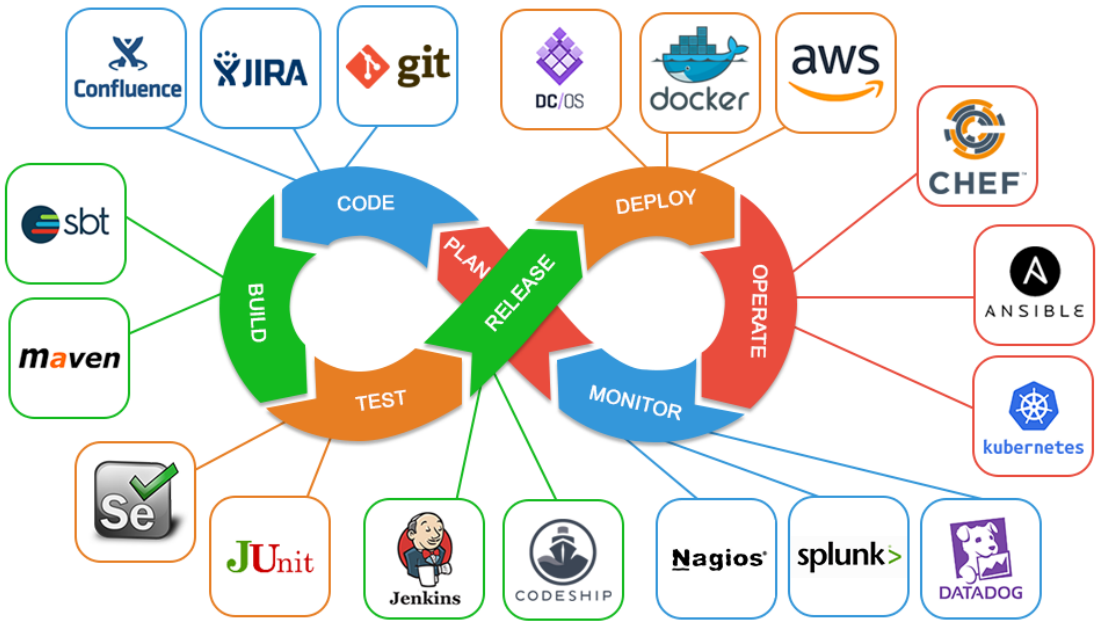
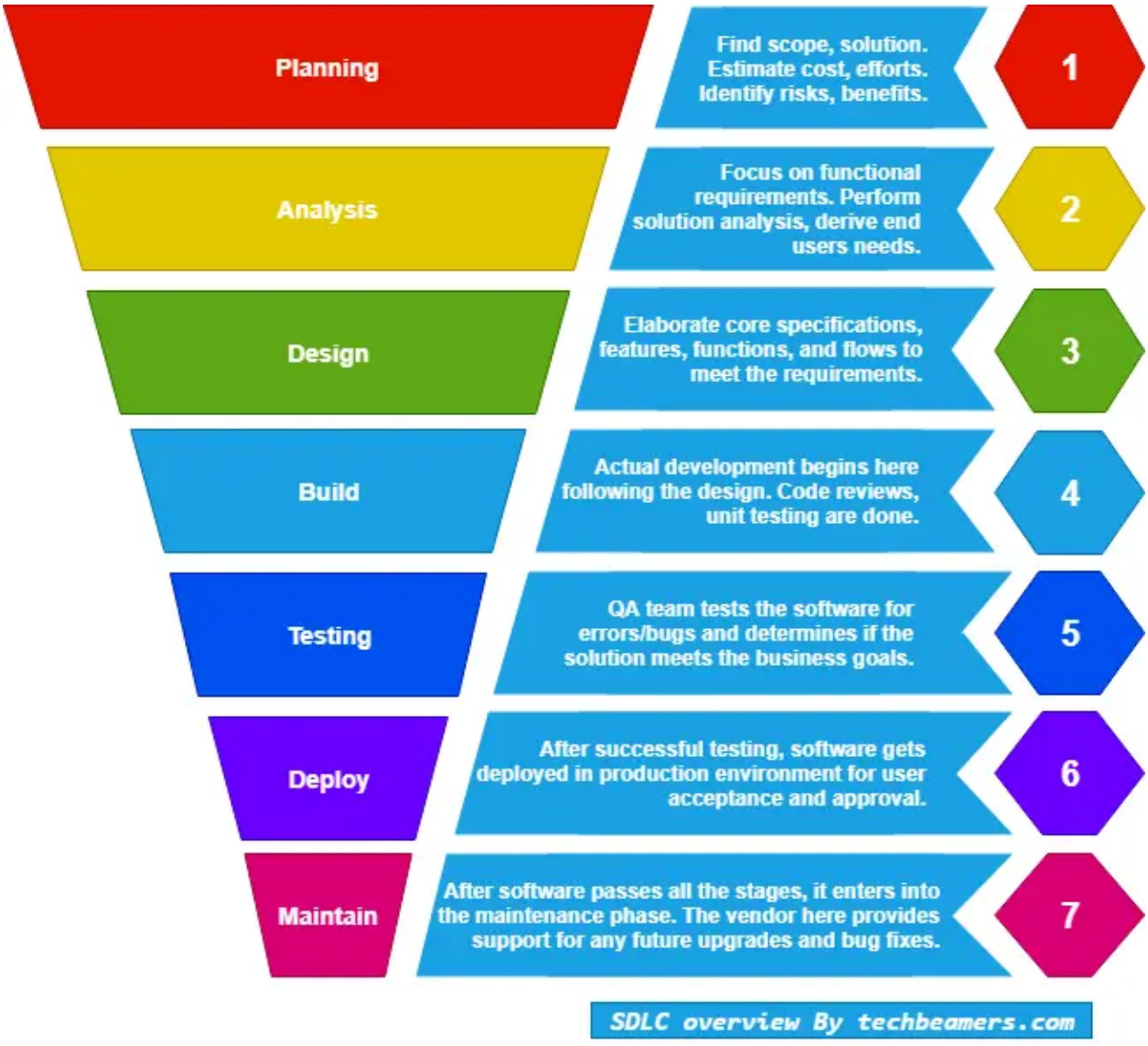
BASICS OF DEVOPS



Software Development Life Cycle (SDLC):



The Software Development Life Cycle (SDLC) is a systematic process for planning, creating, testing, deploying, and maintaining software applications or systems. It provides a structured framework to guide the development process from the initial concept to the final product. The SDLC encompasses various phases, each with its own set of activities, goals, and deliverables.

Here are the typical phases of the SDLC:

Requirements Gathering and Analysis:
Understand the project's goals, scope, and objectives.Gather and document user

requirements, both functional and non-functional.Identify constraints, risks, and potential challenges.

Planning:
Define project scope, objectives, and deliverables.Create a detailed project plan with timelines, tasks, and resources.Allocate budgets, human resources, and technology resources.

Design:
Design the software architecture, including system components and their interactions.Create detailed technical specifications for various components.Design the user interface (UI) and user experience (UX) elements.

Implementation (Coding):
Write code according to the design specifications.Follow coding standards and best practices.Conduct peer code reviews for quality assurance.

Testing:
Develop and execute test cases to validate the functionality of the software.Perform unit testing, integration testing, system testing, and user acceptance testing.Identify and fix bugs, issues, and defects.

Deployment:
Prepare the software for production release.Configure servers, databases, and other necessary resources.Deploy the software to the target environment.

Maintenance and Support:
Provide ongoing support, monitoring, and maintenance for the software.Address any issues, bugs, or performance concerns that arise.Release updates, patches, and enhancements as needed.It's important to note that the SDLC is not a rigid linear process. Different methodologies and approaches, such as Waterfall, Agile, Scrum, Kanban, and DevOps, offer variations in how these phases are executed.

For example:

Waterfall: Emphasizes a sequential approach with distinct phases, where each phase is completed before moving to the next.

Agile: Focuses on iterative development and continuous feedback. It involves frequent releases and collaboration between teams.

Scrum: A specific Agile framework that involves time-boxed iterations (sprints) and a structured set of roles and ceremonies.

Kanban: Visualizes the workflow and emphasizes continuous delivery and incremental improvements.

DevOps: Integrates development and operations teams to automate the deployment pipeline and streamline the software release process.

Each methodology has its strengths and weaknesses, and the choice depends on factors like project complexity, team dynamics, customer requirements, and organizational culture. Regardless of the methodology, the SDLC provides a structured approach to manage software development projects and deliver high-quality software products.

Drawbacks of waterfall methodology:

The Waterfall methodology, while historically significant in software development, has several limitations and challenges that have led to the emergence of more flexible and iterative approaches like Agile.

Here are some of the key problems associated with the Waterfall method:

Rigidity and Lack of Flexibility: Waterfall follows a strict sequential process, where each phase must be completed before the next one starts. This can make it difficult to

accommodate changes or new requirements that emerge during the project. Changes often require revisiting earlier phases, causing delays and additional costs.

Limited Customer Involvement: In Waterfall, customer feedback is typically collected at the beginning and end of the project. This lack of continuous customer involvement can lead to misalignment between the delivered product and the customer's evolving needs.

Late Discoveries of Issues: Testing and quality assurance are often conducted in the later stages of Waterfall projects. This means that issues, defects, or misunderstandings might only be identified after a substantial amount of development work has been completed, leading to costly rework.

Inaccurate Estimations: Due to the sequential nature of Waterfall, accurately estimating project timelines and costs can be challenging. Requirements and scope might evolve over time, making initial estimates less reliable.

Limited Adaptability to Changes: Changes to requirements or scope can be difficult to accommodate in Waterfall. If a significant change occurs, it may require restarting the project from the beginning or managing it as a separate project.

Lack of Transparency: Waterfall projects can lack transparency, especially for stakeholders who are not directly involved in development activities. This can lead to misunderstandings and a lack of visibility into the project's progress.

Long Time to Deliver Value: Since the final product is typically delivered at the end of the project, it can take a long time before customers start seeing any value. This may not align well with today's fast-paced business environments.

Risk Management Challenges: Risk assessment and management are often delayed until later phases in Waterfall projects. This can lead to issues being discovered late in the project, when addressing them might be more difficult and costly.

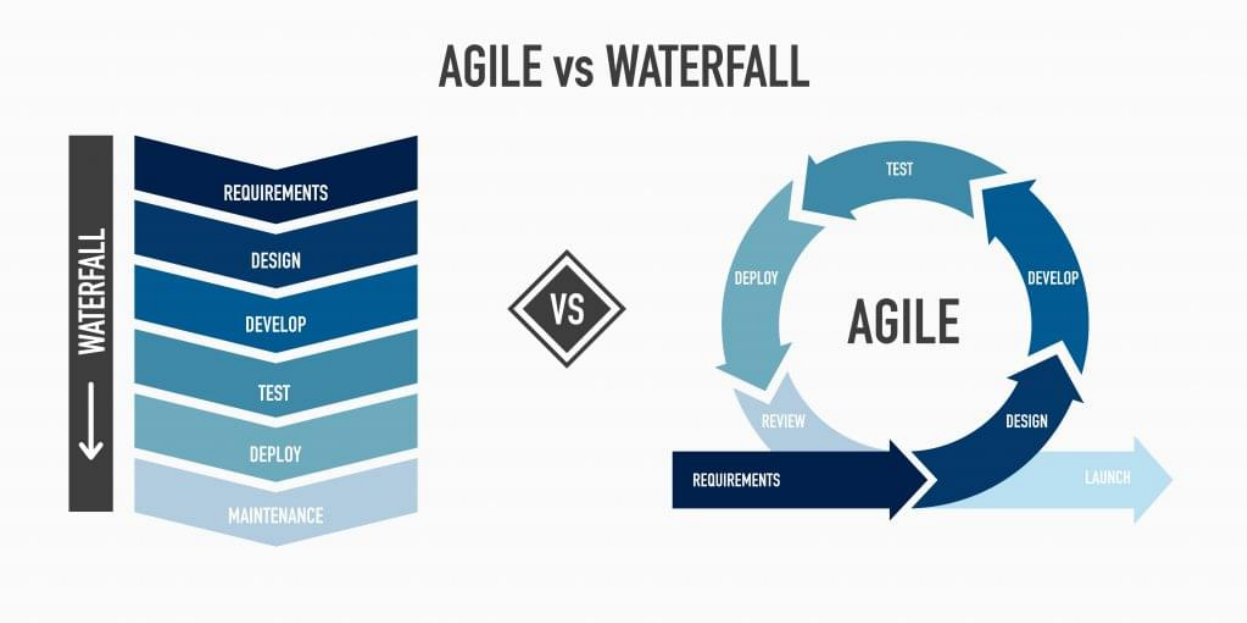
Limited Collaboration: Waterfall tends to emphasize individual roles and responsibilities

within the different phases. This can hinder collaboration and cross-functional teamwork.

Documentation Overload: Waterfall relies heavily on detailed upfront documentation, which can lead to excessive paperwork and documentation that may become outdated or not entirely relevant as the project progresses.

Delayed Feedback Loops: Due to the linear nature of Waterfall, feedback loops from testing, users, or stakeholders can be delayed until later in the process, making it harder to make course corrections.

Agile methodology and drawbacks:



Agile methodology is an iterative and incremental approach to software development that focuses on collaboration, flexibility, and delivering customer value. While Agile has brought many benefits to software development, it also has certain drawbacks and challenges.

Here are some of the drawbacks of Agile methodology:

Lack of Predictability: Agile projects often have variable schedules and evolving scope, making it challenging to predict when specific features will be completed. This can be problematic for projects with strict deadlines or fixed timelines.

Scope Creep: The flexibility of Agile can lead to scope creep, where new requirements are continuously added during the development process, potentially affecting project timelines and budgets.

Resource Intensive: Agile requires frequent communication, collaboration, and involvement from team members, which can be resource-intensive and may not be sustainable for all teams, especially in organizations with limited resources.

Documentation Challenges: Agile emphasizes working software over comprehensive documentation. While this is beneficial for quick delivery, it can lead to insufficient documentation for maintenance and future development.

Dependency on Team Collaboration: Agile relies heavily on team collaboration and communication. If team members are not aligned or communication breaks down, it can lead to inefficiencies and misunderstandings.

Customer Availability: Agile requires active involvement and availability of customers or product owners for regular feedback. If customers are not available, it can slow down decision-making and progress.

Lack of Comprehensive Design: Agile places more emphasis on delivering functionality than on comprehensive upfront design. This can lead to issues if architectural or design considerations are neglected.

Complexity for Large Projects: Agile works well for small to medium-sized projects, but its flexibility and focus on small increments can lead to complexities in managing large, complex projects.
Incompatible with Fixed-Price Contracts: Agile's iterative nature and evolving scope can

make it challenging to work within fixed-price contracts, which often require predefined scope and costs.

Cultural Change Challenges: Transitioning to an Agile culture may require a significant cultural shift within an organization. Resistance to change can hinder successful adoption.

Overemphasis on Short-Term Deliverables: Agile's focus on short iterations can sometimes result in teams prioritizing tasks that yield quick wins, potentially neglecting longer-term architectural or technical improvements.

Testing and Quality Challenges: Frequent changes and iterations can lead to challenges in maintaining consistent testing and quality assurance practices, resulting in potential defects slipping through the cracks.

Difficulty in Measuring Progress: Traditional metrics like project completion percentages or earned value may not align well with Agile's iterative nature, making it difficult to measure and report progress accurately.

What is the need of devops?

DevOps, short for Development and Operations, is a set of practices, principles, and cultural philosophies that aim to bridge the gap between software development and IT operations teams. The primary need for DevOps arises from the challenges and inefficiencies that traditional development and operations approaches often face. DevOps addresses these challenges by promoting collaboration, automation, and a culture of continuous improvement.

Here are some key reasons why DevOps is needed:

Faster Software Delivery: Traditional development and operations processes can be slow and fragmented. DevOps aims to accelerate software delivery by automating manual processes, enabling continuous integration and continuous delivery (CI/CD), and promoting rapid deployment.

Agile Alignment: DevOps complements Agile development methodologies by enabling the continuous delivery of small, incremental changes. This aligns with Agile's iterative approach and ensures that software can be released more frequently and respond to changing requirements more effectively.

Reduced Risk: Automated testing, continuous monitoring, and frequent deployments in DevOps can lead to earlier identification of issues, reducing the risk of major failures in production. This helps catch and address problems before they impact users.

Improved Collaboration: DevOps encourages collaboration between development, operations, and other cross-functional teams. This collaboration leads to better communication, shared goals, and improved understanding of each team's needs and challenges.

Efficient Resource Utilization: By automating infrastructure provisioning, scaling, and management, DevOps helps optimize resource usage and reduce manual intervention, leading to cost savings and more efficient use of resources.

Consistent Environments: Automation ensures that development, testing, and production environments are consistent, reducing the "it works on my machine" problem and minimizing configuration-related issues.

Faster Problem Resolution: DevOps practices like monitoring, log analysis, and automated testing make it easier to identify the root cause of issues. This speeds up troubleshooting and resolution.

Cultural Transformation: DevOps promotes a culture of collaboration, shared responsibility, and continuous improvement. This cultural shift encourages teams to break down silos, communicate openly, and take ownership of the entire software lifecycle.

Customer-Centric Approach: DevOps focuses on delivering value to customers quickly

and consistently. This customer-centric approach aligns with the modern demand for faster feature releases and responsiveness to user feedback.

Innovation and Experimentation: DevOps encourages innovation by enabling teams to experiment, iterate, and deploy new features and ideas rapidly. Failures can be quickly corrected, and successful experiments can be scaled up.

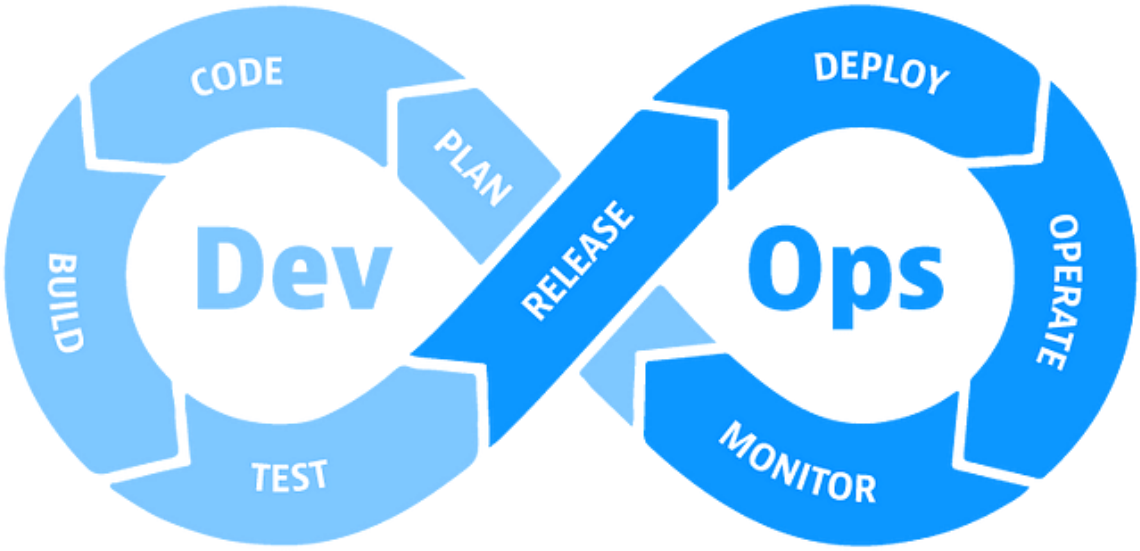
Scalability and Resilience: Automation and infrastructure-as-code practices in DevOps make it easier to scale applications up or down based on demand. It also contributes to building resilient and highly available systems.

Competitive Advantage: Organizations that adopt DevOps practices often find themselves better positioned to respond to market changes, adapt to emerging technologies, and stay competitive in their industries.

DevOps stages along with some commonly used tools for each stage:

What is DevOps?

DevOps is a methodology in the software development and IT industry. Used as a set of practices and tools, DevOps integrates and automates the work of software development and IT operations as a means for improving and shortening the systems development life cycle.



Source. DevOps Cycle

Stages of DevOps?

Continuous Development

ontinuous Integration

Continuous Testing

Continuous Deployment

Continuous Operations

Continuous Monitoring & logging

Continuous Feedback

1. Continuous Development

This stage involves planning and coding the product. Use tools, such as Git, SVN, CVS, etc..

Fig. Continuous Development

2. Continuous Integration

In the continuous integration stage, the source code in the central repository(Github) is regularly updated by developers. Also in this stage, The code compile, validate, code renew, unit testing and integration testing performed by the developers.

In this DevOps process Jenkins, Bamboo, GitLab etc, tool are used.

Fig. Continuous Integration

3. Continuous Testing

In the continuous testing, Developer build code/programme/app will deploy on test server. The tools used in this stage is Selenium, Apache Jmeter, Sonarqube and etc.

Fig. Continuous Testing

4. Continuous Deployment

In this, The application is deployed on the production server to make it available for the intended users. Tools used AWS CodeDeploy, Argo, Octopus Deploy, etc.

Fig. Continuous Deployment

5. Continuous Operations

The continuous operations stage involves the reduction or elimination of planned downtime like scheduled maintenance. The goal of this phase is to increase the uptime or the time the users can use the application. Companies use container management systems like Kubernetes or Swarm in this phase.

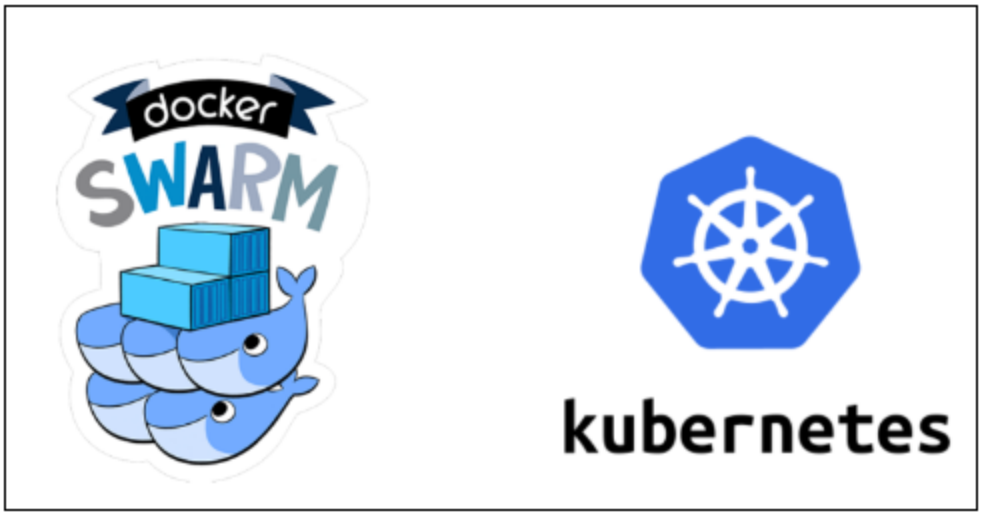


Fig. Continuous Operations

6. Continuous Monitoring & logging

This stage involves monitoring the health, performance, and reliability of the application or code, as well as the infrastructure, as the phases move from development to deployment. also, It observe and detect the compliance issues and security threats during each phase of the DevOps pipeline.

The tools used in this phase are New-relic, Grafana, Nagios, Aws CloudWatch, etc.

Fig. Continuous Monitoring & logging

7. Continuous Feedback

As digital technology continues to transform and the use of social networks continues to grow, user feedback on specific applications is crucial. Feedback today is almost immediate and can spread to multiple people in seconds. And it can be furious and unforgiving.

On the other hand, good feedback can increase both mind share and market share for your organization, directly impacting your corporate image.

With the advent of agile development, design thinking and DevOps, teams have a clear approach to provide new application releases directly to lines of business. Several times I have seen teams developing apps in an agile way — integrating more, testing more, delivering and deploying more — but in the end the user outcome and satisfaction was not optimal.

This is why continuous feedback is fundamental when it comes to continuous delivery in DevOps. An application delivered faster with weekly releases and with higher quality does not guarantee business outcomes nor user satisfaction.

Continuous feedback is essential to application release and deployment because it evaluates the effect of each release on the user experience and then reports that evaluation back to the DevOps team to improve future releases.

The feedback can be gathered in two methods: structured and unstructured. The structured method is applied through surveys, questionnaires, and focus groups. The unstructured feedback collection is done through social media platforms such as Twitter, Facebook, etc. Here, the users take part in this DevOps process by providing their feedback, just like how users provide app reviews on Google Playstore.