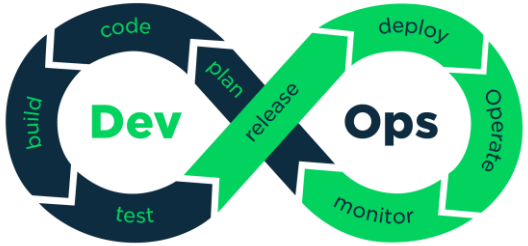
DevOps – Let the Journey Begin



Raman Khanna

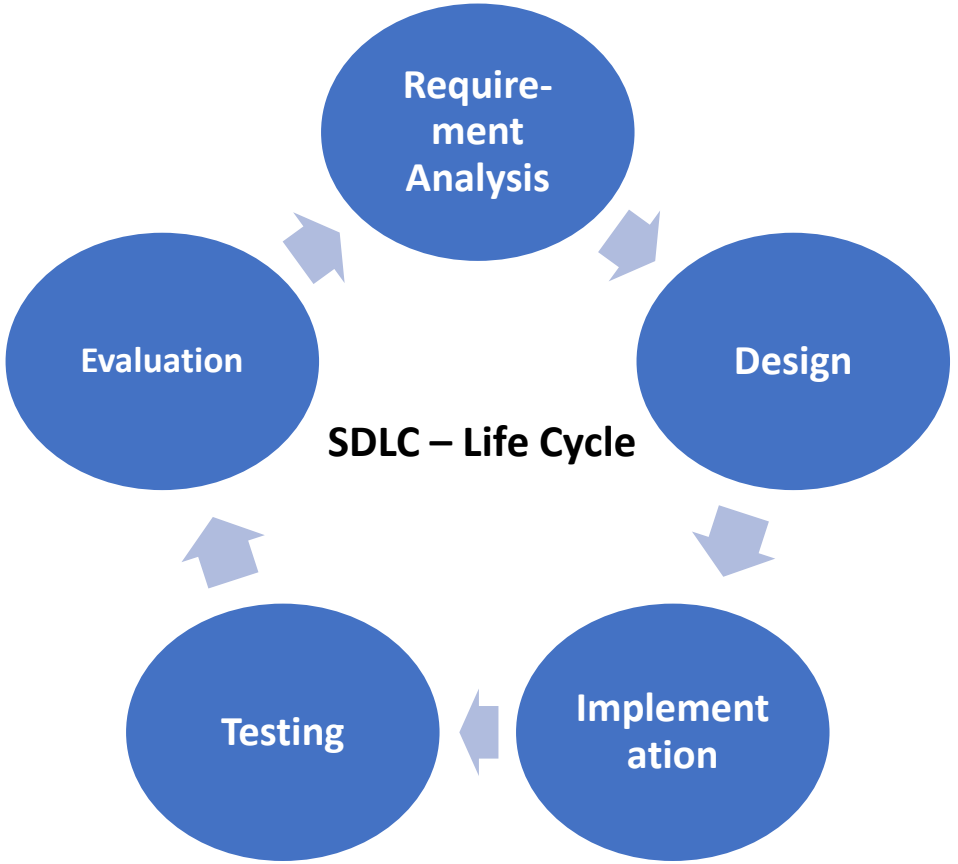
DevOps

What is DevOps?

SDLC Model

SDLC Model

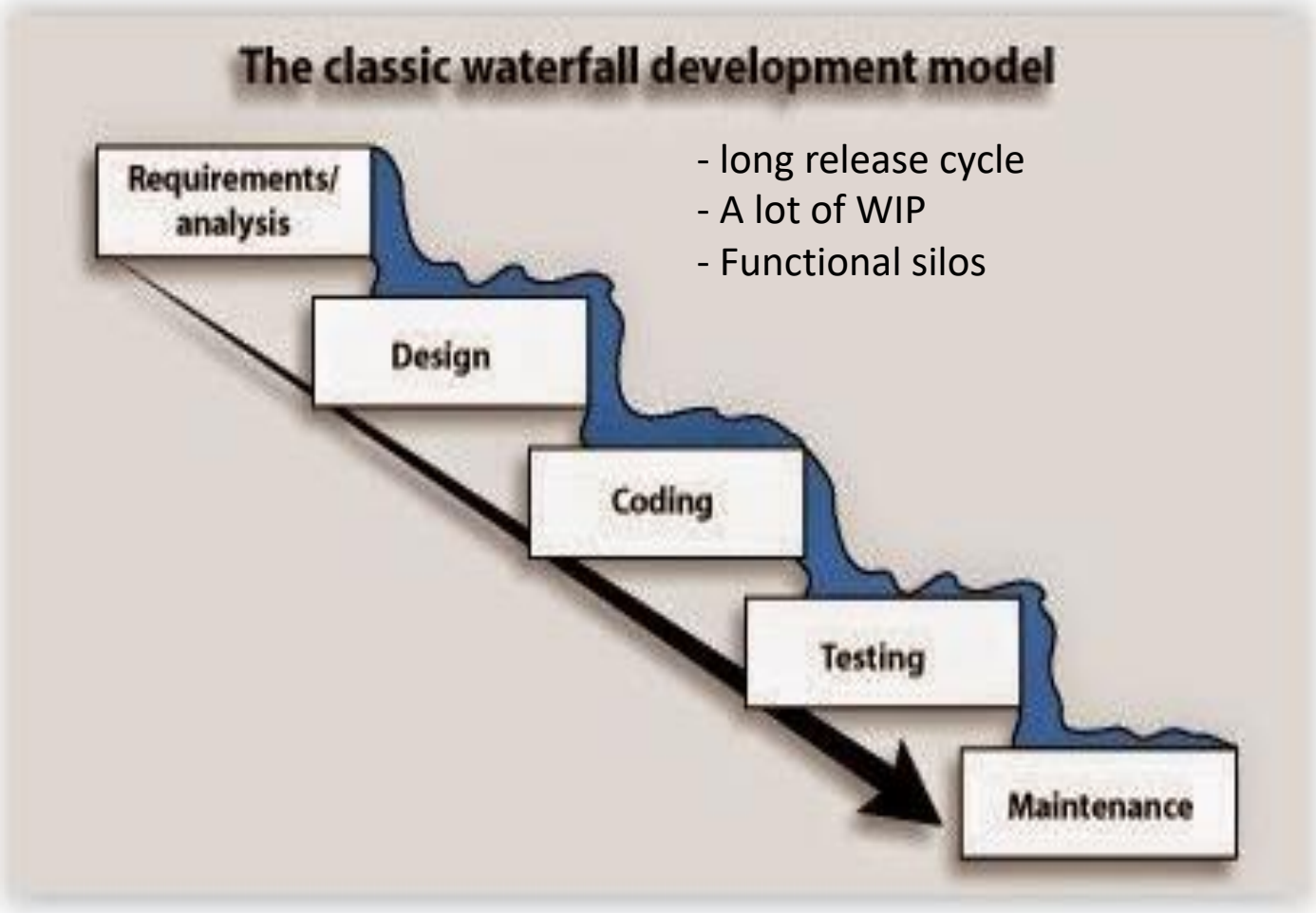
- A systems development life cycle is composed of **several clearly defined and distinct work phases** which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems



Waterfall Development Model

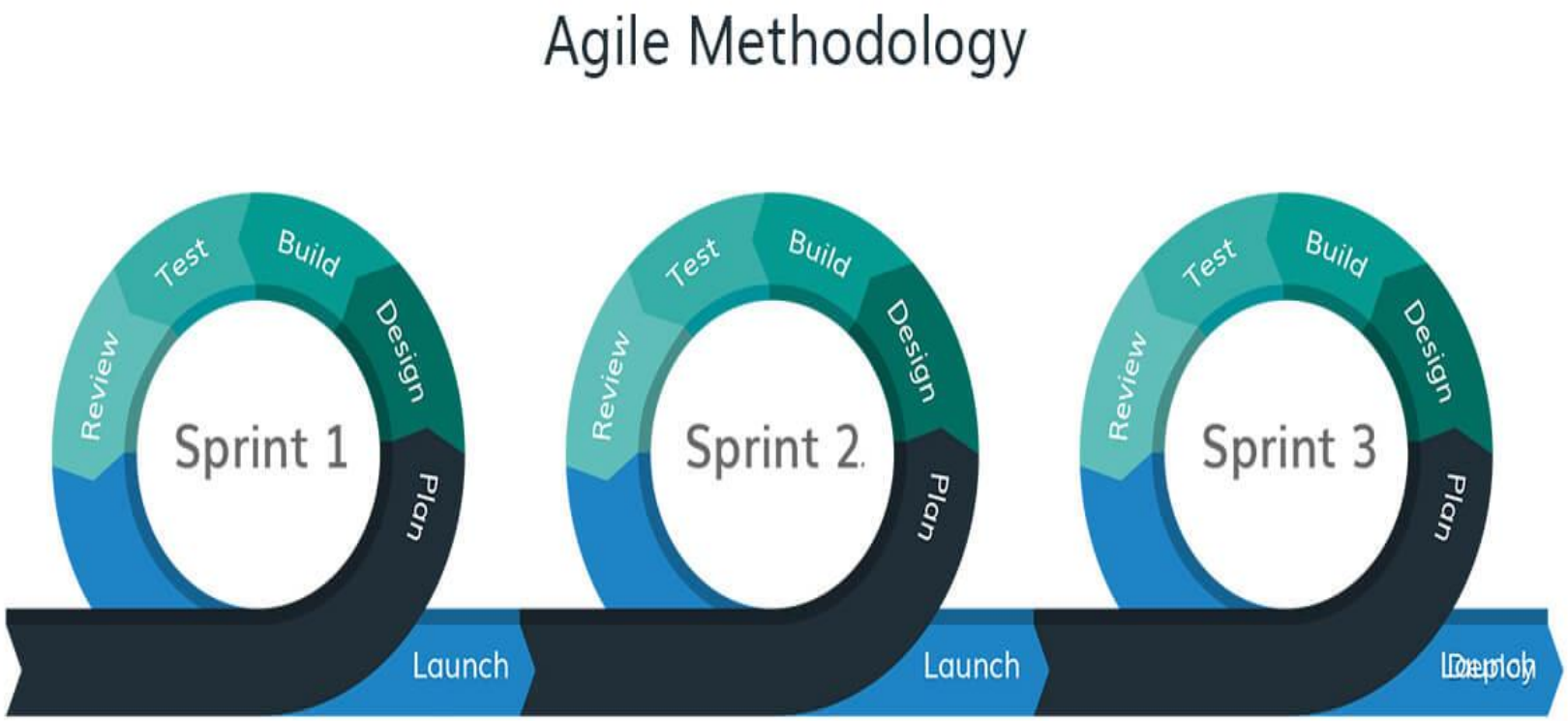
Waterfall Model

1. Determine the Requirements
2. Complete the design
3. Do the coding and testing (unit tests)
4. Perform other tests (functional tests, non-functional tests, Performance testing, bug fixes etc.)
5. At last deploy and maintain



Agile Development Model

Agile



Shorter release cycle

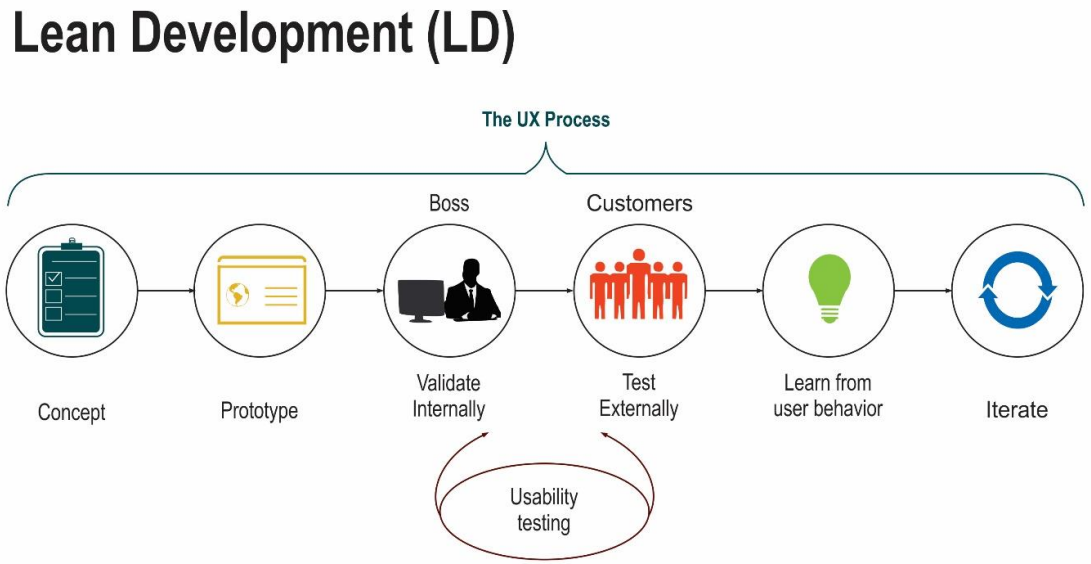
Small batch sizes (MinimumViableProduct)

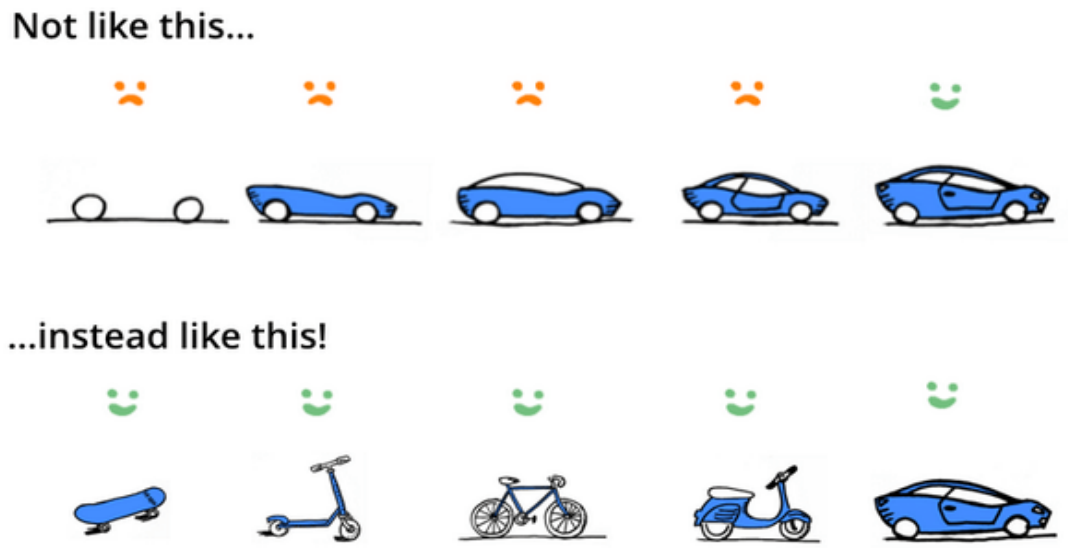
Cross-functional teams

ncredibly agile

Lean Development Model

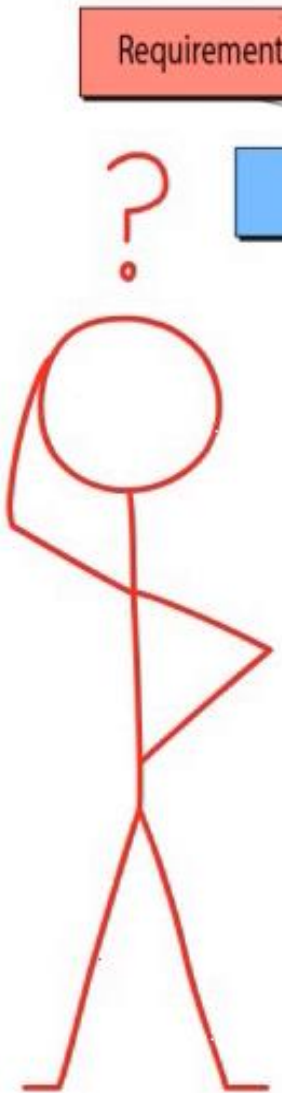
Lean Development

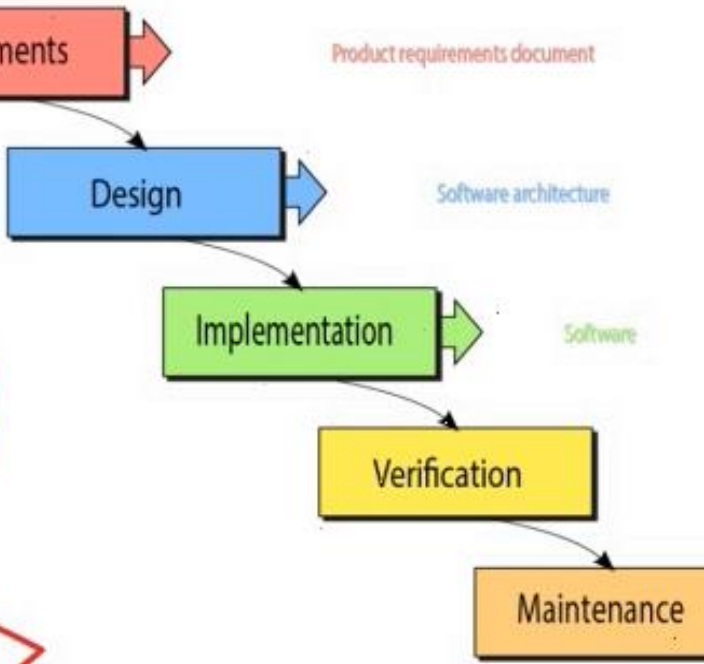




Values of Agile Principle

Individuals and interactions over process and tools.





Working software over comprehensive documents.

Features

Specifications

Layouts

Test Cases

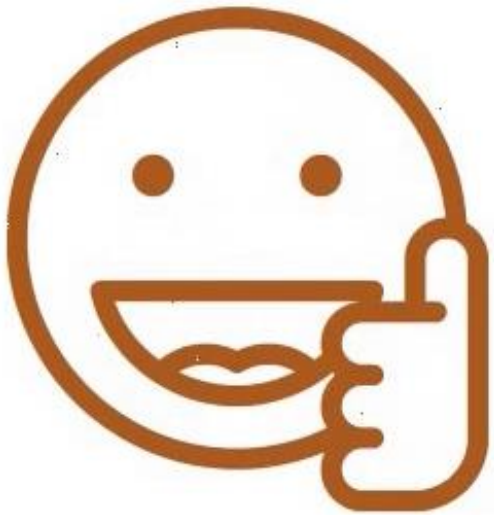
Requirements

Customer collaboration over contract negotiation

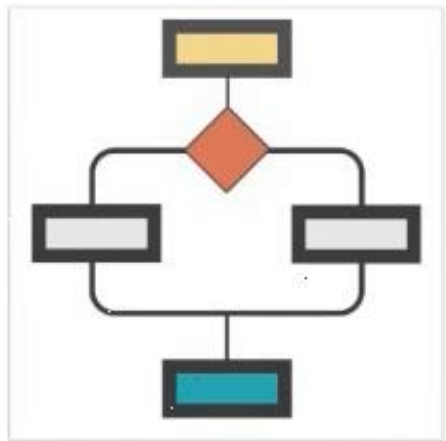
CONTRACT NEGOTIATION

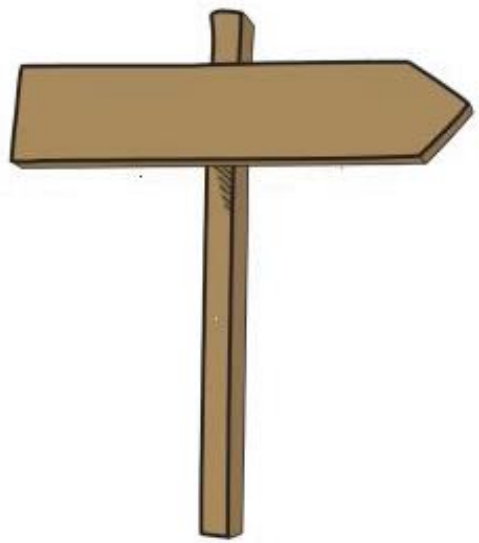
FURTHER CHANGES

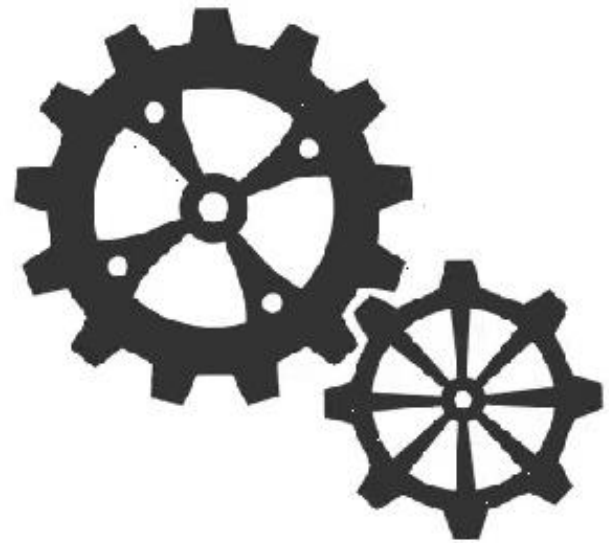
PROJECT COMPLETION



Respond to change over following plan







Principles of Agile Project Management

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development.
Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

JANUARY						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	1	2	3	4	Delivery 5	6
7	8	9	10	11	12	13
14	15	Delivery 16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	Delivery 31	1	2	3

4. Business people and developers must work together daily throughout the project.

> Development team gets an end user view from the business side

5. Build projects around motivated individuals.
Give them the environment and support they need, and trust them to get the job done.

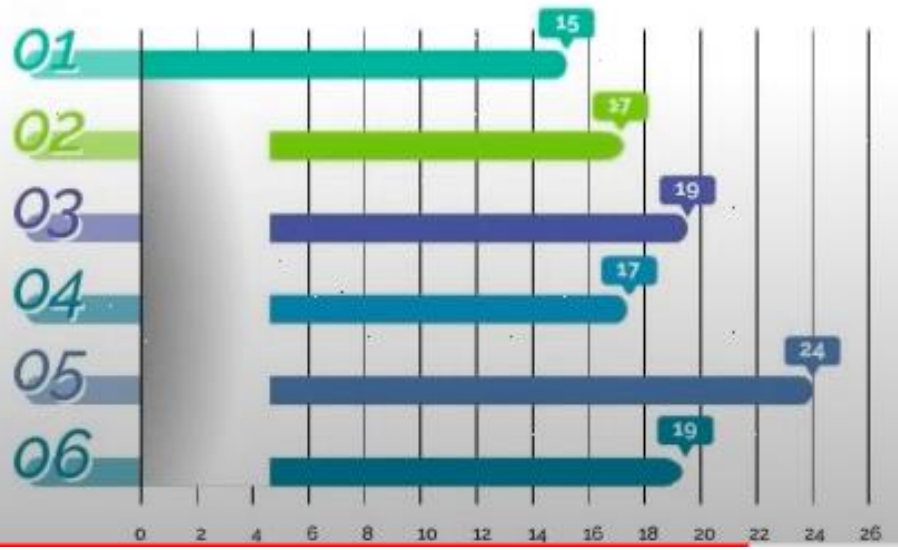
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

A software is not finished when it is successfully tested and delivered, it is finished when it is tested and accepted by the end user.

8. Agile processes promote sustainable development.

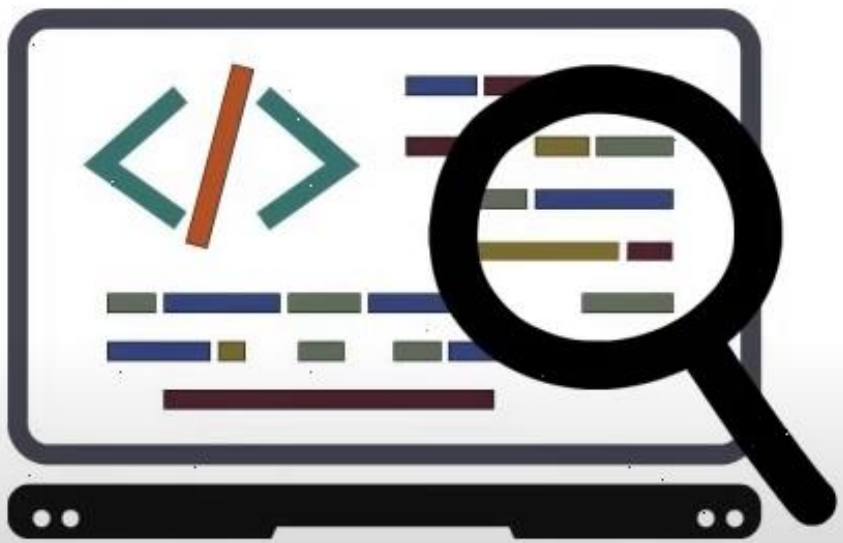
The sponsors, developers, and users should be able to maintain a constant pace indefinitely.



9. Continuous attention to technical excellence and good design enhances agility.

Continuous attention to technical excellence and good design

10. Simplicity--the art of maximizing the amount of work not done--is essential.



COMPLEX CODE

>DEVELOPMENT
>MAINTENANCE

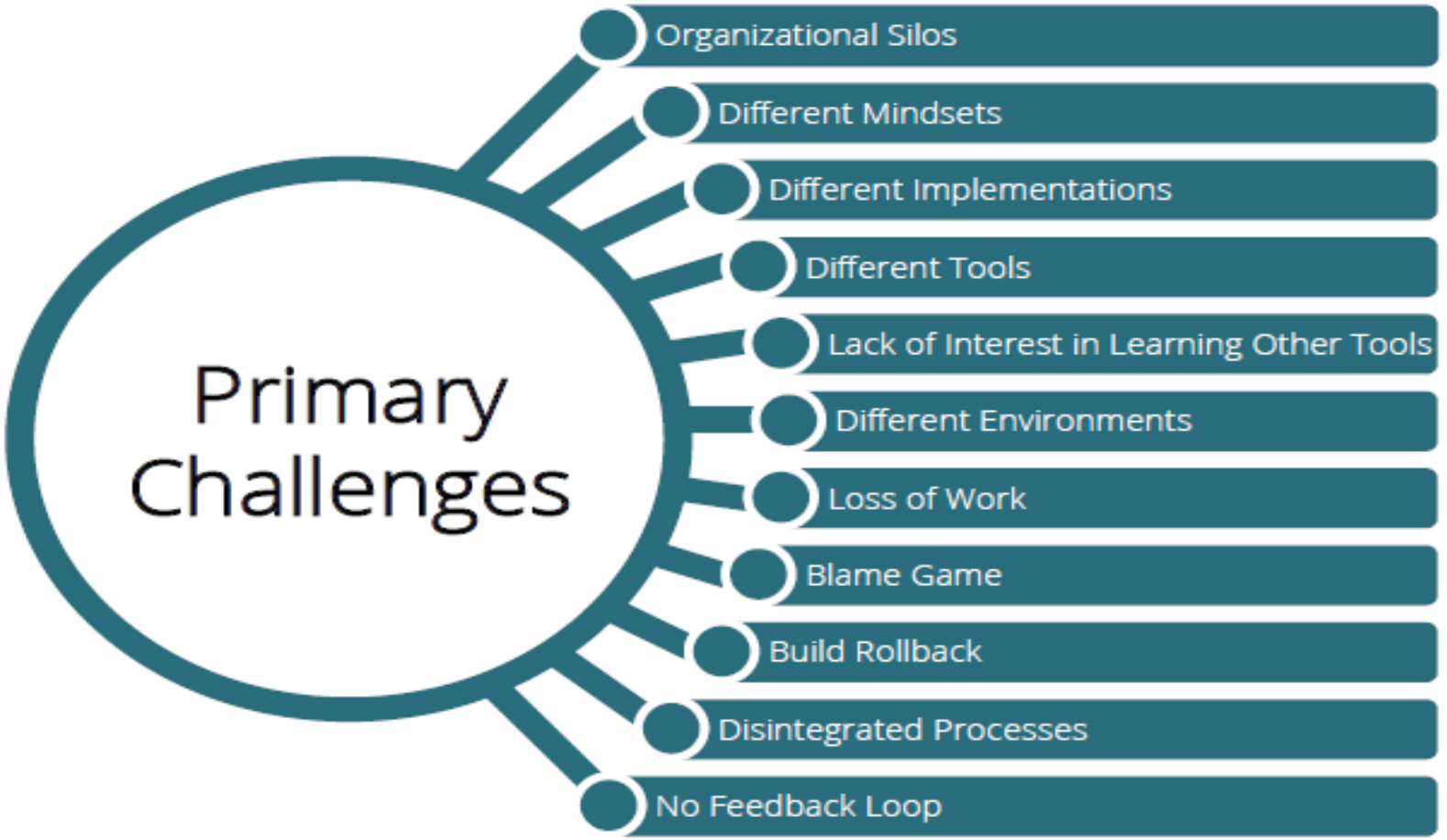
11. The best architectures, requirements, and designs emerge from self-organizing teams

Teams find their own work and manage the associated responsibilities and timelines.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Challenges

Some of the challenges with the traditional teams of Development and Operations are:



A Typical Case Study

- **Development Team:**

- Monday Morning, the writing of code done, unit tests completed, code delivered to the Integration teams to get the code included in CI builds.
- To get the services tested, a ticket is opened for QA teams

- **Build/Release/Testing/Integration Team:**

- Tuesday Morning, ticket accepted, a tester put an email to the developer asking deployment instructions. There is not automated deployments, developer updated to the tester, lets come online and we will deploy the services to the QA environment together.
- Call started, developer identified the “test environment” is not compatible.
- Tuesday afternoon, a ticket raised in Ops Team with new specifications.

- **Ops Team:**

- Wednesday morning, ticket accepted, specifications checked , a new port open request was identified.
- Ticket raised for Security team, ticket accepted, change approved, port opened, email received by the Ops team the work is done.

A Typical Case Study

A Typical Case Study

- **Ops Team:**

- Identified the provisioning requirements again and started work on building the environment.

- **Build/Release/Testing/Integration Team:**

- Thursday Morning, updates received - the environment is ready. Developer and Tester again on call to deploy new services. Services deployed; tester is running test scripts. Next phase is to run regression test cases. Again a new ticket is raised for new test data with production teams and day ends.

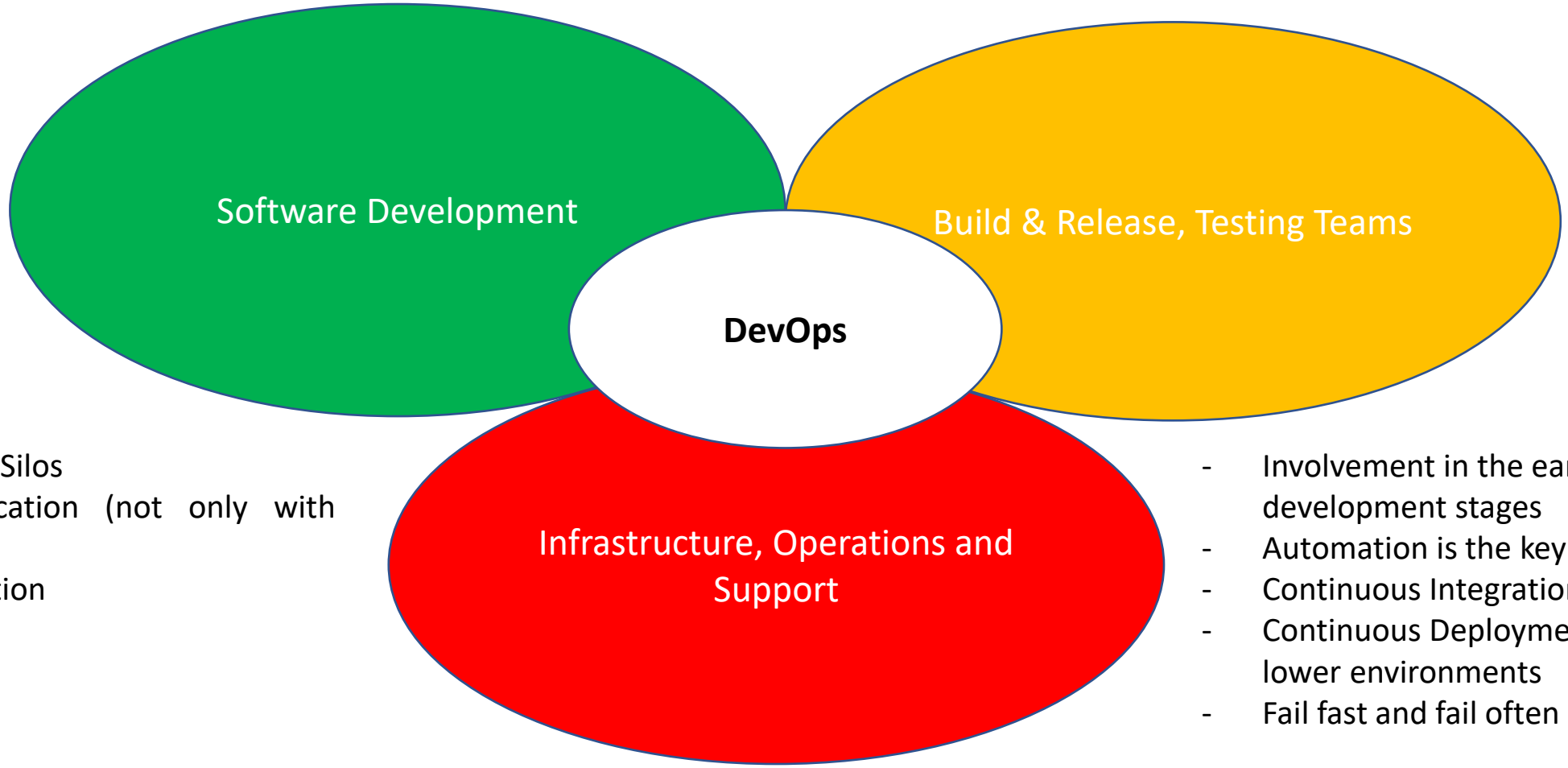
- **Ops Team:**

- Its Friday and the work is not on full swing, ticket accepted but not worked as production team has to complete rest of the works. Somehow the test data is gathered by Friday Evening.

- **Build/Release/Testing/Integration Team:**

- Monday morning, tester gets the data, regression tests run, a defect found, and ticket returned to the development team.

DevOps



- Break the Silos
- Communication (not only with emails)
- Collaboration
- Trust

- Involvement in the early development stages
- Automation is the key
- Continuous Integration
- Continuous Deployments in the lower environments
- Fail fast and fail often

DevOps Model

Collaboration

Communication

Continuity

Automation

Project Management

Customer

Build

Test

Deploy

Planning

Monitoring

Continuous Feedback

Continuous Improvement

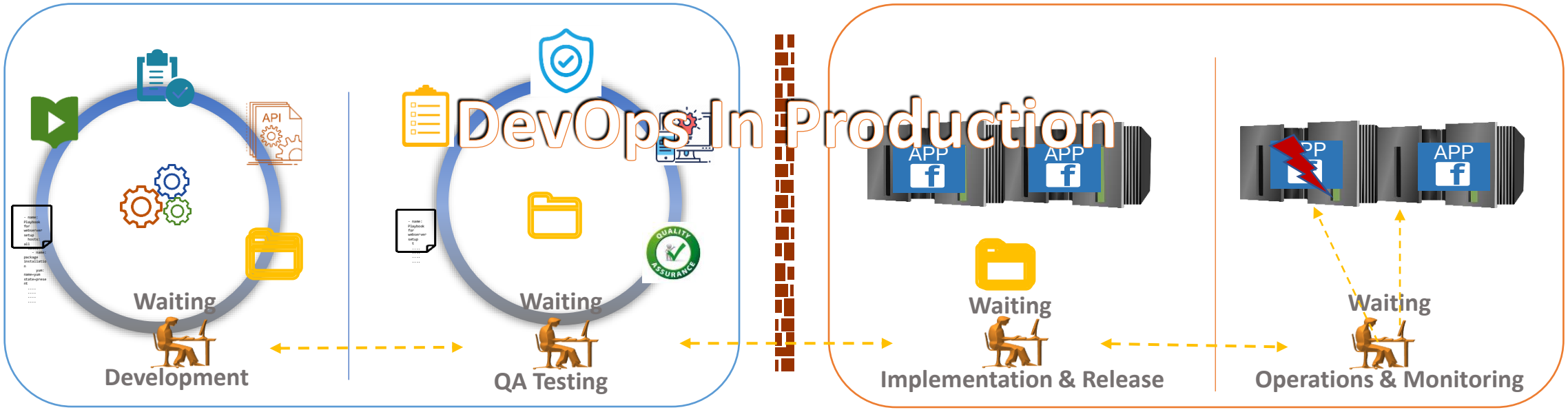
Continuous Planning

Continuous Integration

Continuous Delivery

Continuous Deployment

Continuous Monitoring



Development

Operations

DevOps Essence

Efficiency - Faster time to market

Predictability - Lower failure rate of new releases

Reproducibility – Version everything

Maintainability - Faster time to recovery in the event of a new release crashing or otherwise disabling the current system

DevOps Core Principals

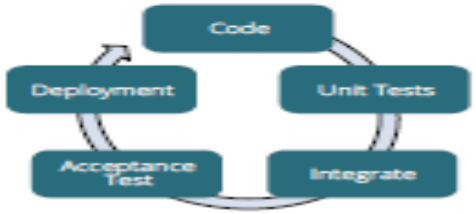
1. Customer-Centric Action

4. Cross-Functional Autonomous Teams

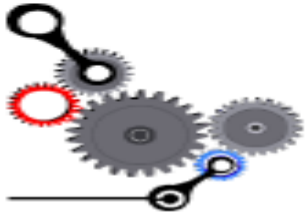
2. Create with the End in Mind

5. Continuous Improvement

3. End-to-End Responsibility



6. Automate Everything You can



How to Build DevOps Organization Culture

Retention is as important as recruitment

Establish Cross-functional team structure

Small teams are better

Give autonomy

Automate with existing staffs and give them a chance to learn

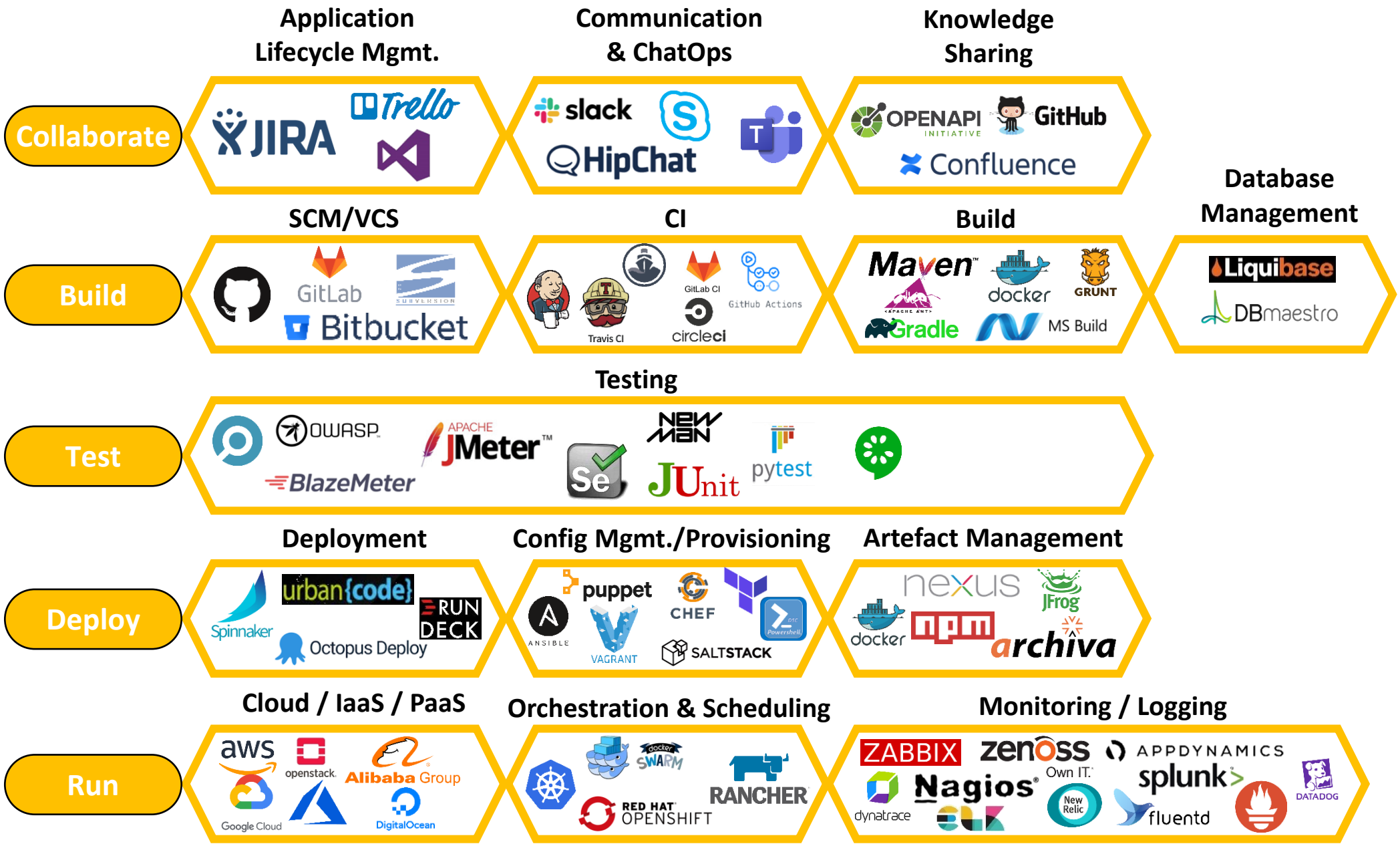
Take out few resources from each team, build a new virtual team for automation.

DevOps Mindset

Moving from traditional development approaches to a DevOps-oriented culture involves major **mindset shifts**:
Traditional Mindset vs. DevOps Mindset

Aspect	Traditional Mindset	DevOps Mindset
Team Structure	Silos between development and operations.	Unified, cross-functional teams working collaboratively.
Focus Area	Separate goals: Dev focuses on features; Ops on uptime.	Shared focus: Delivering features while maintaining reliability.
Speed vs. Stability	One at the expense of the other.	Both are prioritized simultaneously.
Processes	Manual handoffs and approvals.	Automated, streamlined workflows.
Tools	Disconnected tooling per team.	Unified toolchains with centralized governance.
Failure Handling	Blame-oriented and punitive.	Learning-focused and supportive.

DevOps Toolsets



CASE STUDIES

Netflix – High Availability and Continuous Delivery

Challenge:

Netflix faced the challenge of delivering fast, reliable, and scalable streaming services to millions of customers globally. Initially, it took several hours to deploy new code, often resulting in service downtimes. They needed to reduce deployment time and increase uptime across multiple regions while continuously delivering new features.

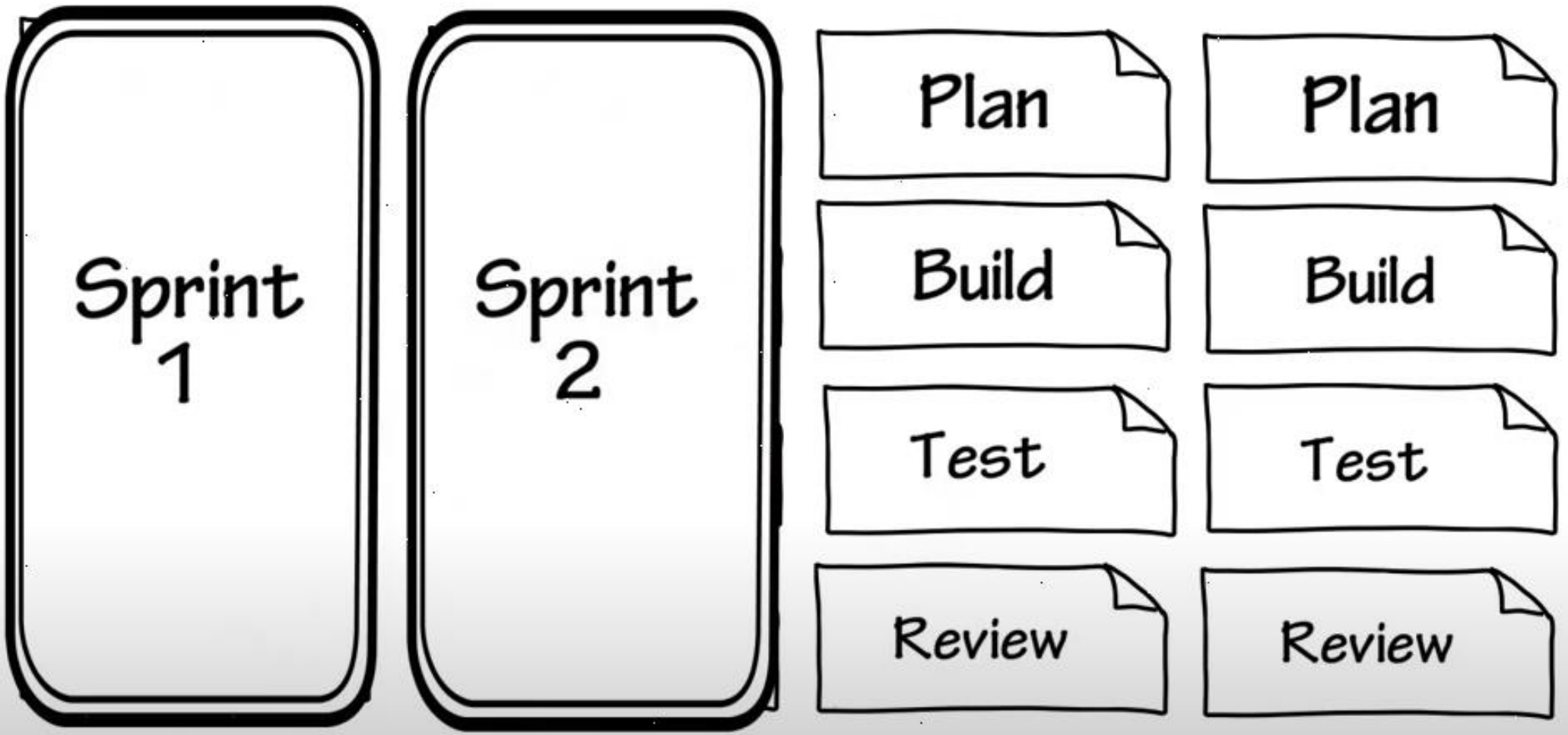
DevOps Implementation:

Cultural Shift: A culture of responsibility and transparency was created. Developers were empowered to deploy their own code, integrating development and operations functions into autonomous teams.
Continuous Integration/Continuous Deployment (CI/CD): Netflix adopted Jenkins as their main automation tool and implemented Spinnaker, a continuous delivery platform they built in-house. The focus was on automated pipelines for testing and deployment to minimize manual intervention.
Automation: Testing and deployment were fully automated, including chaos engineering practices like Simian Army to stress-test the infrastructure and anticipate failures before they occur in production.
Infrastructure as Code (IaC): The use of cloud-based automation tools allowed Netflix to manage its infrastructure without human intervention, ensuring a repeatable, scalable environment across the cloud.

Outcome:

Netflix was able to dramatically reduce deployment times, bringing deployment frequency from hours to minutes.
Automated tools enabled Netflix to push thousands of changes per day, ensuring the system remained scalable and resilient despite the high volume.
The shift to a microservices architecture allowed them to improve fault tolerance, enabling self-healing services.
The continuous feedback loop from monitoring and chaos testing helped mitigate downtime by pinpointing weaknesses before they became critical.

Scrum


Sprint 1
Sprint 2
Plan
Plan
Build
Build
Test
Test
Review
Review
Several incremental releases called Sprints
Potentially Shippable Product

Scrum

Breaks projects into Epics and Stories

Epics – Large Stories or Work Items, usually broken into small storeies

User Stories: Smallest Unit of work

User stories can be prioritized.

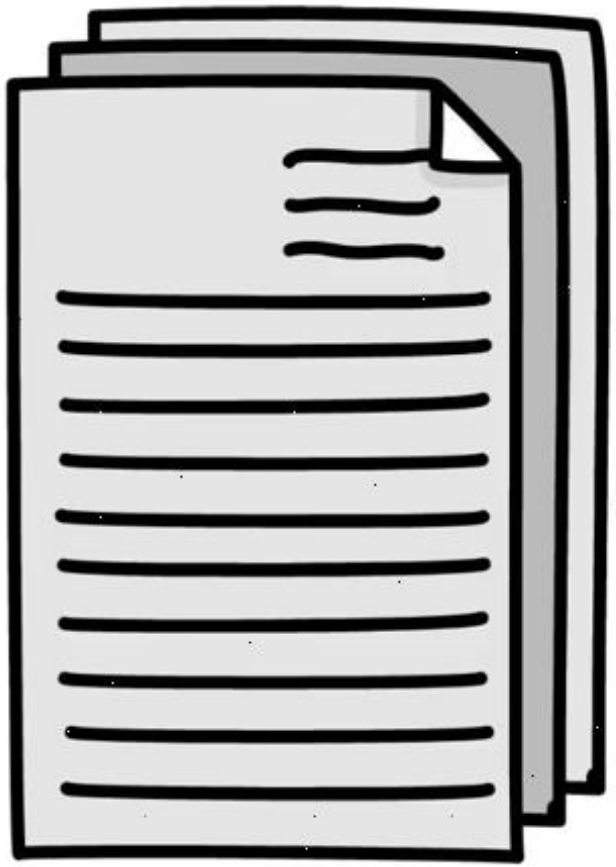
Development is performed in short cycles.

Terms like Product Owner, Development Team, Scrum Master

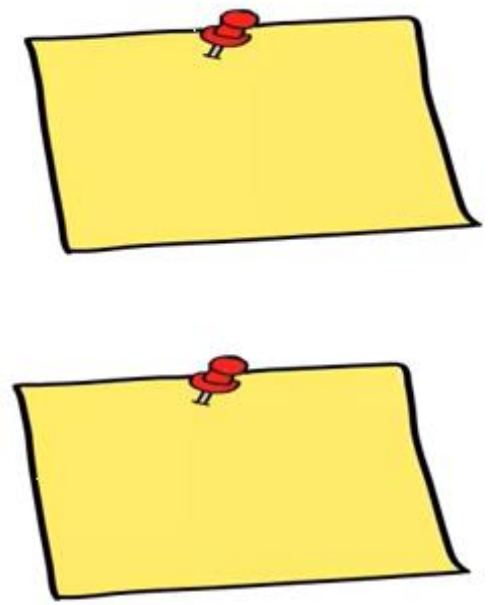
User Stories

3 Artifacts

Product Backlog



Sprint Backlog



Burndown Chart

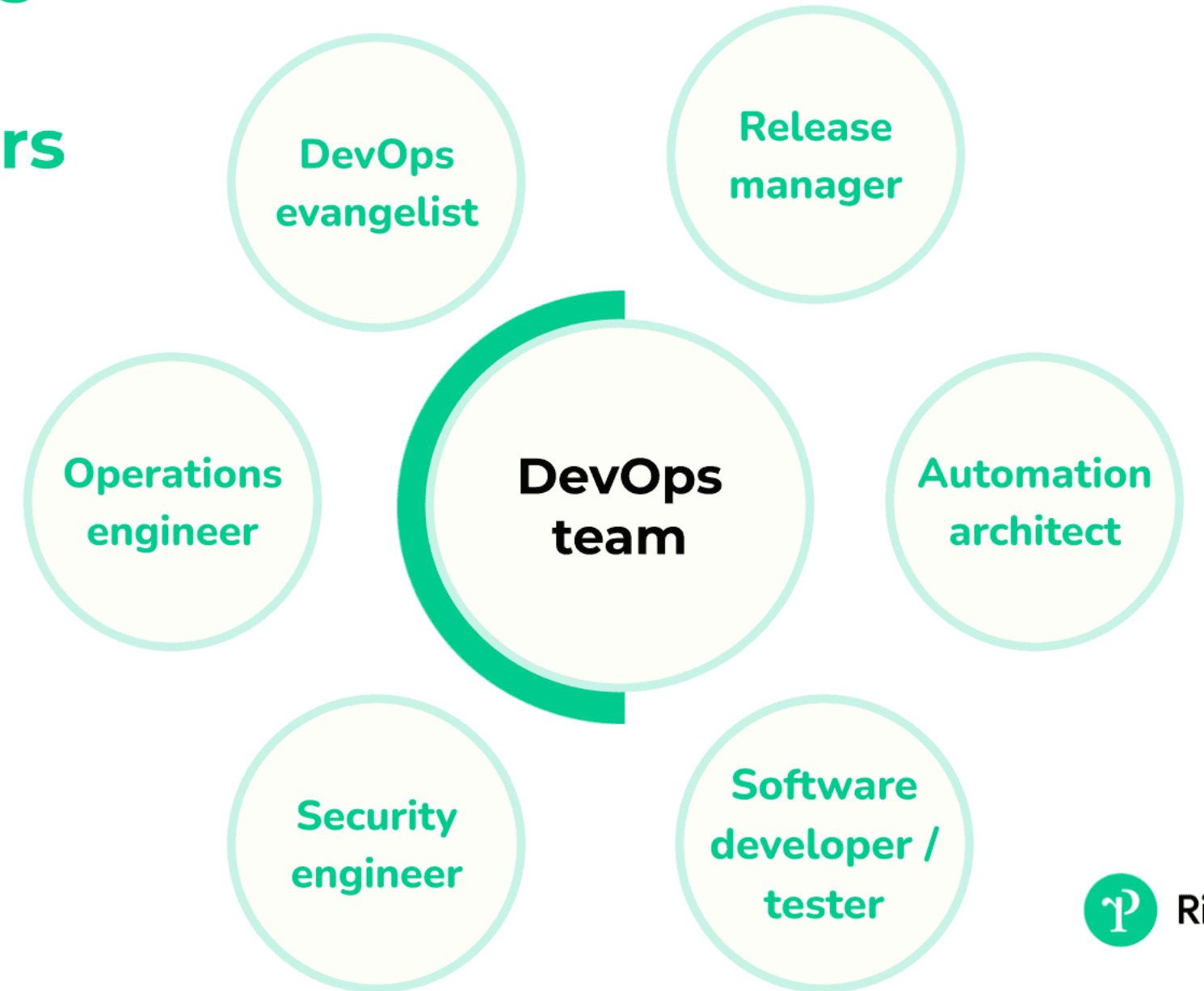
3 Roles

Product Owner

Scrum Master

Team

DevOps team members



Right People Group

Challenge ?

DevOps is about breaking down the silos that traditionally separate the development and operations teams. In the past, the development teams would write code and toss it over the wall to the operations team, who were then responsible for deployment and maintenance.

This often led to a “not my problem” mentality, with developers moving on to the next project without considering the operational implications of their code.

Now, the DevOps team bridges that gap, creating a team environment where collaboration and shared goals are the norm. The DevOps culture is where everyone works together to get high-quality software out the door faster and more efficiently.

Achieving this means mixing cultural shifts, new practices, and adopting tools that speed up the process of delivering software, all while keeping everyone on the same page.

DevOps Team Structure
Key Roles in a DevOps Team:

DevOps evangelist
Release manager
Automation architect
Software developer/tester
Security engineer (DevSecOps)
Operations engineer

1. DevOps Evangelist

Role: This team member champions the DevOps culture within the organization, working hard to get everyone on board. They bridge the gap between the operations team and the development team, highlighting the benefits of working together and fostering a spirit of continuous improvement.

Responsibilities:
•Promote DevOps culture and principles throughout the organization.
•Provide guidance on tooling, automation practices, and methodologies.
•Align business and development goals, helping teams understand the benefits of DevOps.
•Identify obstacles and ensure smooth collaboration between Dev, Ops, and other teams.

Ownership:
•Facilitating the adoption of continuous integration (CI) and continuous delivery (CD).
•Championing the value and efficiency of DevOps practices to both technical and non-technical teams.

2. Release Manager

Role: Also known as the DevOps engineer, the release manager takes charge of the release process. DevOps engineers handle the CI/CD pipeline, making sure software releases are smooth and on schedule. This role requires them to work closely with both developers and operations to address any release issues promptly.

Responsibilities:
- Plan, coordinate, and monitor the release lifecycle (from development to production).
- Ensure proper release documentation, including release notes and technical requirements.
- Facilitate communication and synchronization between multiple teams (Development, QA, and Operations).
- Manage release schedules and ensure the timely delivery of software.

Ownership:
- Overseeing release planning, deployment, and management.
- Ensuring environment readiness and rollback mechanisms are in place if needed.

3. Automation Architect

Role: The automation architect's job is to find ways to automate repetitive development and deployment tasks. This role is all about cutting down on manual work, which helps reduce mistakes and speeds up the software delivery process.

Responsibilities:
- Identify repetitive tasks in the software development lifecycle that can be automated.
- Create and maintain CI/CD pipelines.
- Ensure smooth integration of tools (Jenkins, GitLab, CircleCI, etc.) with version control systems.
- Automate the testing process and integrate automated quality checks.

Ownership:
- Responsibility for the overall automation strategy and execution.
- Creating reusable automation frameworks that are both reliable and scalable.

4. Full-Stack Contributors in DevOps

Role Overview:
Full-stack engineers in a DevOps setting align development, operations, and testing responsibilities to ensure high-velocity software delivery.

Key Responsibilities:
•**Cross-functional Expertise:** Contribute to the entire development lifecycle from frontend to backend, CI/CD pipelines, and even post-production monitoring.
Example: Developing features while configuring their deployment strategies.
•**Code Quality:** Write modular, reusable code optimized for cloud environments.
•**Collaboration:** Work alongside the DevOps team to implement configurations and resolve production bugs.

Ownership:
•Contributing to the quality of the code from both functional and non-functional perspectives.
•Ownership of unit testing, code reviews, and test automation.

Note :
In the world of DevOps, software developers and testers collaborate more closely than ever. They share responsibilities, ensuring that testing happens early and often. This teamwork leads to spotting and fixing issues sooner, which ultimately makes the software more reliable.

4. Security/DevSecOps Specialist

Role Overview:
The security engineer weaves security practices into the development and operations workflow. Starting security measures early in the software development process is their main goal, helping to prevent vulnerabilities before they become serious problems.

Key Responsibilities:

•**Security Audits:** Conduct regular code reviews, security audits, and compliance checks.
Example: Detecting vulnerabilities with tools like SonarQube or Checkmarx during development cycles.
•**Infrastructure Security:** Implement secrets management using Vault and role-based access control (RBAC) configurations in cloud providers.
•**Secure Pipelines:** Incorporate automated tools for scanning containers, dependencies, and images for vulnerabilities.
Example: Using AquaSec to scan Docker images.
•**Incident Management:** Implement procedures and simulations for handling security breaches or malware attacks.

Ownership:
•Securing infrastructure, applications, and code as part of the automation process.
•Ensuring compliance and security policies are automated in the pipeline.

5. Operations engineer

Operations engineers focus on the systems that run the software. They ensure everything is set up correctly for the software to run smoothly, from servers to databases. They also give feedback to developers to help improve the system's reliability and performance.

Responsibilities:

- **Infrastructure Management:** Setting up and managing cloud environments, physical servers, or hybrid infrastructures.
- **Automation of Provisioning and Configuration:** Automating infrastructure provisioning and configuration using Infrastructure as Code (IaC) tools (e.g., Terraform, Ansible, Puppet).
- **Monitoring and Logging:** Implementing robust monitoring and alerting systems to keep track of the health and performance of the infrastructure.
- **Incident Management:** Responding to production incidents, managing outages, and providing root cause analysis.
- **Performance Optimization:** Fine-tuning infrastructure for maximum performance and efficiency, ensuring scalability and cost-efficiency.

Ownership:

- **Infrastructure Deployment:** Responsible for deploying the necessary infrastructure and scaling it when required (managing VMs, networks, databases, etc.).
- **High Availability & Recovery:** Ensuring that the system is highly available and able to handle failure without downtime, involving backup systems, and disaster recovery strategies.
- **Operational Efficiency:** Maintaining operational best practices, driving cost-effective performance, and ensuring that SLAs and uptime are met.

Ownership Breakdown:

•**DevOps Evangelist** is the cultural driver, ensuring that DevOps is understood and adopted across the organization.

•**Release Manager** ensures streamlined and reliable releases, owning delivery workflows and schedules.

•**Automation Architect** is the technical expert, owning the automation of processes (build, deployment, etc.), ensuring scalability and reliability in the pipelines.

•**Software Developer/Tester** ensures that code is written, tested, and quality-checked, including automated testing, which guarantees functional quality and speed.

•**Security Engineer (DevSecOps)** embeds security measures across the SDLC to protect applications and infrastructure from vulnerabilities and attacks.

How they work together

During planning

This stage sets the tone for collaboration. DevOps evangelists, release managers, and security engineers get together with the development and operations teams right from the start. They aim to make sure future projects are both doable and sensible from all angles.

Key actions include:

Aligning goals: Everyone checks that the plans fit with the business's aims and what's technically doable.

Bringing security and operations in early: Security folks make sure security isn't an afterthought, aiming to spot issues before they blow up.

Keeping everyone on the same page: DevOps evangelists ensure everyone knows what's expected, helping avoid surprises later.

In development

When plans turn into actual work, developers, quality assurance, and automation architects sync up closely. In this phase, they:

Work together from the start: Testers jump in early in the coding process, helping catch problems when they're easier to fix.

Cut out the grunt work: Automation experts bring in tools to handle the repetitive bits, letting the team tackle bigger challenges.

In deployment

Getting the software out there is a big moment. Release managers, operations engineers, and automation architects make sure there are no hiccups during deployment. They focus on:

Timing deployments right: Release managers pick deployment times that cause the least disruption.

Matching production and development environments: Automation leads use code to keep environments consistent, reducing surprises.

Watching like hawks: Operations teams keep an eye on the rollout, ready to jump on any issues.

In maintenance

With the software up and running, attention turns to keeping it that way. Operations engineers lead the charge, keeping tabs on how things are going and flagging up any tweaks needed. Security engineers also stay alert to keep things tight on the security front. This involves:

Learning from real life: Operations feed back to the software development teams on any snags or areas for improvement.

Staying secure: Security teams keep the software safe against new threats, ensuring it stays compliant.

Collaboration with dedicated IT teams

DevOps works smoothly with all IT teams, whether that’s networking, databases, or customer support. For instance:

With networking teams: To make sure the infrastructure changes needed for new deployments are spot on.
With database teams: To align database updates with app changes, keeping everything running smoothly.
With helpdesk teams: User issues flagged by support can guide tweaks and improvements in development.

In short, DevOps practices are about creating a unified environment where software is developed, deployed, and maintained efficiently and securely. This teamwork not only speeds things up but also boosts the software’s reliability and security, leading to a better experience for everyone involved.

Understanding Team Dynamics and alignment

In a DevOps setting, the effectiveness of the entire software lifecycle relies heavily on the team dynamics and alignment of the various roles. Unlike traditional siloed approaches, where development, operations, security, and testing teams work in isolation, DevOps emphasizes breaking down barriers, creating cross-functional teams, and encouraging collaboration across different roles.

Scenario: Building and Launching an E-commerce Platform

The development of a large, scalable e-commerce platform requires collaboration between all teams—**development**, **testing**, **operations**, **security**, **product management**, and others. The platform needs to handle high traffic volumes while ensuring high performance, security, and stability. Modern **DevOps** practices are key, ensuring that everything is automated—from development, testing, security integration, to deployment—and continuous integration and delivery (CI/CD) drive the workflow.

Workflow Breakdown of the DevOps Process:

1. Initial Requirements & Planning

•**Collaboration with Product Team**: The **Product Manager** works closely with stakeholders and users to understand the platform's needs—such as handling product browsing, shopping carts, secure checkout, payment gateway integration, user authentication, and order management.
•The **Scrum Master** ensures the team adheres to Agile practices, helping the development team define the **product backlog**, prioritize tasks, and set up a **two-week sprint cycle**.
Tools: Jira, Confluence, Slack

2. Development of Initial Features

•**Developers** begin coding the initial features (such as user authentication and product display) by breaking them into smaller tasks. The main goal is to create modules that can later be integrated with other features, including payment gateways and user reviews.
•Developers use **version control systems** (e.g., **Git** or **GitLab**) to commit their changes and manage code versions, ensuring a smooth integration process as the feature set grows.
Tools: Git, GitLab, Bitbucket, IntelliJ, VS Code

3. CI/CD Integration & Automated Unit Testing

•Automated Unit Testing: The developers write unit tests for each feature—such as testing the addProductToCart() method for adding products and calculating prices using a framework like JUnit or Jest.
•Once committed, the **CI pipeline** automatically triggers. Jenkins, GitLab CI, or CircleCI initiates the pipeline, where unit tests are automatically executed on the committed code to check for errors, regressions, or potential failures.
•If tests pass, the **build process** is started—compiling code into runnable artefacts that can be deployed.
Tools: Jenkins, GitLab CI, CircleCI, JUnit, Jest, Postman (for API tests)

4. Secure Coding Practices & Code Quality Review

•**DevSecOps Integration**: Security is implemented from the very start. As developers write code, they also integrate **Static Application Security Testing (SAST)**, utilizing tools like **SonarQube** to scan for potential vulnerabilities in the codebase, including SQL injection or cross-site scripting (XSS) attacks.
•**Security Reviews**: Developers ensure input validations and sanitize user input before committing. **Security engineers** conduct a **review** using code scanning tools like **Snyk** or **Checkmarx** to detect vulnerabilities early in the development phase.
Tools: SonarQube, Checkmarx, Snyk

5. Integrating with Third-Party APIs

•As the development team adds functionality (e.g., shopping cart, order tracking, payment gateway), they use **API integration** for services such as **payment processing** (e.g., Stripe, PayPal).
•This integration is tested by developers and testers who use **mocked API services** to simulate the third-party service behavior during the development phase.
Tools: Postman, Swagger, Mockoon

6. Full Feature Testing (Unit, Integration, End-to-End)

•**Test Automation by Testers**: With the developers' feature complete and committed to the repository, **QA teams** begin building **integration tests**. They confirm that the entire checkout system, for example, works as intended—testing how each module (cart, payment, checkout) interacts.
•**End-to-End (E2E) Testing** is performed to simulate the full user experience (from browsing products to completing the checkout and payment) using automated test suites.
•Tools like **Cypress** and **Selenium** are used for testing **E2E**, ensuring that the product behaves correctly when accessed by actual end users, simulating how users interact with the application on different devices.
•**DAST (Dynamic Application Security Testing)** is also part of the testing cycle, where **DAST tools like OWASP ZAP** probe the deployed application to ensure no security vulnerabilities are left unchecked.
Tools: Cypress, Selenium, OWASP ZAP, Burp Suite

7. Staging & Pre-production Testing

•**Staging Deployment**: Once the tests in the **CI pipeline** pass, the code is pushed into a staging environment that mirrors production (running on the cloud). This environment serves as a testing ground for **integration with third-party services**, user experience validation, and full-performance tests.
•**Load Testing & Stress Testing** is executed in the staging environment with tools like **JMeter** to ensure that the platform can scale and handle traffic surges, particularly during peak shopping seasons or promotions.
•The application is monitored for **bottlenecks** and optimized for **CPU usage** and **memory consumption** during load tests.
Tools: JMeter, LoadRunner, Grafana, Prometheus

8. Final Security Testing

•Before moving to **production**, an additional layer of security testing occurs. The security engineers perform **penetration testing** and ensure that **DAST tools** (like **OWASP ZAP**) are ran against the staging app to uncover any security vulnerabilities.
•**Penetration Testing (manual)**: During this phase, security engineers may manually attempt to breach the application's security, mimicking possible hacker behavior, scanning for weaknesses like cross-site scripting (XSS) or improper encryption methods.
Tools: OWASP ZAP, Burp Suite, manual security audits

9. Continuous Deployment (CD) & Rollout to Production

•Upon passing all tests and security reviews, the **Release Manager** schedules the application deployment to the **production environment**. This stage typically begins by automatically deploying to a **Canary Release** environment for **incremental traffic** exposure to detect potential issues before a full production rollout.

•The deployment process uses automated **CI/CD pipelines** built with **Jenkins**, **ArgoCD**, or **GitLab CI**, ensuring seamless push from **Git repositories** to **Kubernetes containers**, leveraging **Docker** to containerize the application.

•**Monitoring & Observability** tools like **Prometheus** and **Grafana** are deployed alongside the application to track **real-time user data**, app crashes, server load, and security vulnerabilities.

•The platform is actively **monitored** post-deployment, alerting the DevOps team in case issues arise (e.g., crashes, performance degradation).

Tools: Jenkins, GitLab CI, ArgoCD, Docker, Kubernetes, Prometheus, Grafana

10. Post-Production Monitoring and Maintenance

•**Observability & Monitoring**: Continuous monitoring via **logging platforms** such as **ELK stack (Elasticsearch, Logstash, Kibana)** or **Splunk** ensures that operational issues are identified and resolved without impacting users.

•Alerts are configured for critical production failures like sudden drops in **service performance**, **failure in the payment API**, or **increased error rates**.

•Developers fix any **issues or bugs** reported in production, which automatically triggers a new cycle of **CI/CD**, which involves rebuilding, retesting, and deploying.

•**Feedback Loops with Product Team**: **Customer feedback** and **usage analytics** (collected through A/B testing tools and user activity tracking) are sent back to the **product manager** to refine existing features and plan new ones.

Tools: Elasticsearch, Splunk, Prometheus, Grafana, Datadog

11. Continuous Improvement and Iteration

•Over time, **new features** are added to the e-commerce platform based on feedback and changing market conditions (e.g., customer reviews, payment methods, new security standards).

•The **CI/CD pipeline** becomes more complex, implementing newer technologies like **microservices** and **service meshes**. More advanced **automated tests** are created as features evolve.

•**Feature toggles** are used for selective activation of new functionality, making incremental deliveries safe by decoupling feature releases from code deployments.

Tools: FeatureFlags (LaunchDarkly), Jenkins, Kubernetes, Docker

3 Ceremonies

Sprint Planning

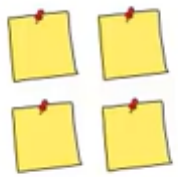
Daily Scrum

Sprint Review

Scrum Workflow

Product Backlog

Sprint Planning



Sprint Backlog

Sprint

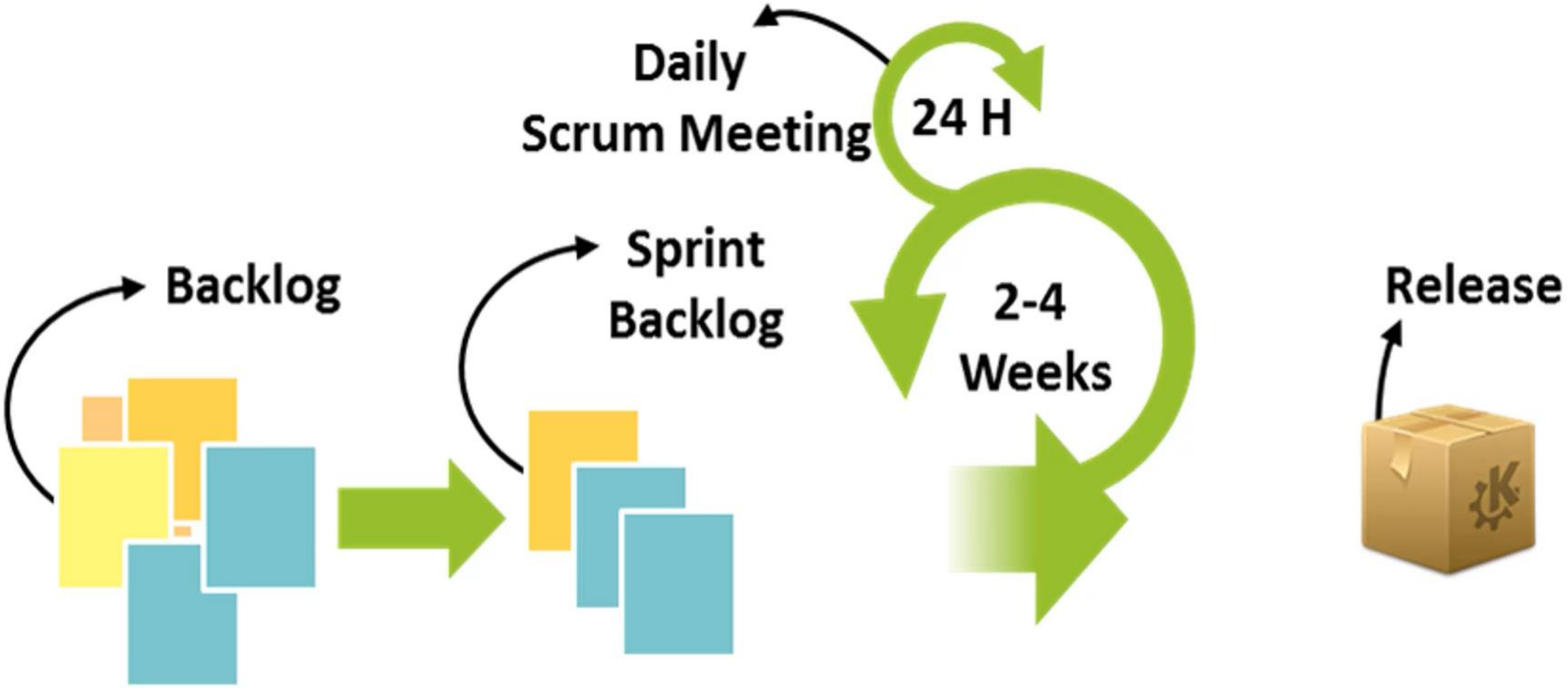
Potentially Shippable Product

Sprint Review

Retrospective

Repeat this workflow for each sprint

Scrum flow


Daily
Scrum Meeting
24 H
Backlog
Sprint
Backlog
2-4
Weeks
Release

Jenkins

PUSH

GitHub

BUILD

Maven

TEST

Maven

Se

DEPLOY

Apache TOMCAT

MONITOR

Jenkins

PIPELINE CREATION