compile, test, package, and deploy projects using predefined plugins and configurations.
- **Project Lifecycle Management:** Maven defines a standard project structure and lifecycle, making it easier for developers to understand and navigate different phases of a project, such as compile, test, package, install, and deploy.
- **Consistency:** Maven enforces project structure conventions, ensuring consistency across projects. This consistency simplifies collaboration and makes it easier for new developers to understand and contribute to projects.

Introduction:

In an introductory context, when starting with Maven, it's crucial to understand its key components:

- **POM (Project Object Model):** Maven uses XML-based POM files to define project configurations, dependencies, build profiles, and more. The POM serves as the project's backbone, defining its structure and settings.
- **Repositories:** Maven uses repositories to store project dependencies. There are local, central, and remote repositories where dependencies are fetched from or deployed to.
- **Plugins:** Maven employs plugins to execute specific tasks during the build process, such as compiling code, running tests, packaging artifacts, and more. Plugins extend Maven's functionalities.
- **Convention over Configuration:** Maven follows a convention-based approach, reducing the need for extensive configuration. However, it allows customization through configuration when necessary.

To start using Maven, you typically install it, define a project by creating a POM file, specify project details, dependencies, and configure plugins. Then, using Maven commands (**mvn**), you can perform various tasks like compiling code, running tests, and packaging the application. Maven's simplicity and consistency make it a popular choice among developers for managing and building projects.

Purpose and introduction of Ansible

Ansible is an open-source automation platform used for configuration management, application deployment, and task automation. Its primary purpose is to simplify complex IT tasks, making them more manageable, repeatable, and scalable.

Purpose:

- **Configuration Management:** Ansible helps in defining and maintaining the desired state of systems, ensuring consistency across servers, networks, and infrastructure elements. It uses "playbooks" to describe configurations and tasks that need to be applied to various systems.
- **Automation:** It automates repetitive tasks, allowing administrators and developers to focus on more critical aspects of their work. Ansible performs tasks such as software installation, system updates, and configuration changes across multiple servers simultaneously.
- **Orchestration:** It orchestrates complex workflows involving multiple systems. For instance, Ansible can manage the deployment of applications across development, testing, and production environments in a consistent and controlled manner.
- **Simplicity and Agentless Operation:** Ansible employs a simple syntax (using YAML) and operates in an agentless manner, meaning it doesn't require software to be installed on managed systems. This simplicity facilitates quicker adoption and reduces overhead.

Introduction:

- **Inventory:** Ansible operates based on an inventory, a file or set of files that contain information about the systems it manages. These systems can be organized into groups and managed collectively using playbooks.
- **Playbooks:** Playbooks are Ansible's configuration, deployment, and orchestration language. They are written in YAML format and describe a set of tasks to be executed on remote systems defined in the inventory.
- **Modules:** Ansible utilizes modules to perform specific tasks on managed nodes. These modules are small programs that Ansible calls to execute tasks like installing packages, managing users, copying files, etc.
- **Idempotency:** Ansible encourages idempotent actions, meaning performing an action multiple times produces the same result as performing it once. This helps maintain system consistency and reliability.

Getting started with Ansible involves setting up an inventory file containing details of servers or systems to manage, creating playbooks to define tasks and configurations, and executing these playbooks using the **ansible-playbook** command. Ansible's simplicity, ease of use, and broad range of functionalities make it a popular choice for automating and managing IT infrastructure across various environments.

To learn in more detail: -

Maven: - <https://www.javatpoint.com/maven-tutorial>

Jira: - <https://confluence.atlassian.com/jira> &
<https://www.atlassian.com/software/jira/guides/getting-started/who-uses-jira#for-agile-teams>

Splunk: - <https://www.javatpoint.com/splunk>

Ansible: - <https://www.tutorialspoint.com/ansible/index.htm>