DevOps

History of DevOps

- **Emergence of Agile:** In the early 2000s, Agile methodologies started gaining popularity in software development, emphasizing iterative and collaborative development.

Patrick Debois, a Belgian software developer, pioneered DevOps by organizing the first "DevOpsDays" conference in 2009. This marked the beginning of the movement to integrate development and operations, revolutionizing software development processes worldwide. His efforts continue to shape the DevOps landscape today.

History of DevOps

- **Emergence of Agile:** In the early 2000s, Agile methodologies started gaining popularity in software development, emphasizing iterative and collaborative development.
- **Emergence of Lean Manufacturing:** In the same period, Lean Manufacturing principles started getting applied to software development, emphasizing eliminating waste and maximizing efficiency.
- **The Need for DevOps:** As software development became more complex, the need for more efficient and collaborative processes between development and operations teams became apparent.
- **Early DevOps Practices:** Early DevOps practices included continuous integration, continuous delivery, and automated testing.

History of DevOps (cont'd)

- **The Rise of Cloud Computing:** The emergence of cloud computing in the mid-2000s made it possible to automate infrastructure provisioning, further accelerating the DevOps movement.
- **Standardization:** In 2012, the DevOps movement gained more momentum with the publication of "The Phoenix Project," a book that promoted the adoption of standardized practices across development and operations teams.
- **DevOps Goes Mainstream:** By 2014, DevOps had become a mainstream approach to software development, with large organizations like Amazon, Netflix, and Etsy using it to improve their software delivery processes.

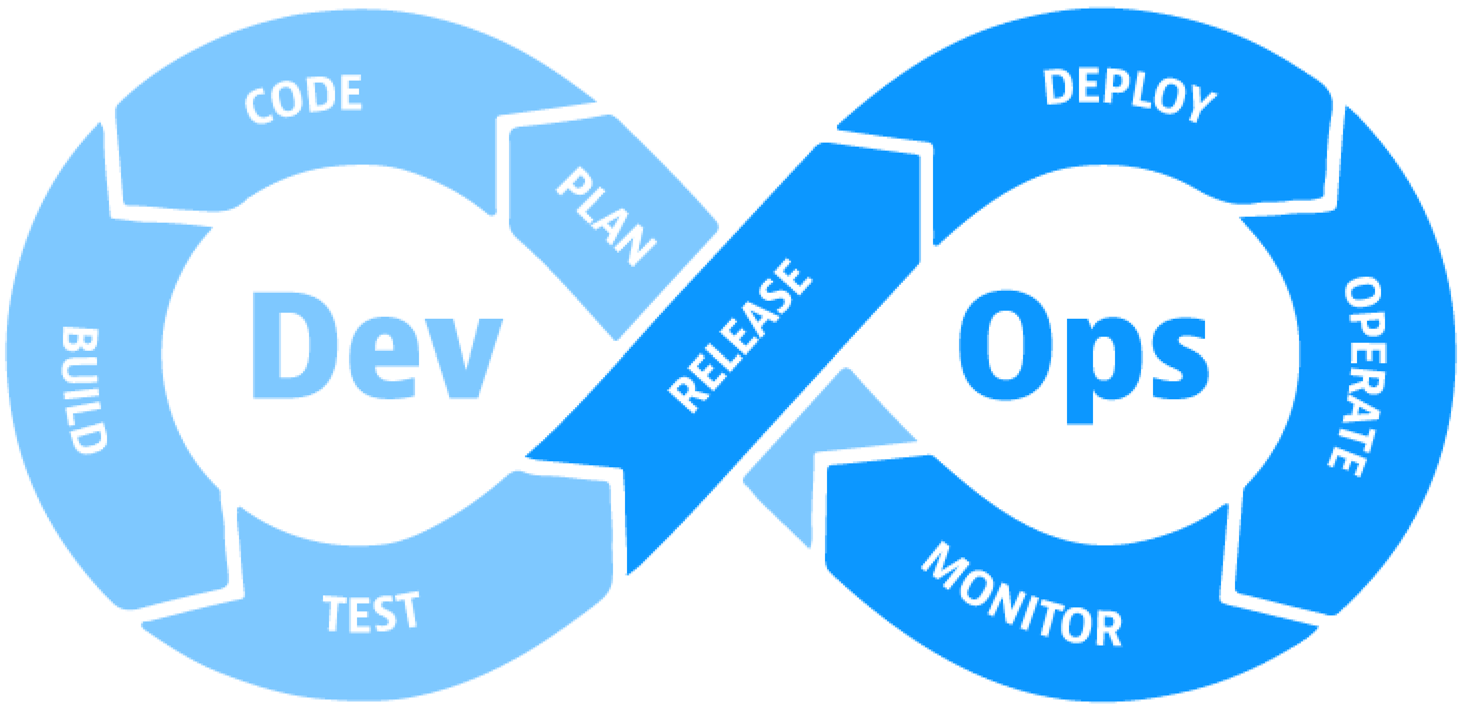
History of DevOps (cont'd)

- **DevSecOps:** In recent years, there has been a shift towards a more holistic approach to software development, known as DevSecOps. This approach emphasizes integrating security practices into the DevOps process, ensuring that security is a key consideration from the earliest stages of development.
- **Cloud-Native DevOps:** The latest phase of the DevOps movement is focused on cloud-native DevOps, involving using cloud computing resources and services to streamline the development process. This includes the use of microservices architecture, serverless computing, and other cloud-native technologies.

History of DevOps (some numbers)

- DevOps was first used in an Agile conference in 2008
- Amazon was one of the first adopter
  - 1000 deploys per hour
- Netflix since 2013
  - 100 release per day
- Etsy
  - 1 billion dollars in transactions every year
- Flickr
  - 40000 photos per second
- WebMD
  - Deployment reduced from 2 days to 60 seconds

DevOps


CODE
PLAN
BUILD
Dev
TEST
RELEASE
DEPLOY
Ops
OPERATE
MONITOR

Main concepts DevOps

- **New ideas** about software development
- Ecosystem of **tools** (to automate and streamline the software development lifecycle)
- **Collaboration** (promotes a culture of collaboration between development, fostering communication and shared responsibility.)
- Blurring **responsibilities**
- Increase **efficiencies** (reducing time to market and improving product quality)

- **Evolution** of software (to embrace agility, flexibility, and rapid iteration)
- Beyond traditional roles of **dev**elopers and **op**eration**s**
- **Blur** developers, operations and testers
- **All** personnel is involved in software development

DevOps as a Methodology

Definition: DevOps is a twofold methodology combining philosophies, tools, and practices from software development ("Dev") and IT operations ("Ops").

Its goal is to accelerate software product delivery.

- **Benefits:** Increased velocity allows organizations to **reduce time to market** for their products, better meeting customer needs.
- **Culture of Collaboration:** DevOps relies on a culture of collaboration, **breaking down silos** between development and operations teams. Together, they form the cohesive "DevOps" team.

DevOps as a Profession

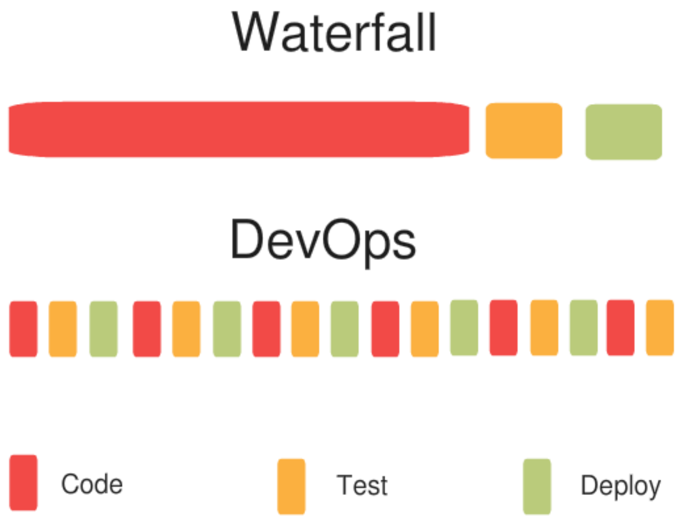
Definition: DevOps is also a profession. A DevOps engineer is responsible for both the **reliability** of the software product ("Ops") and the **speed** of its development ("Dev").

- **Multidisciplinary Role:** DevOps encompasses duties previously performed by distinct roles such as system administrators, release engineers, and software developers.

- **Key Responsibilities:** DevOps engineers implement continuous integration (CI) and continuous delivery (CD) processes, automate software build and deployment, and manage on-premise or cloud infrastructure.

Traditional roles

- Even in agile methodologies, role are quite **sharp** and **fixed**
- Similar to phases in the waterfall model
- Developers
  - Write the code
- Operations
  - Prepare the hardware (computers and networks)
  - Apply patches to operating systems
  - Deploy the application
- Quality assurance
  - Test the code
  - Give feedback to developer



Limitations of traditional roles

- More **complexity**
- Developers must **know** the environment in which their program will execute
- Innovative ideas can **hardly** emerge
- Tester have **little idea** of the needed functionalities
- **Deployment** problems
- Programs are **dependent** on the software environment
- **Black box**: people cannot look into the application

- **Collaboration** is needed

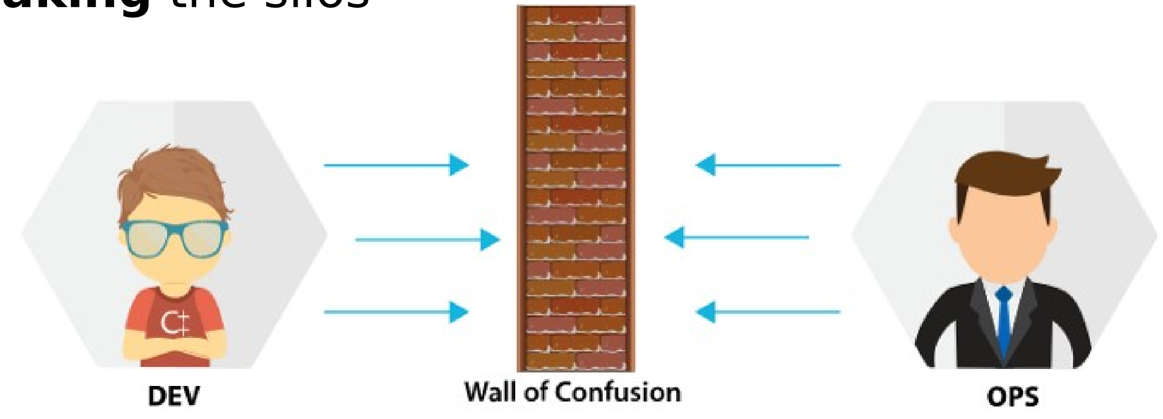
DevOps

- DevOps can be applied to **traditional** software development cycle
- Addresses **collaboration**

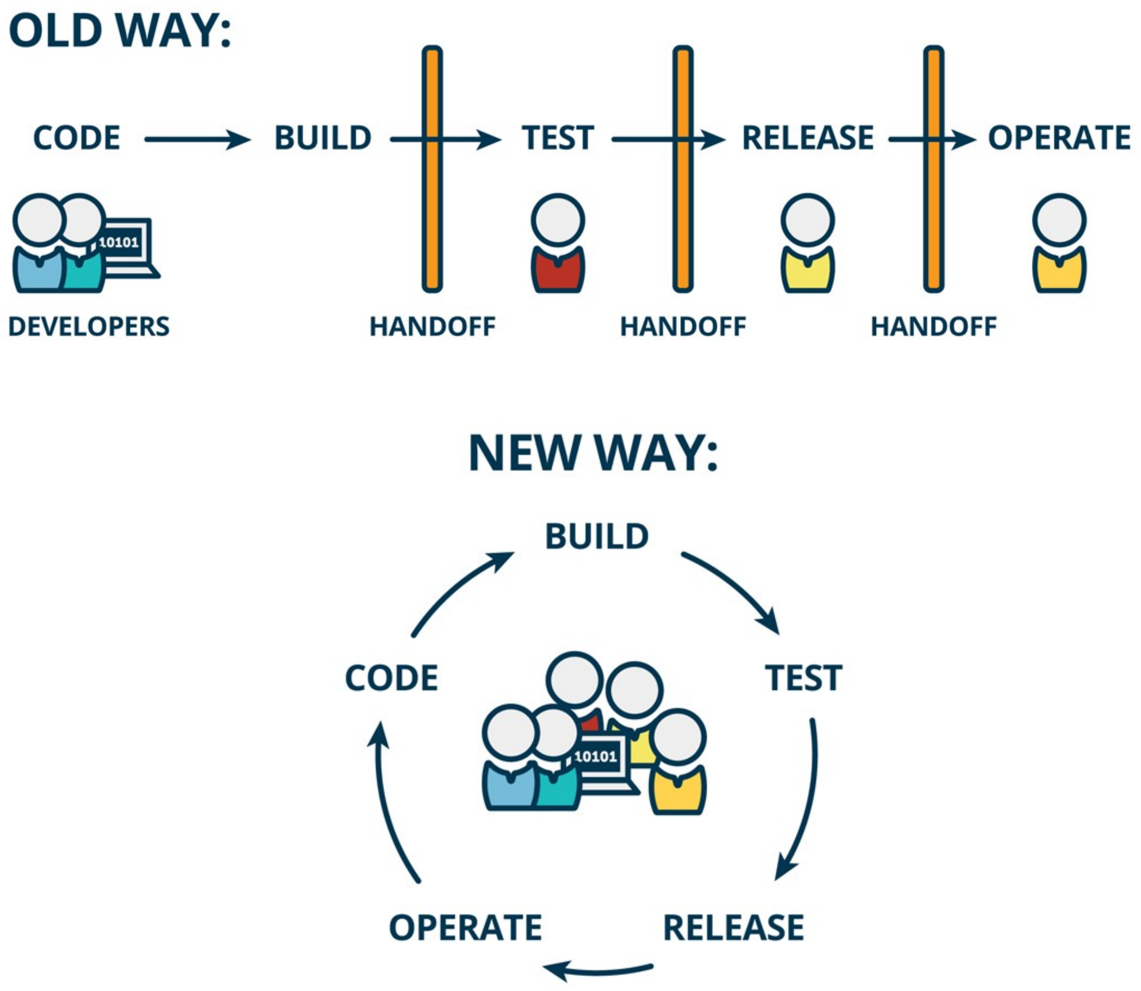
- Collaboration reduces software development **time**
- Collaboration improves **quality** and **efficiency**
- Software development is an "**enterprise-wide** venture" rather than an "IT project"
- Rely on **automation** of the software process
- **Standardization** of software deployment
- Security in considered in **all** phases

Silos

- Silos represent **specific** activities that are clearly **separated** from others
- Can be considered as the phases
- For example:
  - Requirement analysis
  - Program design
  - Program coding
  - Program documentation
  - Program testing
- DevOps aims at **breaking** the silos



Silos (2)


OLD WAY:
CODE
BUILD
TEST
RELEASE
OPERATE
10101
DEVELOPERS
HANDOFF
HANDOFF
HANDOFF
NEW WAY:
BUILD
CODE
TEST
10101
OPERATE
RELEASE

Cons

- DevOps is perceived **differently** by different organizations
- **No** universally agreement on benefits (making it difficult for some organizations to justify its adoption)
- DevOps adds **complexity** (especially in organizations with existing processes and systems that may need to be reconfigured or replaced)
- DevOps process must be **managed** (without proper oversight, DevOps initiatives can fail to deliver the expected benefits)
- DevOps has a **cost-benefit** implementation curve
  - It can be **hard** at the beginning
  - Initial investment required to adopt DevOps practices may outweigh the immediate benefits

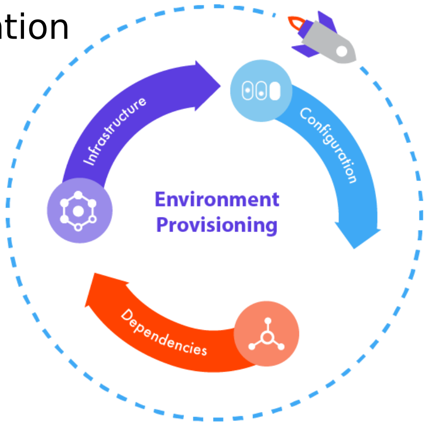
Cons (2)

- DevOps adds **redundancy** that may not be needed (leading to inefficiencies if not carefully managed)
- DevOps requires organization **acceptance**, not only single person or team acceptance
- There are **no standards** in DevOps tools (leading to interoperability issues)
- DevOps requires **financial** commitments
  - For tools, training, and infrastructure
  - A barrier for some organizations
- Not clear Return On Investment (**ROI**)
  - The ROI of DevOps initiatives may not be immediately clear, making it difficult for organizations to justify the associated costs

Hardware provisioning

- Use of **data centers** (to host and manage hardware infrastructure)
- Hardware is **moved**
- Managed **hosting**
- Data centers provide the “**ops**” part

- Separation between “dev” and “ops” can lead to **problems** (communication and collaboration challenges)
- Cloud provisioning has **blurred** the lines between dev and ops
  - By providing virtualized infrastructure that can be provisioned and managed programmatically
- No longer **physical** servers
- Provisioning tasks become more **encapsulated** and **automated**
  - Streamlining the deployment process and reducing manual intervention



Configuration management

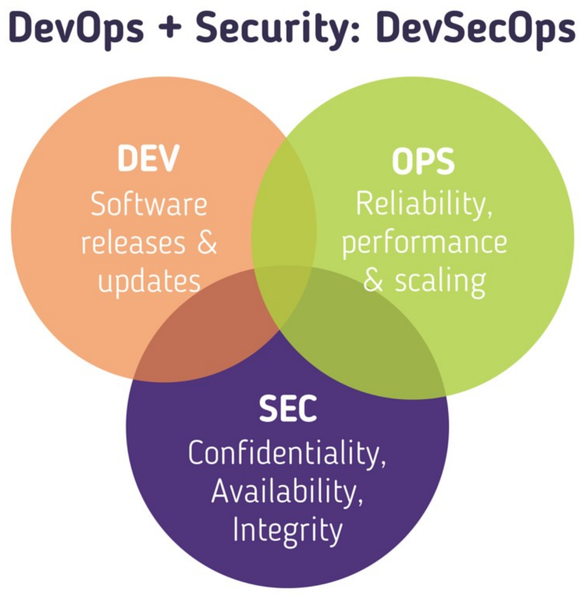
- **Different** configurations
  - depending on the environment they are deployed in
  - leading to multiple configuration variations
- Testing more configurations could be **more complex** than testing the application (time-consuming)
- Syndrome “it works on **my** machine”
  - the application behaves differently on different environments due to configuration differences
- Different **operating systems** (each requiring its own specific configuration)
  - Different versions
  - Different distributions
- Applications depend on **shared** piece of code
  - JARs, assembly, plugins (which need to be properly configured and managed)
- DevOps has **tools** for automatic configuration
  - Configuration as code
  - Allowing for consistent and repeatable deployments across different environments

Security

- Security is considered in **all** phases, by **all** people
- Security should be included **as early as possible** in a DevOps application
  - from the earliest stages to ensure that vulnerabilities are identified and addressed early on.

Security is one of the **aim** of DevOps
  - Not just efficiency
  - but as a key aspect of software quality.

- Security **experts** must be involved in choosing the DevOps tools
- Security is to be **aligned** to business (ensuring that security efforts contribute to overall business success)
- Security adds **value** to the development
  - Investing in security measures adds value to the development process by protecting against potential **breaches**, **data loss**, and other **security risks**.


DevOps + Security: DevSecOps
DEV
Software releases & updates
OPS
Reliability, performance & scaling
SEC
Confidentiality, Availability, Integrity

Virtualization

- Physical hardware is **slow**, **expensive** and **error prone**
- Instant **virtualization**
  - rapid provisioning of virtual machines and environments
  - enabling developers to quickly set up and tear down infrastructure as needed.
- Platform-as-a-Service (**PaaS**)
  - **Ready-to-use environments** and tools for building, deploying, and managing applications without the need to manage underlying infrastructure
- Tools can **automate** the virtualization
  - E.g., Vagrant
- **Disposable** environments
- “what if” scenarios
  - Virtualization allows developers to **simulate** scenarios by creating multiple environments with different configurations, helping to identify potential issues and plan for contingencies.

Operations

- Failures **cannot** be avoided completely, it is essential to acknowledge this reality and plan for **resilience**.
  - Fail small (small and localized rather than widespread, minimizing their impact on operations)
  - Fail early (early in the development and deployment process helps prevent larger issues down the line)
  - Recover fast (recover quickly from failures is crucial for maintaining system uptime and minimizing disruptions to users)
- **Smaller** operation processes
  - Breaking down operations into smaller, manageable processes makes it easier to identify and address issues, improving overall system reliability

DevOps workflow

- **Alignment** of goals
  - DevOps promotes the alignment of goals across development, operations, and other teams, ensuring everyone is working towards the same objectives.
- Better **collaboration** between groups
  - breaking down silos and encouraging cross-functional teamwork
- Continuous **improvement**
  - Not just tested functionality (testing infrastructure, security, and other non-functional aspects of software development)
  - Teams constantly seeking ways to streamline processes, increase efficiency, and enhance quality
- Continuous **delivery**
  - Release new features and updates to users quickly and frequently

Requirement gathering

- Requirements are based on **user** feedback
- Requirements reflect what is **possible**
  - grounded in reality and reflect what is technically feasible
- Often **nontechnical** requirements
  - requirements often include non-technical aspects such as usability, performance, and **scalability**
- Requirements reflect current **skill** set
  - capabilities influence the feasibility and implementation of requirements
- **Shared** goals
  - aligning stakeholders around shared goals and objectives to ensure everyone is working towards the same outcomes

Some common System Requirements:

- **Automation:** for reducing errors and conflicts, and increasing speed.
  - We aim for fully automated releases to enhance reliability and decrease downtime, utilizing tools like Chef or Ansible for an "Infrastructure as Code" approach.
- **Scalability:** particularly of data, is essential in DevOps.
  - When building features, we need to consider technical impacts upfront and continually.
  - Questions about shared session states, separate read vs. write connections, and efficient data partitioning are crucial. Can we scale our data on demand?
- **Documentation:** time saved by not documenting can lead to more time spent tracking down information later.
  - Not having documentation as a mandatory requirement is product feature thinking, not DevOps thinking.
  - Any change to the system must include critical information, including dependencies, expected UI states, API calls, config options, and more, to achieve DevOps goals.

Development cycle

- No **restricted** activity
  - Developers have the freedom to explore and experiment without being restricted by rigid processes or procedures
- No **black box**
  - The development process is transparent, with clear visibility into each stage and the ability for stakeholders to understand and contribute to the process
- **Collaboration** between business and developers
  - to ensure that development efforts align with business goals and requirements
- All is **automated**
  - Development processes are automated wherever possible, reducing manual effort and enabling faster, more efficient delivery
- Developers are allowed to perform **operations** tasks
  - Developers are empowered to handle operational tasks, blurring the lines between development and operations and enabling **faster resolution of issues**.

Testing

- Quality Assurance testing (QA)
  - ensures that software meets quality standards through rigorous testing and validation
- User Acceptance Testing (UAT)
  - involves testing the software from the end-user's perspective to ensure it meets their requirements and expectations.
- DevOps enables **collaboration** between QA and UAT
  - testing efforts are coordinated and aligned with development activities
- **Prevent** defects, not fix it
  - leading to higher-quality software and faster delivery
- QA is **no** more a phase
  - Enacted during all the activities
- Quality as **culture**
  - every team member taking responsibility for delivering high-quality software
- QA owns the quality process through SDLC
  - ensuring that quality standards are upheld at every step

How to do DevOps

- Let us imagine to be in a company
- We want to start using DevOps

- Which are the steps we need to follow?
- What do we need to do?

STEP 1 - Pick a component

- The first step is to **start small**.
- Pick a component that is currently in production.
- The ideal component has a **simple code base** with few dependencies and minimal infrastructure.
- This component will be a proving ground where the team cuts its teeth on implementing DevOps.

STEP 2 - Adopt an agile methodology like scrum

- DevOps often pairs with an agile work methodology like Scrum.
- Not all all Scrum rituals and practices need to be adopted.
- Key Scrum elements to consider: Backlog, Sprint, Sprint Planning.

STEP 3 - Use Git-based source control

- **Version control** is a DevOps best practice enabling greater collaboration and faster release cycles.
- Tools like **GitHub** or **Bitbucket** facilitate sharing, collaboration, merging, and backup of software.

STEP 4 - Integrate Source Control with Work Tracking

- **Integrate source control** with the work tracking tool to save time for developers and management.
- Having a **single place** to see everything related to a project is essential for efficiency.

STEP 5 - Write tests

- **CI/CD pipelines** require tests to validate code deployed to various environments.
- Start with **unit tests** and aim for a baseline code coverage, incrementally increasing over time.
- Use **test-driven development** when fixing bugs to organically increase code coverage.
- When **fixing bugs**, write tests that fail in environments where the bug is live, then fix the bug and observe that the tests pass.
- Testing allows teams to see the **effect of code** changes on system behavior before exposing end-users to those changes.

STEP 6 - Build out a CI/CD process to deploy the component

- Consider Deploying to Multiple Environments
  - Building a CI/CD pipeline that deploys to only one environment can lead to hardcoded configurations and limited flexibility.
  - Start by creating separate pipelines for deploying infrastructure and code to multiple environments.
  - The pipeline structure typically involves running unit tests, integration tests, and system tests before and after deployment to each environment.

STEP 6 - Build out a CI/CD process to deploy the component

- Pipeline Structure:
  - Begin with unit and integration tests before deploying to the test environment.
  - Run system tests after deployment to ensure the system is functioning correctly.

- Additional Steps:
  - Expand the pipeline with code linting, static analysis, and security scanning to enforce coding standards, detect anti-patterns, and identify vulnerabilities.

STEP 6 - Build out a CI/CD process to deploy the component

- Infrastructure:
  - Infrastructure differences between environments can impact software execution.
  - Define infrastructure in code using tools like AWS CloudFormation, Terraform, Ansible, Puppet, or Chef.
  - Write modular pipelines for deploying infrastructure to handle dependencies and ensure repeatability.

- Code:
  - CI/CD pipelines for code deployment should focus on testing, repeatability, and recovery.
  - Deployments should be re-entrant and idempotent to ensure repeatability and recoverability.
  - Use feature flags or rollback mechanisms to recover from bad deployments and maintain system stability.

STEP 7 - Add monitoring, alarms, and instrumentation

- Monitoring for System Health:
  - **Monitor** the behavior of the running application in each environment, checking for errors, timeouts, and crashes.
  - Detect problems and raise **trouble tickets** for resolution, while also writing additional tests to catch similar issues in the future.

- Fixing Bugs:
  - A DevOps team takes ownership of software operations, conducting root cause analysis and **fixing bugs promptly**.
  - Write tests to detect and **prevent recurring issues**, reducing technical debt and maintaining operational agility.

- Performance Optimization:
  - After establishing basic health monitoring, focus on **performance tuning** to optimize system efficiency.
  - Prioritize optimization efforts based on monitoring data, **targeting the slowest and most costly components** for improvement.

STEP 8 - Use feature flags to implement canary testing

- Implement Canary Testing:
  - **Wrap each new feature** in a feature flag with an allow list containing test users.
  - New feature code runs only for users in the allow list, enabling **controlled testing** in production-like environments.
  - Monitor metrics, alarms, and instrumentation for signs of problems while the feature is in a canary environment.

- Addressing Problems:
  - Address any issues discovered in an environment **before promoting changes** to the next environment.
  - Problems found in production environments are treated with the **same urgency** as those in test or staging environments.

Tools

- Jenkins
  - Continuous integration tool
- Buildbot
  - Python
- CruiseControl
- MS Team Foundation Server
- CABIE
  - Perl

Tools (2)

- HipChat
  - Communication tool
  - Integrates into other tools
- GitHub
  - Code repository
  - And social network
  - Public repository (projects can be viewed by anyone)
  - Manage forking
- Jira
  - Project issues management
  - Integrates into other tools

Tools (3)

- Confluence
  - Team collaboration
  - Supports mobile devices
  - DB integration
- BitBucket
  - Version control tool
  - Support Git and Mercurial
  - Private repository

References

- Gene Kim, Kevin Behr, and George Spafford. 2013. “The Phoenix Project: A Novel about It, Devops, and Helping Your Business Win (1st ed.)”. IT Revolution Press.
- “DevOps For Dummies”, IBM Limited Edition Published by John Wiley & Sons, Inc. 111 River St. Hoboken, NJ Copyright 2014 by John Wiley & Sons.
- DORA DevOps report