innovate achieve lead

BITS Pilani
Pilani Campus

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

By Prof A R Rahman
CSIS Group WILP

innovate achieve lead

BITS Pilani
Pilani Campus

Course Name: Introduction to DevOps
Course Code : CSI ZG514/SE ZG514

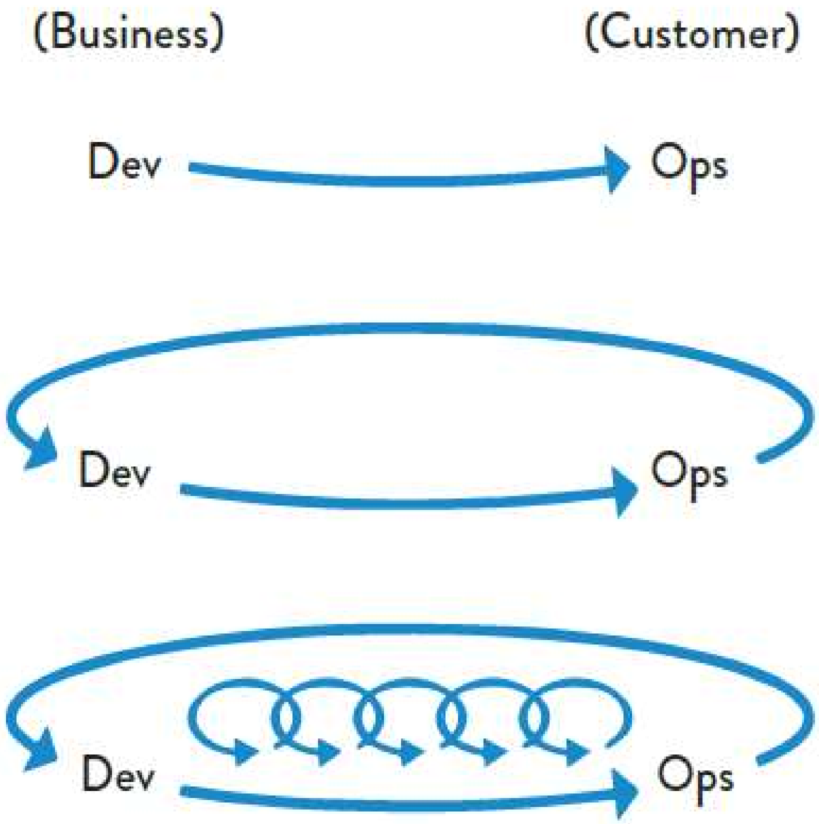
By Prof A R Rahman
CSIS Group WILP

CS – 3 DevOps Dimensions

Coverage

- Three dimensions of DevOps – People, Process, Technology/Tools
- DevOps- Process
- DevOps and Agile
- Agile methodology for DevOps Effectiveness
- Flow Vs Non-Flow based Agile processes
- Choosing the appropriate team structure: Feature Vs Component teams
- Enterprise Agile frameworks and their relevance to DevOps
- Behaviour driven development, Feature driven Development
- Cloud as a catalyst for DevOps

The Three Ways: The Principles Underpinning DevOps



DevOps

- DevOps is a way of thinking and a way of working.

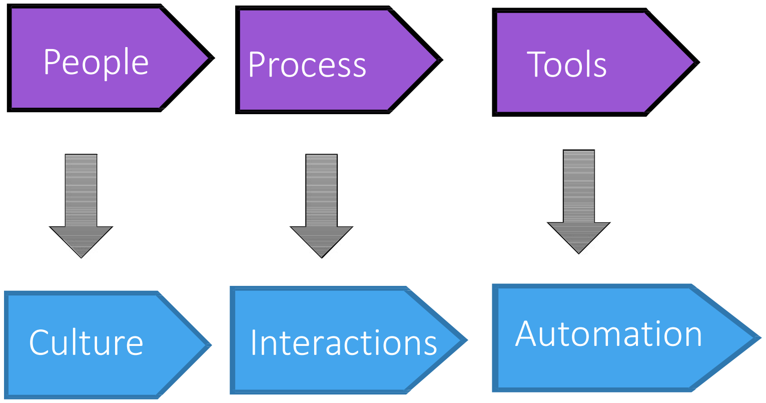
- DevOps spans all stakeholders in an organization, including business owners, architecture, design, development, quality assurance (QA), operations, security, partners, and suppliers.

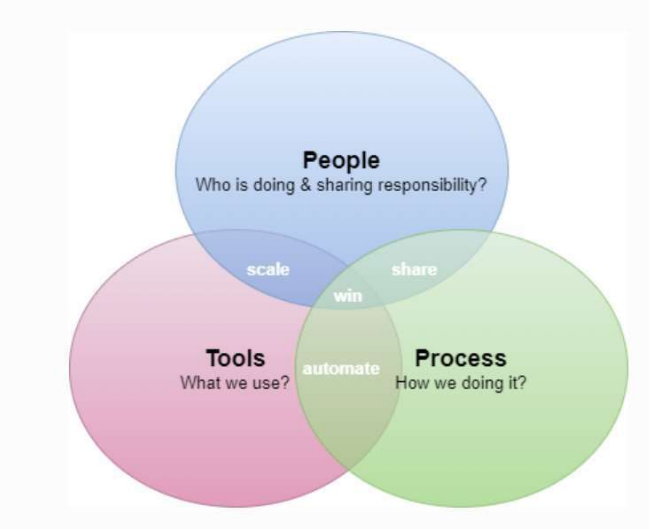
- It is a framework for enabling people and teams to practice their crafts in effective and lasting ways.

DevOps

- Adopting any new capability typically requires a plan that spans people, process, and technology.
  - Making people more efficient
  - Streamlining processes
  - Choosing the right tools

DevOps





- Process:

  Sequence or layers of procedures that is consisted of people, machines, materials, methods and governing policies that are used all together to design and delivery a good or service (product). Conversion of an input into an output through-out procedures.

- Practice:

  Series of steps to achieve a consistent result (output).

- A simple process may be described by a single procedure. But a more complex process, will have multiple procedures.

Dimensions of DevOps

Three dimensions of DevOps

People

- Initially, stakeholders would need to be identified, ensuring that everybody delivering value to the business are working tightly together on the common goal of adding value to the customer.
- Highly motivated people with good collaboration is necessary

Process

- Process of designing, building and testing software should be well presented to each individual team member
- Processes and practices that help improve the value and delivery to customers

Tools

- Tools and services that can help enable different DevOps practices and different teams which can be used to make things easier

DevOps– People

- Addresses the people aspect of adopting DevOps, including creating the necessary culture.

- An organization may adopt the most efficient processes or automated tools possible, but they’re useless without the people who eventually must execute those processes and use those tools.

DevOps– People

- DevOps CULTURE
- (1) Identifying business objectives
  - Creating a culture is getting everyone headed in the same direction and working toward the same goal
  - Identifying common business objectives for the team and the organization as a whole.
  - identify goals that it wants to achieve; then it can develop a common set of milestones toward those goals for different teams of stakeholders to use
  - Collaboration and communication across stakeholders

DevOps– People

- (2) Create an environment of sharing
  - leaders of the organization to work with their teams to create an environment and culture of collaboration and sharing.
  - Leaders must remove any self-imposed barriers to cooperation.
  - Typical measurements reward operations teams for uptime and stability, and reward developers for new features delivered, but they pit these groups against each other.

DevOps– People

- (2) Create an environment of sharing
  - The leaders of the organization should further encourage collaboration by improving visibility.
  - Establishing a common set of collaboration tools is essential, especially when teams are geographically distributed and can’t work together in person.
  - Giving all stakeholders visibility into a project’s goals and status is crucial for building a DevOps culture based on trust and collaboration.

DevOps– People

- DEVOPS TEAM

  - The arguments for and against having a separate DevOps team are as old as the concept itself.

  - DevOps liaison teams, which resolve any conflicts and promote collaboration.

  - Such a team may be an existing tools group or process group, or it may be a new team staffed by representatives of all teams that have a stake in the application being delivered.

  - DevOps team, your most important goal is to ensure that it functions as a Center of Excellence that facilitates collaboration without adding a new layer of bureaucracy or becoming the team that owns addressing all DevOps related problems

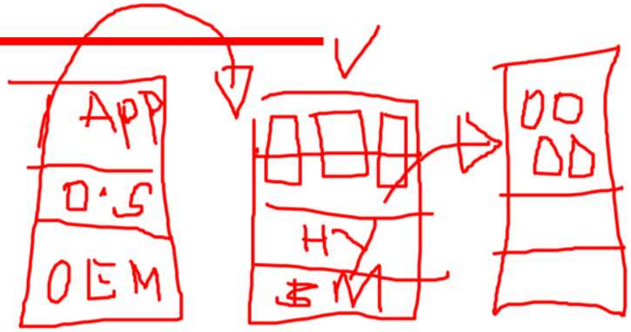
DevOps– People

Key competencies to succeed

- Culture – Shared organizational assumptions to resolve problems
- Leadership
  - visioning, strategic management, flexibility, and the ability to inspire others to innovate and perform
  - individuals with advanced communications skills, the knowledge of diverse cultures, and people who behave collaboratively when working in teams

DevOps– People

Key competencies to succeed



- Resource/management
  - effective management of available resources
  - collaboration across multi-functional teams is at the heart of DevOps, delivering an accountable business outcome

- Subject matter experts (SME's) in different areas like – Automation, Build & Release management, CI/CD, Containers, Security, Cloud (AWS/GCP/Azure)
- DevOps engineers who are members from different department (development, QA, Operations)

DevOps– Process

- Processes define what those people do
- organization can have a great culture of collaboration, but if people are doing the wrong things or doing the right things in the wrong way, failure is still likely.
- KEY PROCESSES
- (1) As a Business Process
- Makes the business more agile and improves its delivery of capabilities to customers.
- Business Process defines collection of activities or tasks that produces a specific result (service or product) for customers.

DevOps– Process

- KEY PROCESSES (Contd..)
- (1) As a Business Process
- DevOps business process involves taking capabilities from the idea (typically identified with business owners) through development and testing to production.
- Business process isn’t mature enough to be captured in a set of simple process flows.
- Need to identify areas of improvement , both by improving the processes themselves and by introducing automation

DevOps– Process

- (2) Change management process

- Set of activities designed to control, manage, and track change by identifying the work products that are likely to change and the processes used to implement that change.

- Traditional change management approaches tend to be limited to change request or defect management, with limited capability to trace the events between the change requests or defects and the associated code or requirements.

- Requires all stakeholders to be able to view and collaborate on all changes across the software development life cycle

DevOps– Process

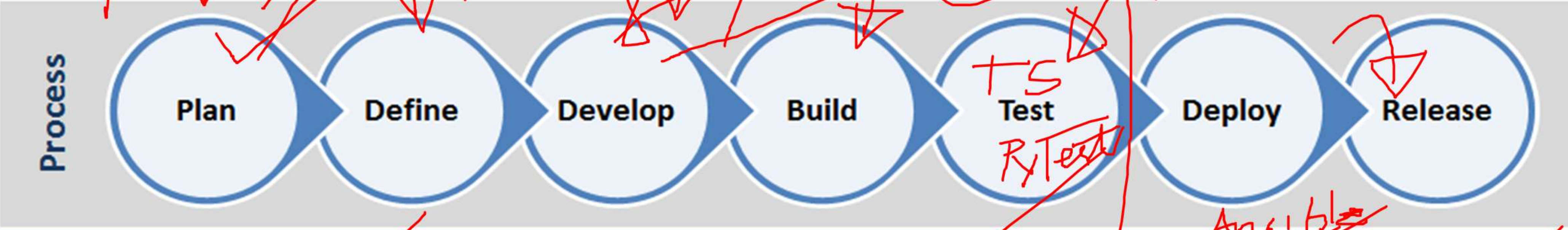
- (2) Change management process

- Organization uses is an inherent part of the broader DevOps process flow.

- Change management should include processes
  - ✓ Work-item management
  - ✓ Configurable work-item workflows
  - ✓ Project configuration management
  - ✓ Planning (agile and iterative)
  - ✓ Role-based artifact access control

- Includes processes that enable the enterprise to link work items to all artifacts, project assets, and other work items that are created, modified, referenced, or deleted by any practitioner who works on them.

DevOps– Process



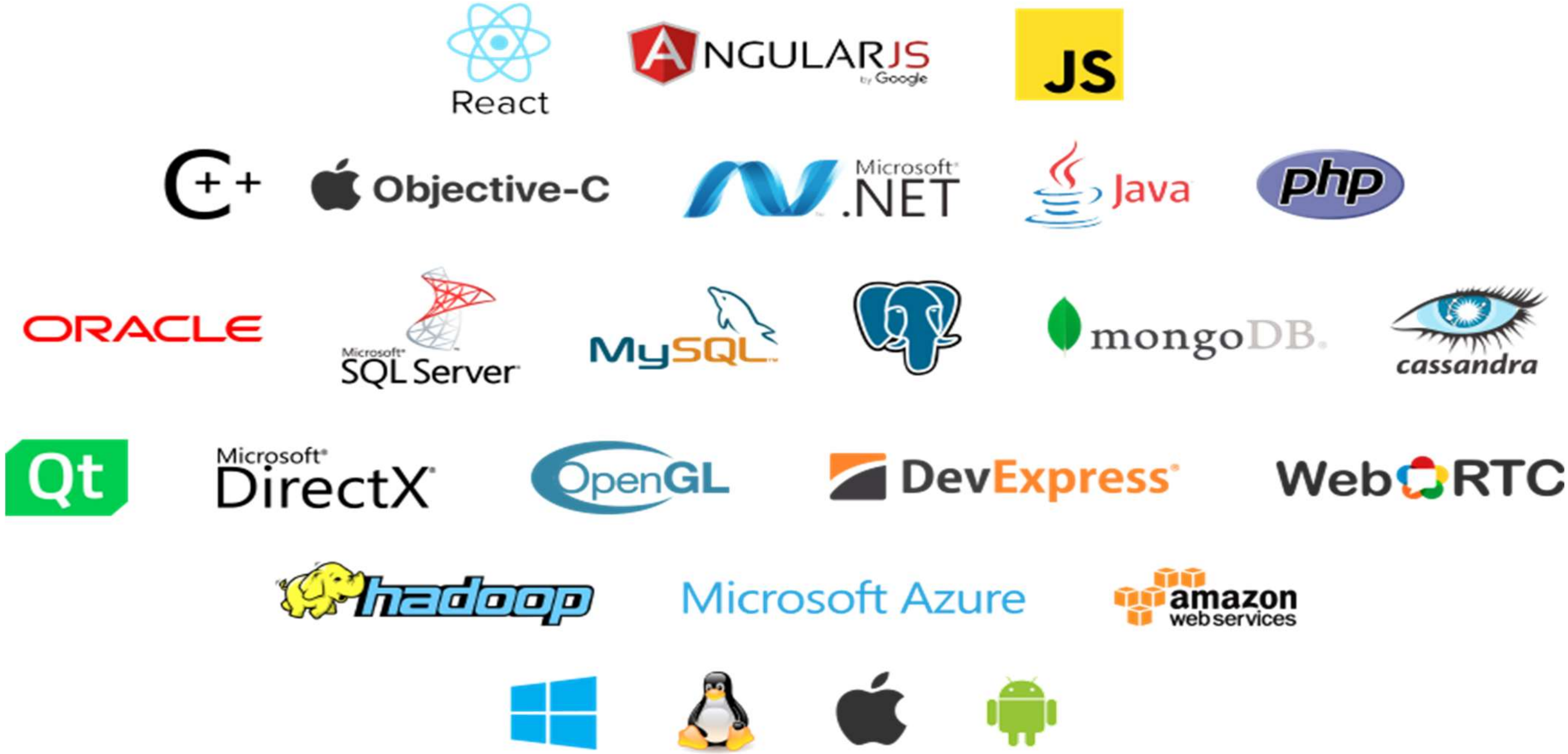
Tools						
Wiki	Sharepoint Jira HP ALM	IntelliJ GIT/Stash Maven OpenGrok	Sonar Jenkins Maven	Selenium Cucumber HP ALM HP LeanFT Load Runner Fortify	Chef Puppet Docker Nexus	Nexus Puppet

Ansible

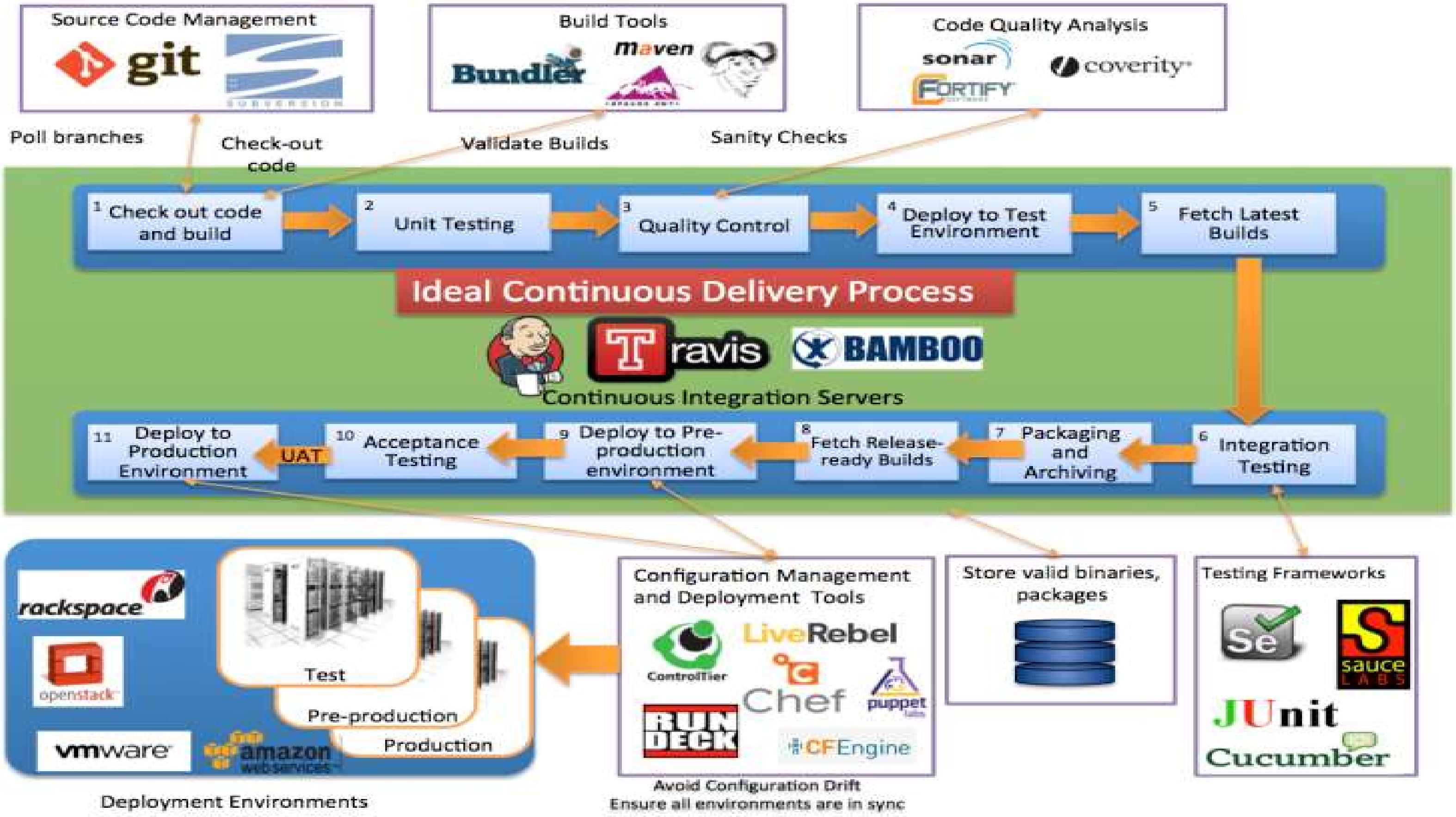
DevOps– Tools

Sifter
YouTrack
TestRail
Atlassian JIRA
waffle.io
capsule
trac
Integrated SCM & Project Management
COUNTERSOFT
Teamweek
esa LLC.
Wrike
Wunderlist
unfuddle
Trello
GQueues
salesforce
habitica
Podio
Pivotal Tracker
producteev
Eventum
GECKOBOARD
Atlassian Bitbucket
GitHub
Worksection
teamwork.com
PUSHPIN planner
breeze
TAIGA
asana
Basecamp®
assembla
Bugzilla
REDMINE
FRESHBOOKS cloud accounting

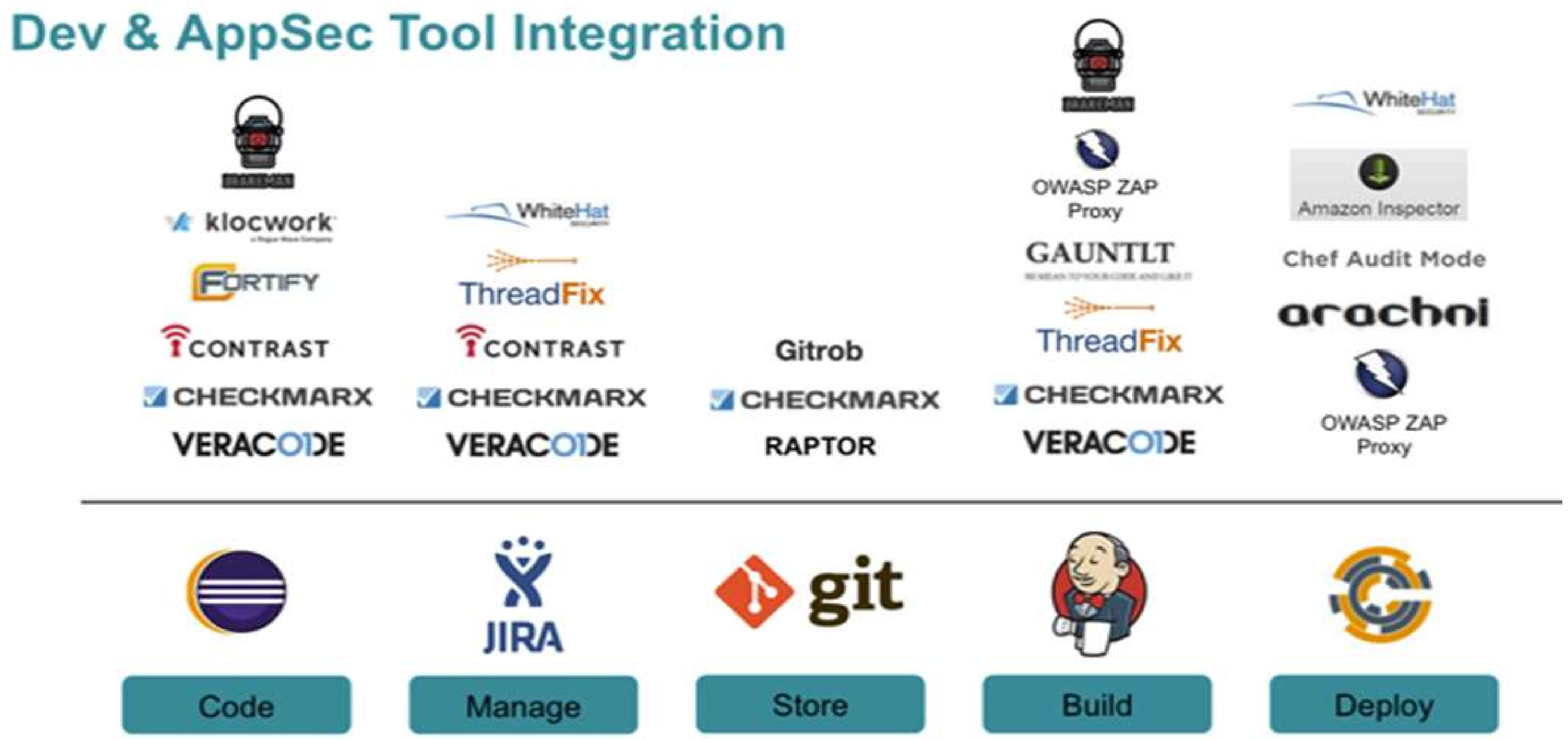
DevOps– Tools



DevOps– Tools



DevOps– Tools

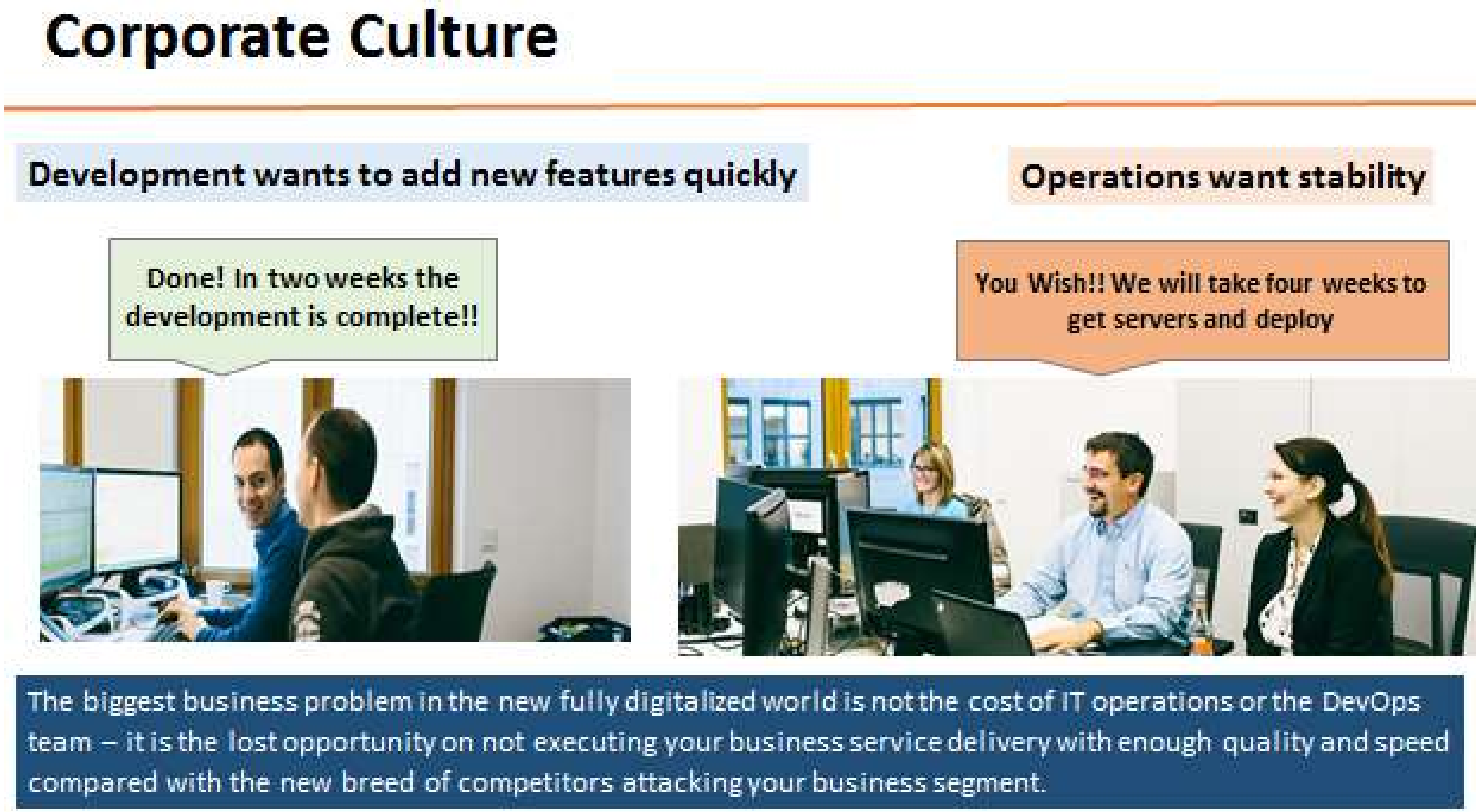


DevOps –Tools example

- Infrastructure as code — Ansible, Puppet, Chef
- CI/CD — Jenkins, Shippable, Bamboo
- Test automation — Selenium, Cucumber, Apache JMeter
- Containerization — Docker, Rocket, Unik
- Orchestration — Kubernetes, Swarm, Mesos
- Deployment — Elastic Beanstalk, Octopus, Vamp
- Measurement — NewRelic, Kibana, Datadog
- ChatOps — Hubot, Lita, Cog
- AWS => Code Commit, Code deploy ,code pipeline, Beanstack, cloud formation, AWS workflow

/ IaC
↳ TF → Open Tofu
→ Pulumi

Dispelling DevOps Myths



Corporate Culture

Development wants to add new features quickly

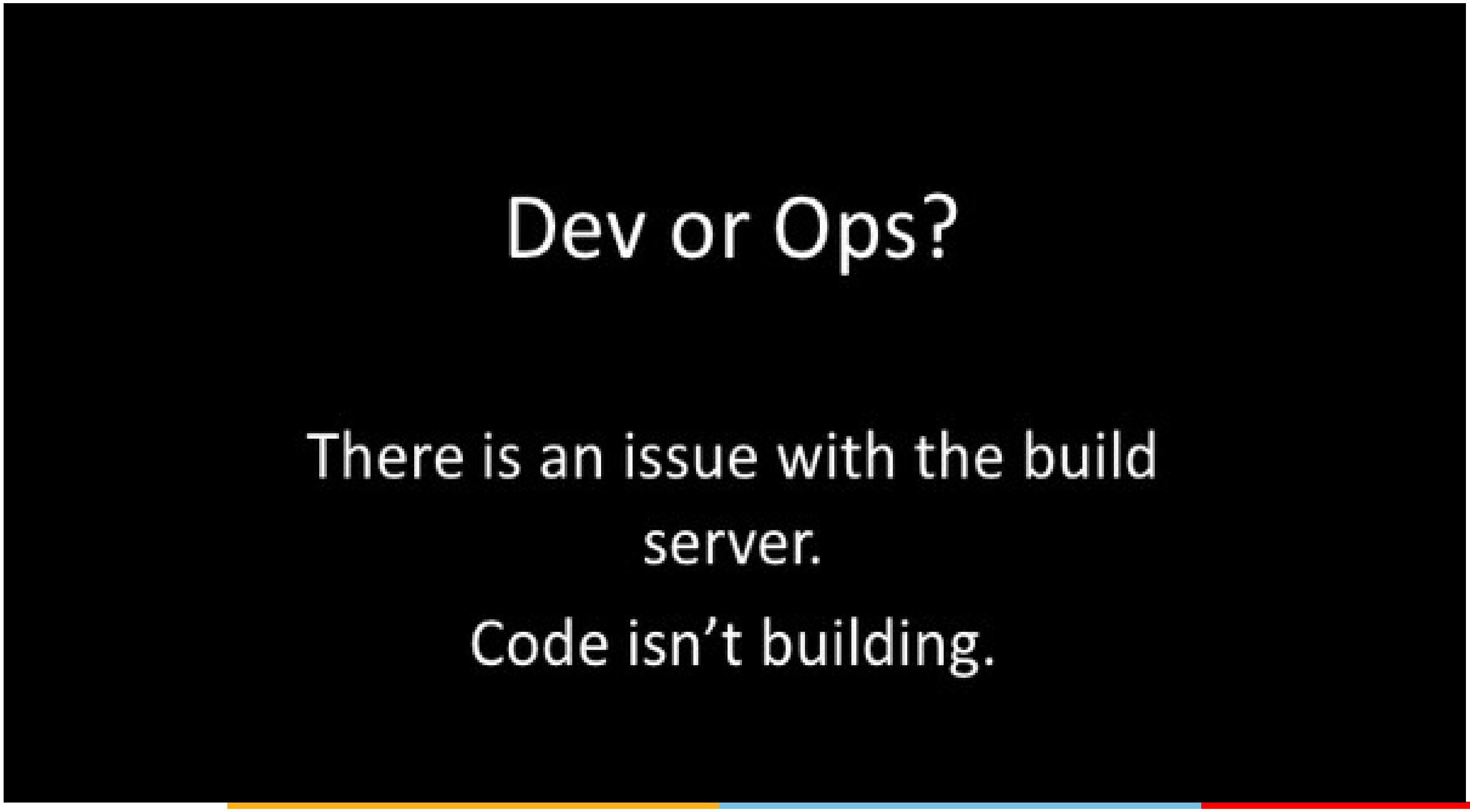
Operations want stability

Done! In two weeks the development is complete!!

You Wish!! We will take four weeks to get servers and deploy

The biggest business problem in the new fully digitalized world is not the cost of IT operations or the DevOps team – it is the lost opportunity on not executing your business service delivery with enough quality and speed compared with the new breed of competitors attacking your business segment.

Dispelling DevOps Myths

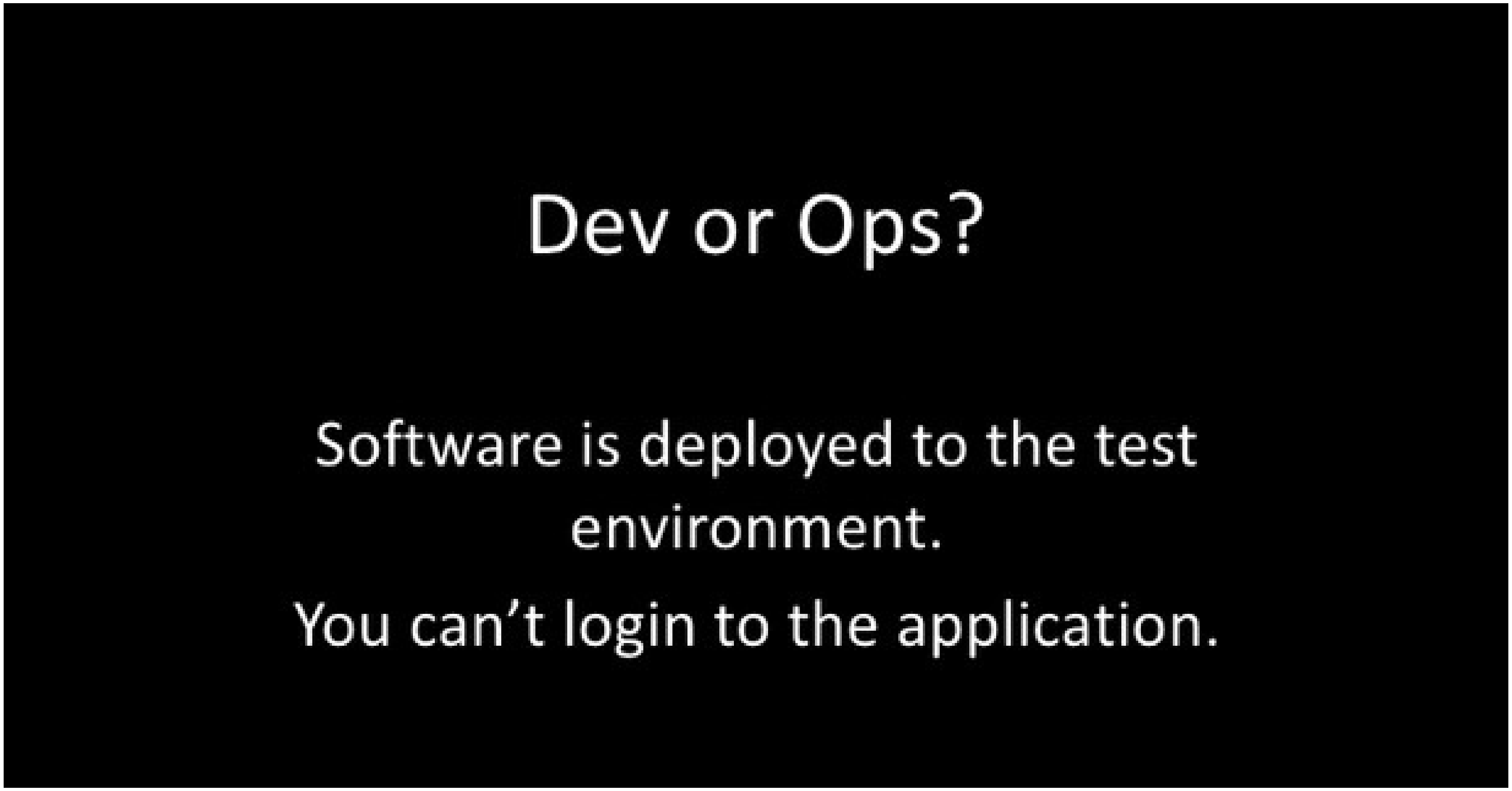


Dev or Ops?

There is an issue with the build server.

Code isn’t building.

Dispelling DevOps Myths

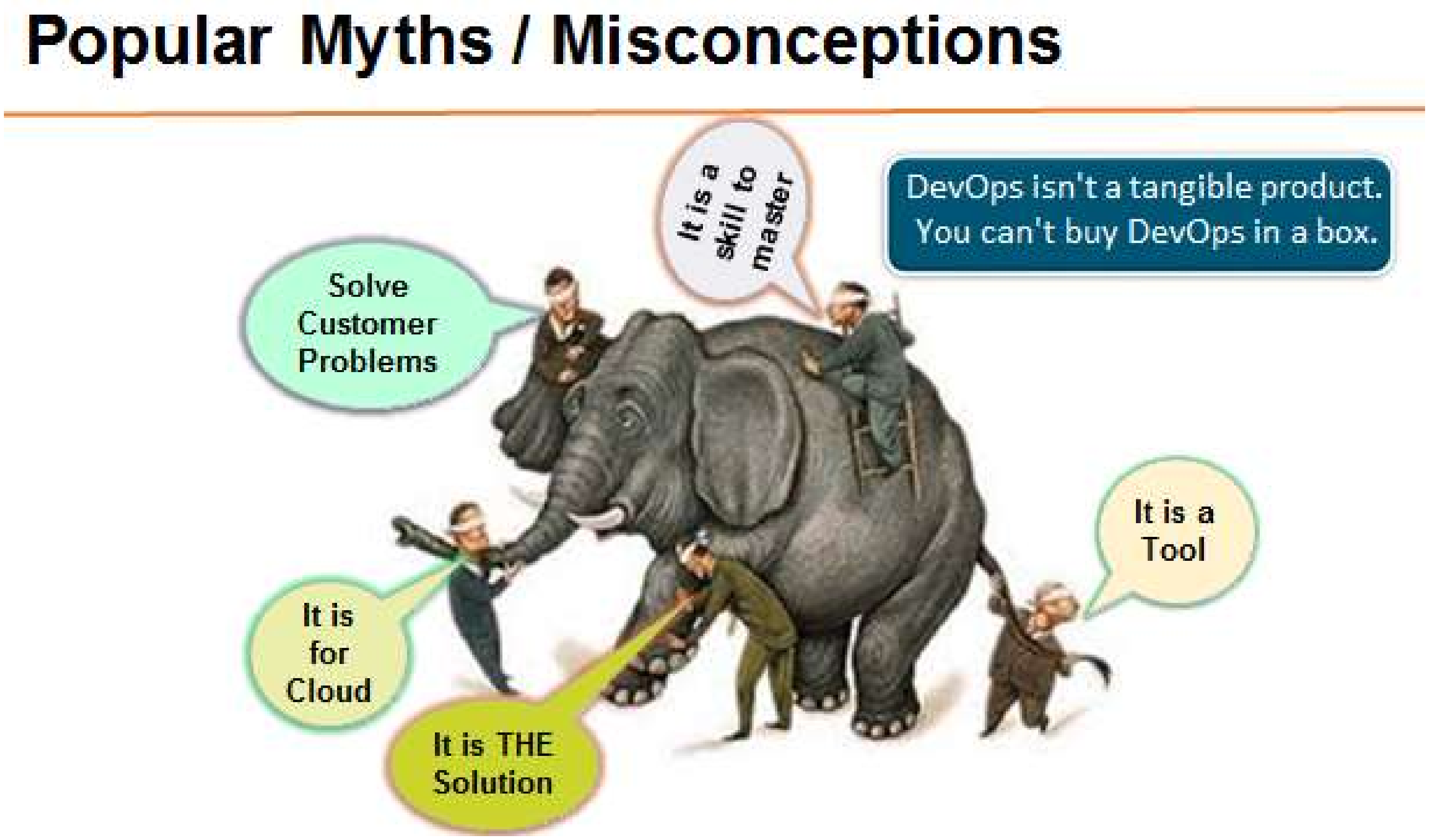


Dev or Ops?

Software is deployed to the test environment.

You can’t login to the application.

Dispelling DevOps Myths



Popular Myths / Misconceptions

DevOps isn't a tangible product. You can't buy DevOps in a box.

Dispelling DevOps Myths

You Need a DevOps Guy…

DevOps is a methodology, not a skill. It is something you practice as a team not an individual.

Dispelling DevOps Myths

DevOps supports any SDLC…

DevOps with legacy Waterfall or DevOps without Continuous Quality has a high risk of failure or not meeting customer satisfaction expectations.

Dispelling DevOps Myths

DevOps Clashes with Existing Processes…

DevOps smoothly integrates with existing processes like ITIL, Agile etc.
DevOps is actually a way to improve processes like ITIL.

Food for Thought -

- What is the life cycle model followed in your project?
- What is your role name?
- What are the DevOps tools you use?
- Please share your experience

DevOps & Agile

- DevOps emphasizes the relationship of DevOps practices to agile Practices
- Agile planning organizes work in short iterations (e.g. sprints) to increase the number of releases
- Team has only high-level objectives outlined, while making detailed planning for two iterations in advance.
- This allows for flexibility and pivots once the ideas are tested on an early product increment.
- Close the gap between requirement and build/test

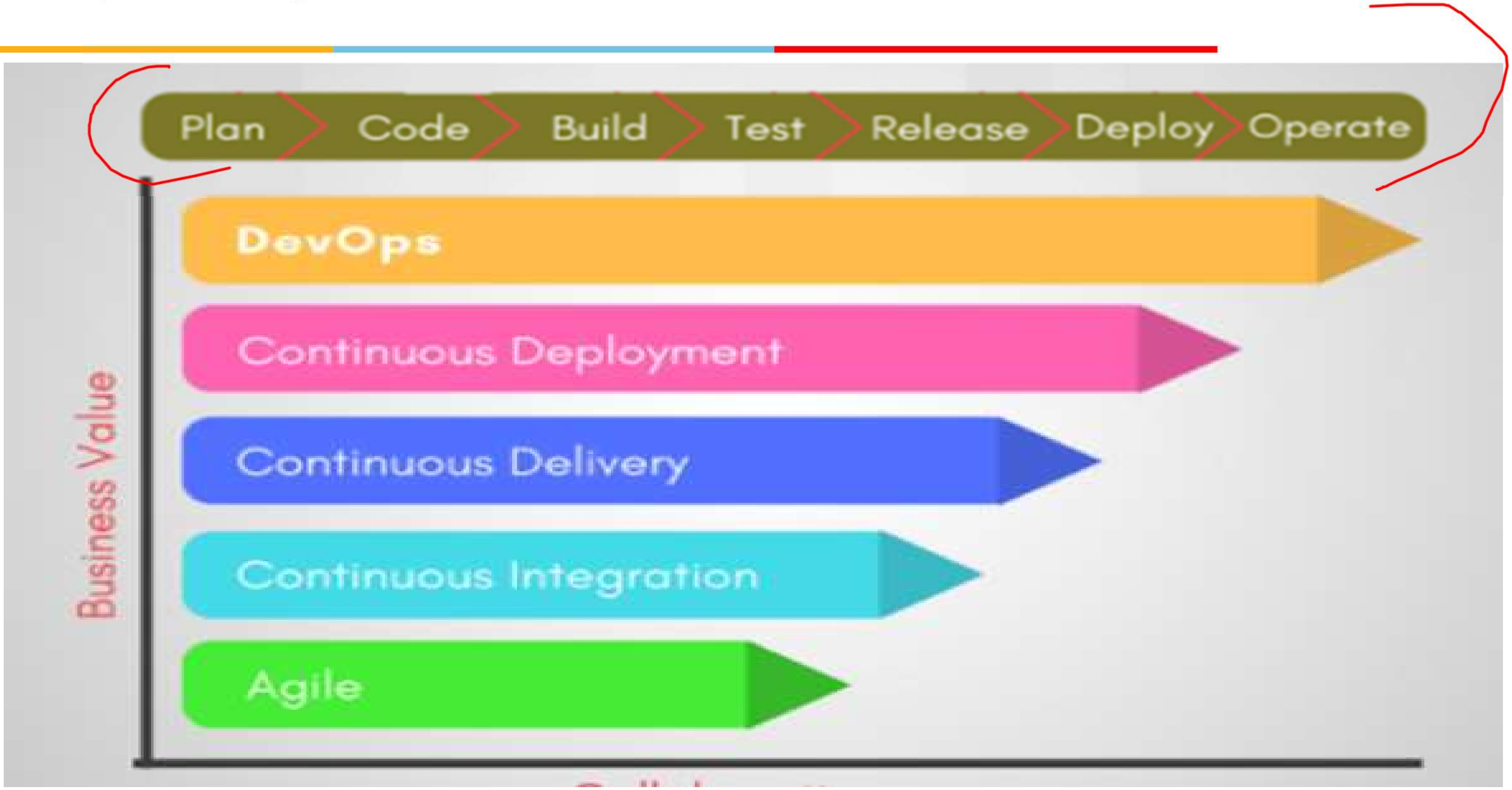
DevOps and Agile

- Agile refers to an iterative approach which focuses on collaboration, customer feedback, and small, rapid releases.
- DevOps is considered a practice of bringing development and operations teams together.
- Agile helps to manage complex projects.
- DevOps central concept is to manage end-to-end engineering processes

DevOps – Development + Operations

- Bamboo, Go, Jenkins, TeamCity.
- Offers workflow-based integration/build and customizable.
- upload ,manipulation, unit testing ,static analysis, compile, integration test

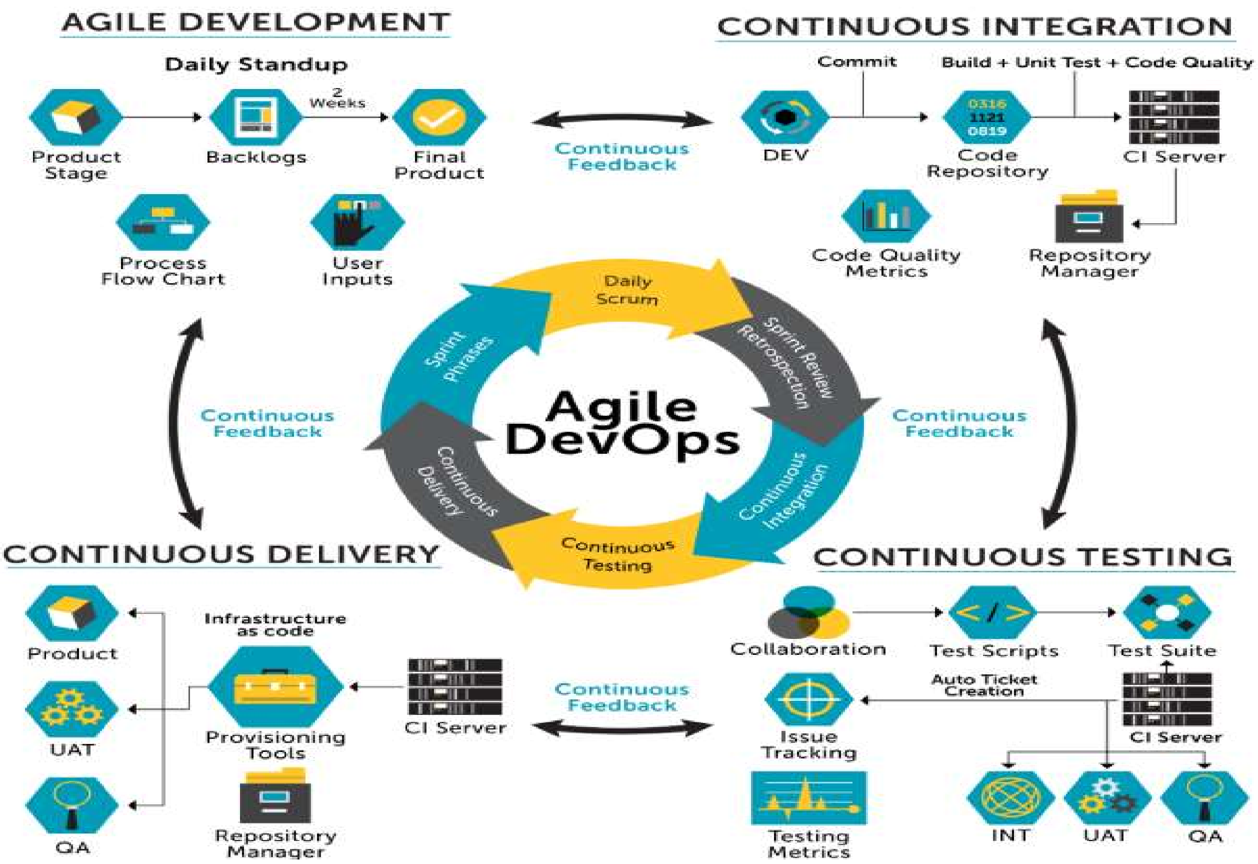
DevOps & Agile



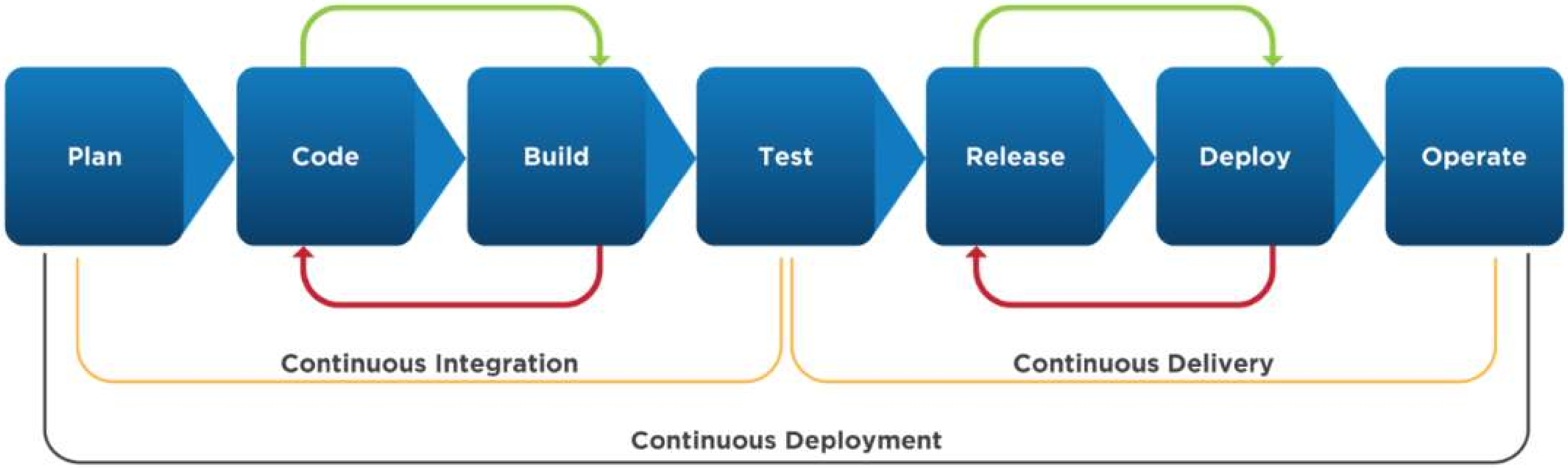
DevOps – Model & Practices

- DevOps requires a delivery cycle that comprises planning, development, testing, deployment, release, and monitoring with active cooperation between different members of a team.
- Core practices that constitute the DevOps
  - Agile planning
  - Continuous Integration
  - Continuous Inspection
  - Continuous Delivery
  - Continuous Deployment
  - Continuous monitoring
  - Automation
  - Infrastructure as a code
  - Containerization
  - Microservices
  - Cloud infrastructure

DevOps – Model & Practices



DevOps- Model & Practices



Planning
- Requirement finalization
- Updates & new changes
- Architecture & design
- Task assignment
- Timeline finalization

Code
- Development
- Configuration finalization
- Check-in source code
- Static-code analysis
- Automated review & peer review

Build
- Compile code
- Unit testing
- Code-metrics
- Build container images or package
- Preparation or update in deployment templated
- Create or update monitor dashboards

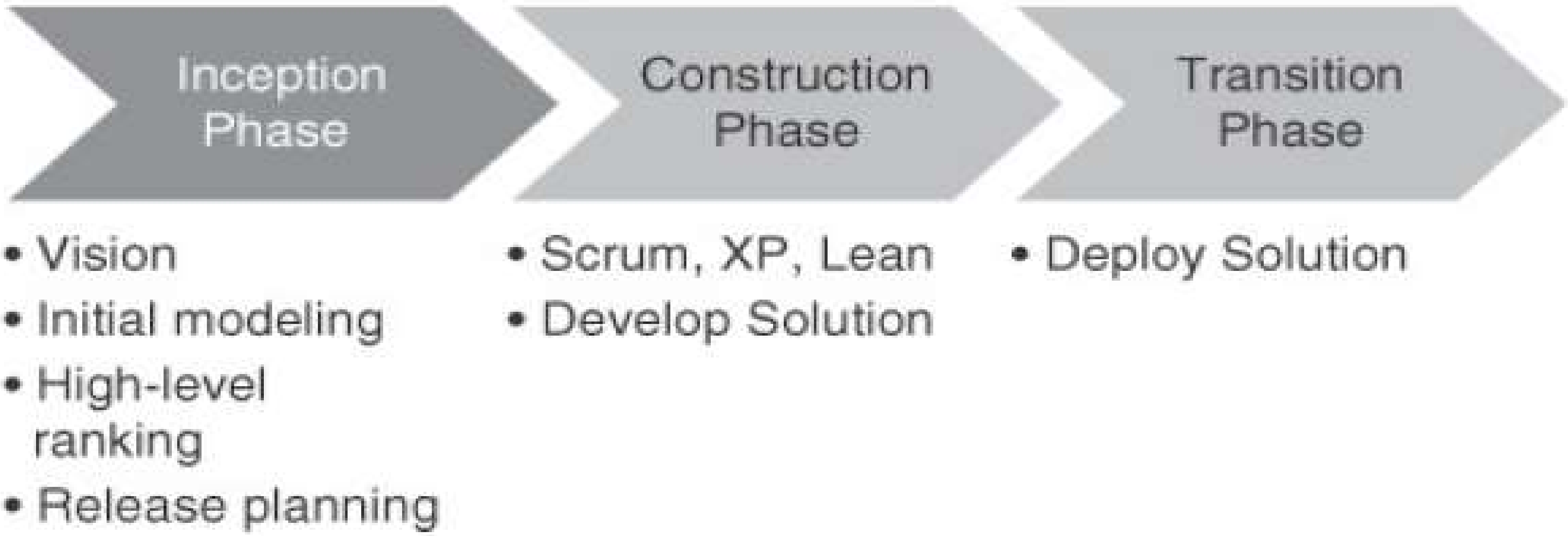
Test
- Integration test with other component
- Load & stress test
- UI testing
- Penetration testing
- Requirement testing

Release
- Preparing release notes
- Version tagging
- Code freeze
- Feature freeze

Deploy
- Updating the infrastructure i.e staging, production
- Verification on deployment i.e smoke tests

Operate
- Monitor designed dashboard
- Alarm triggers
- Automatic critical events handler
- Monitor error logs

Agile methodologies for DevOps effectiveness



Inception Phase

- Vision
- Initial modeling
- High-level ranking
- Release planning

Construction Phase

- Scrum, XP, Lean
- Develop Solution

Transition Phase

- Deploy Solution

Agile methodologies for DevOps effectiveness

- Disciplined Agile Delivery phases for each release.
  - Inception phase - Release planning (Feature prioritization, scheduling) and initial requirements
  - Construction phase - management of the code branches, the use of continuous integration and continuous deployment
  - Transition phase - solution is deployed and the development team is responsible for the deployment, monitoring the process of the deployment, deciding whether to roll back and when
  - Choosing appropriate Team Size - size of the team should be relatively small
  - Team Roles - Team lead (Scrum Master) Team member (Developer)

Flow-Based Agile

Flow based Agile

- Team assesses the features backlog and then decide on which feature to work on based on their available capacity.

- Since there is no fixed time interval for releases, the team together with the stakeholders will need to decide on a schedule for planning, review and testing

Non – Flow Based Agile

Non – Flow based Agile

- Requires an initial prioritization of features, which then allows the team to start work by progressing from the highest priority feature down to the lowest.
- The iterations are regularly spaced out in constant, equal intervals (thus the name) which then builds up a cycle that the team becomes familiar with.
- A single iteration may have multiple features, but features will be carried over to the next iteration if it is incomplete.

Choosing the appropriate team structure:

- Choosing the right DevOps team structure depends heavily on organizational size, culture, and specific needs.

- There isn't a one-size-fits-all solution, but understanding different models like centralized, decentralized, or hybrid, and aligning them with factors like product complexity and existing processes is crucial for success.

Choosing the appropriate team structure:

Factors to Consider:

Organizational Size and Complexity:

Smaller organizations might thrive with a centralized structure, while larger enterprises often benefit from decentralized or hybrid approaches.

Product Complexity:

Organizations with fewer product lines may find it easier to collaborate across teams, while those with diverse offerings might need specialized teams.

Choosing the appropriate team structure:

Factors to Consider:

Existing Culture and Processes:
A strong collaborative culture can facilitate the adoption of DevOps principles, while organizations with siloed departments may require more effort to integrate.

Technical Leadership and Alignment:
Shared goals between development and operations, along with strong technical leadership, are crucial for success.

Resource Availability:
The extent to which an organization is willing to invest in DevOps, including training and tools, will influence the chosen structure.

Choosing the appropriate team structure:

- Centralized DevOps Team:
- A single team responsible for both development and operations.
- Effective for smaller teams and simpler product lines.
- It promotes clear communication and a shared vision, but can become a bottleneck if the team is too large or the manager is overwhelmed.
- Decentralized DevOps Team:
- Separate development and operations teams with strong collaboration and communication.
- Good for larger organizations with diverse product offerings, but requires careful coordination to avoid silos.

Choosing the appropriate team structure:

- Embedded DevOps Team:
- Development and operations teams integrate and work closely together on specific projects.
- This can be effective for focused initiatives and can lead to faster delivery times.
- Hybrid DevOps Team:
- A combination of centralized and decentralized approaches.
- This allows for flexibility and adaptation based on specific needs and can be a good long-term solution.

Beyond Structure: Collaboration and Communication

Regardless of the chosen structure, fostering a collaborative and communicative environment is crucial. This involves:

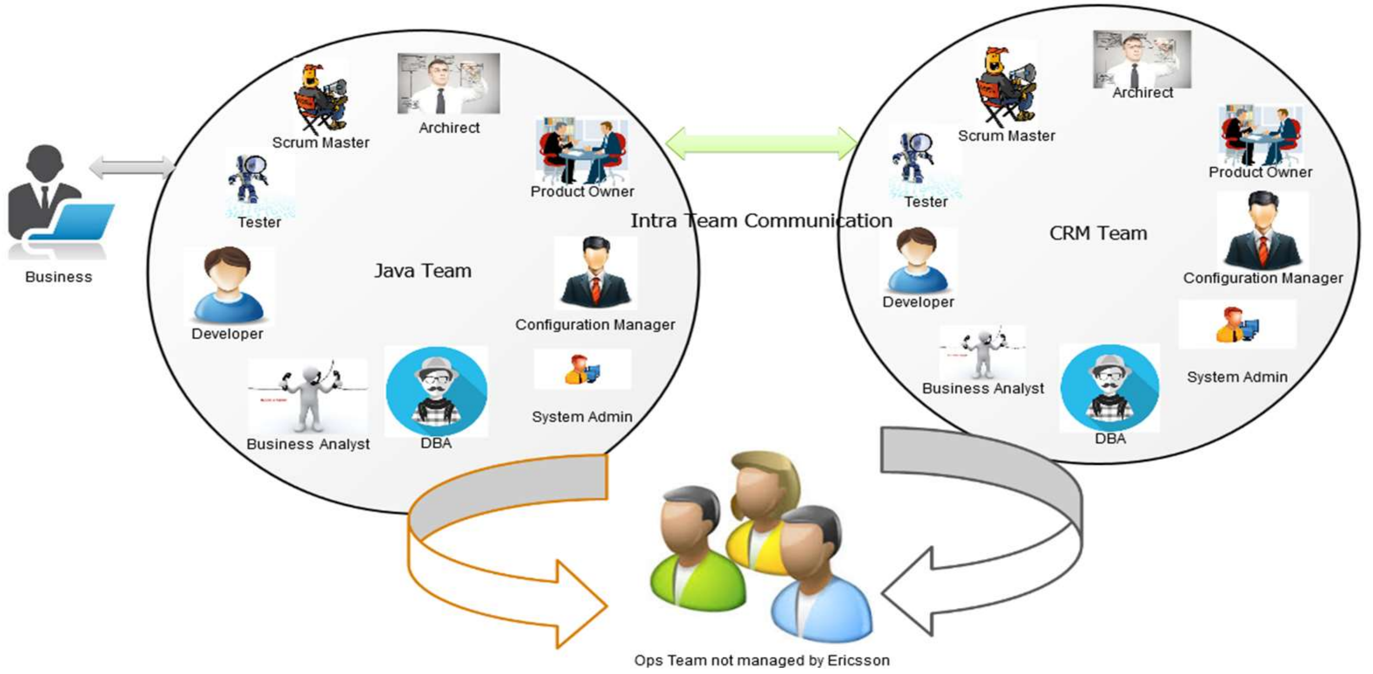
- Open Communication Channels:
- Encourage regular meetings, shared communication tools, and feedback mechanisms to ensure
- everyone is on the same page.

- Shared Goals and Vision:
- Development and operations teams should understand and work towards the same objectives, fostering a sense of shared responsibility.

- Continuous Improvement:
- Regularly assess and refine processes and workflows based on feedback and performance data.

Beyond Structure: Collaboration and Communication

By carefully considering these factors and choosing the appropriate structure, organizations can unlock the full potential of DevOps and achieve faster, more reliable software delivery.



Feature Teams:

- Focus:
- These teams are responsible for delivering complete, end-to-end customer-facing features, from requirements gathering to deployment.

- Expertise:
- They are cross-functional, meaning they possess all the necessary skills (e.g., front-end, back-end, testing, design) within the team to deliver a feature independently.

- Workflow:
- They pull features directly from the product backlog and are responsible for all aspects of their implementation and delivery.

Feature Teams:

- Advantages:

- Faster time-to-market for features, reduced handoffs and dependencies, increased ownership and accountability for feature delivery, and improved collaboration within the team.

- Disadvantages:

- Can require broader skill sets within team members, potential for less deep specialization in specific technical areas, and may require careful management of shared components.

Component Teams:

- Focus:
- These teams specialize in developing and maintaining a specific technical component or subsystem of a larger application (e.g., a database team, a UI framework team, an API team).

- Expertise:
- Members typically possess deep expertise in their specialized component.

- Workflow:
- They deliver reusable components that are then integrated by other teams to build end-user features.

- This can lead to handoffs and dependencies between teams.

Component Teams:

•Advantages:

•Promotes deep technical expertise, reusability of components, and potentially better architectural consistency within a specific component.

•Disadvantages:

•Can lead to slower feature delivery due to handoffs and coordination overhead, potential for silos, and a lack of end-to-end feature ownership.

Enterprise Agile Frameworks:

- complementary approaches that, when combined, enable organizations to achieve agility and efficiency in software development and delivery.
- Enterprise Agile frameworks like SAFe and Disciplined Agile provide the structure for scaling Agile principles across large organizations, while DevOps focuses on the technical practices needed for continuous integration, delivery, and deployment.

Enterprise Agile Frameworks:

•Scaling Agile:

•Frameworks like SAFe (Scaled Agile Framework), Disciplined Agile (DA), and Large-Scale Scrum (LeSS) provide guidance on how to apply Agile principles across multiple teams and departments in large organizations.

•Focus on Alignment:

•These frameworks emphasize aligning teams, projects, and the overall business strategy with Agile principles.

Enterprise Agile Frameworks:

- Coordination and Dependencies:

- They help manage dependencies between teams and coordinate complex releases of software and other outcomes.

- Value Delivery:

- Enterprise Agile frameworks aim to ensure that Agile practices are used to deliver value effectively and consistently.

- Customization:

- Organizations often adapt these frameworks to suit their specific needs and context.

Relevance to Each Other:

- Agile provides the foundation:
- Agile principles like iterative development, collaboration, and customer focus provide the foundation for DevOps.

- DevOps enables Agile at Scale:
- DevOps practices, like CI/CD automation, allow organizations to realize the full potential of Agile by accelerating the delivery of software in large-scale environments.

- Continuous Improvement:
- Both Agile and DevOps foster a culture of continuous improvement, with feedback loops driving ongoing optimization of processes and products.

Relevance to Each Other:

- Reduced Time to Market:

- By combining Agile and DevOps, organizations can significantly reduce the time it takes to deliver software and respond to changing business needs.

- Enhanced Collaboration:

- DevOps enhances collaboration between teams, which is also a key principle of Agile.

Examples of Frameworks Integrating Agile and DevOps:

- SAFe:
- SAFe incorporates DevOps principles and practices into its framework, emphasizing the integration of development and operations teams.

- Disciplined Agile (DA):
- DA recognizes the importance of DevOps and provides guidance on how to integrate DevOps strategies into its framework.

- By leveraging both Enterprise Agile frameworks and DevOps, organizations can create a more agile, efficient, and responsive software development and delivery pipeline, leading to better business outcomes.

Key aspects of TDD:

- Test-driven development reverses traditional development and testing.
- So, instead of writing your code first and then retroactively fitting a test to validate the piece of code you just wrote, test-driven development dictates that you write the test first and then implement code changes until your code passes the test you already wrote.
- In TDD, you write your unit test first, watch it fail, and then implement code changes until the test passes.
- Sounds backwards, right? But the code you produce when you use this testing methodology is cleaner and less prone to breaking in the long run.

Key aspects of TDD:

- A unit test is simply a test that covers a small portion of logic, like an algorithm, for example. Unit tests should be deterministic.

- When I say “deterministic” I mean that unit tests should never have side-effects like calls to external APIs that deliver random or changing data. Instead, you’d use mock data in place of data that could potentially change over time.

Five steps of test-driven development

- There are 5 steps in the TDD flow:
- Read, understand, and process the feature or bug request.
- Translate the requirement by writing a unit test.
- If you have hot reloading set up, the unit test will run and fail as no code is implemented yet.
- Write and implement the code that fulfills the requirement.
- Run all tests and they should pass, if not repeat this step.
- Clean up your code by refactoring.
- Rinse, lather and repeat.

Five steps of test-driven development

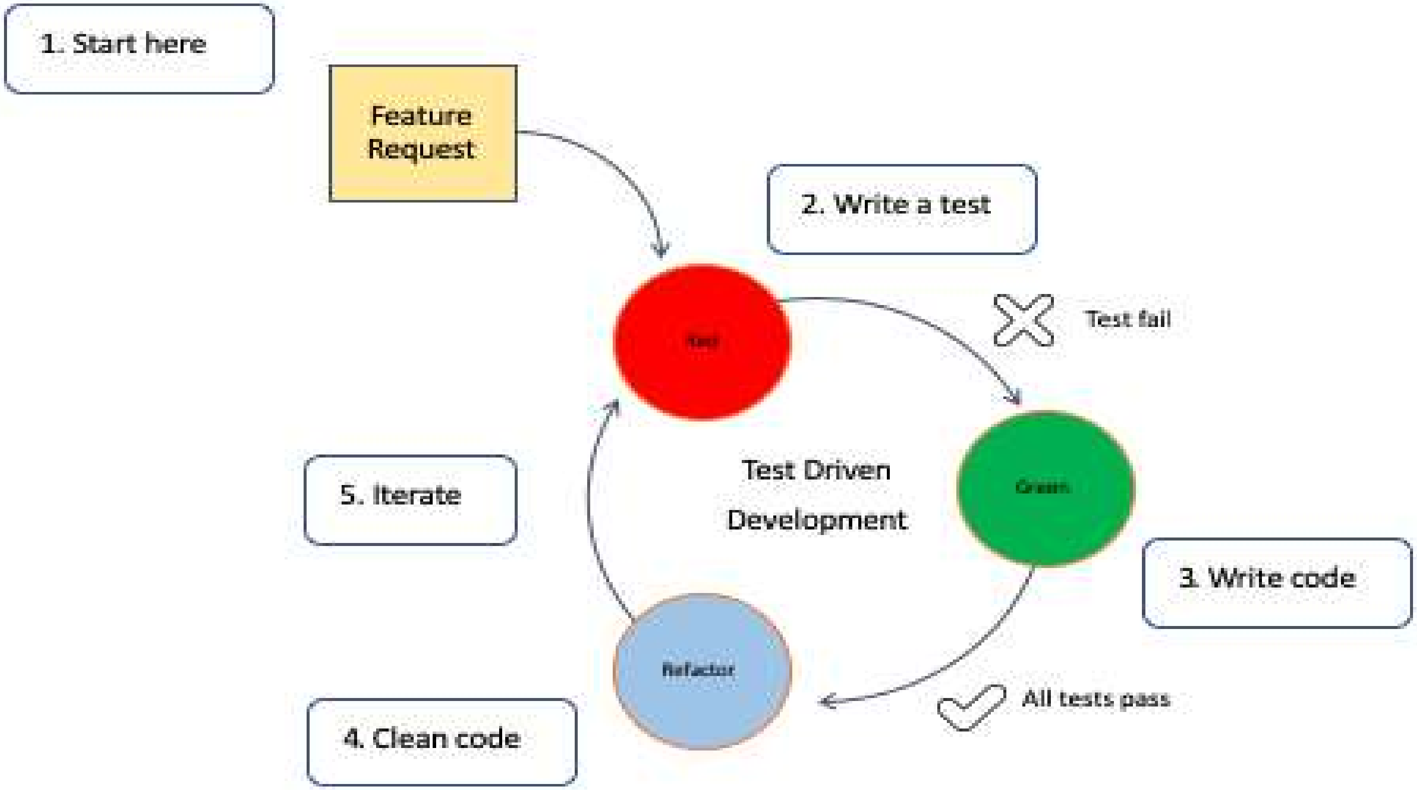


Figure shows these steps and their agile, cyclical, and iterative nature.

Five steps of test-driven development

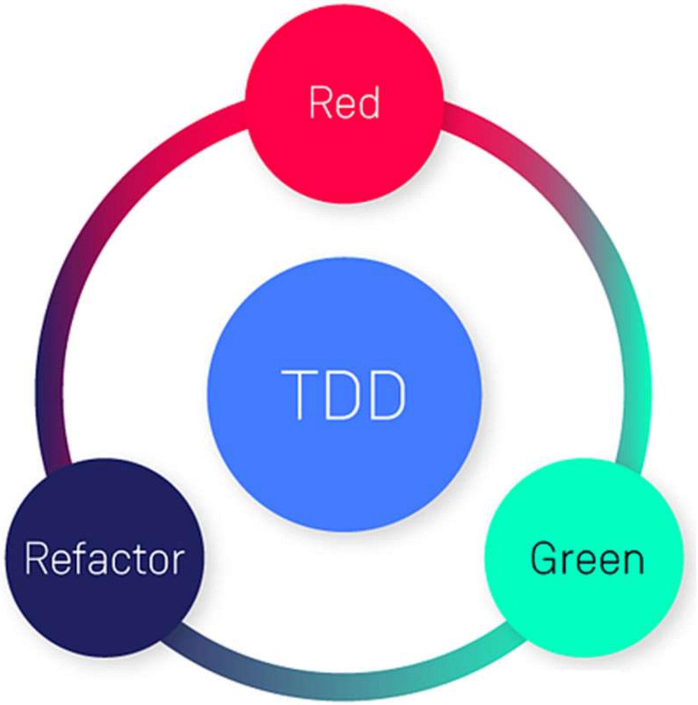
- This workflow is sometimes called Red-Green-Refactoring, which comes from the status of the tests within the cycle.

- The red phase indicates that code does not work.

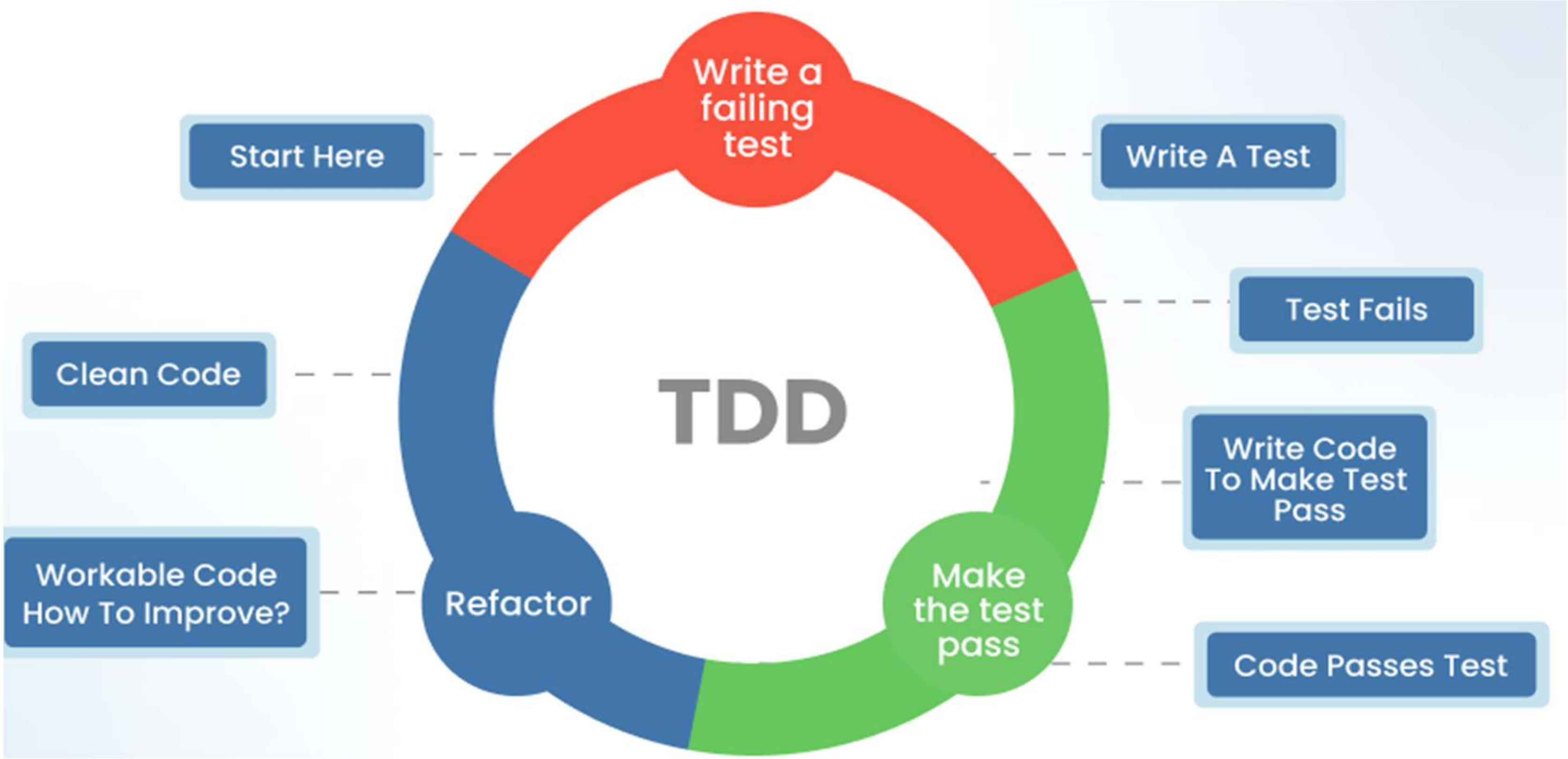
- The green phase indicates that everything is working, but not necessary in the most optimal way.

- The blue phase indicates that the tester is refactoring the code, but is confident their code is covered with tests which gives the tester confidence to change and improve our code.

Five steps of test-driven development



Five steps of test-driven development



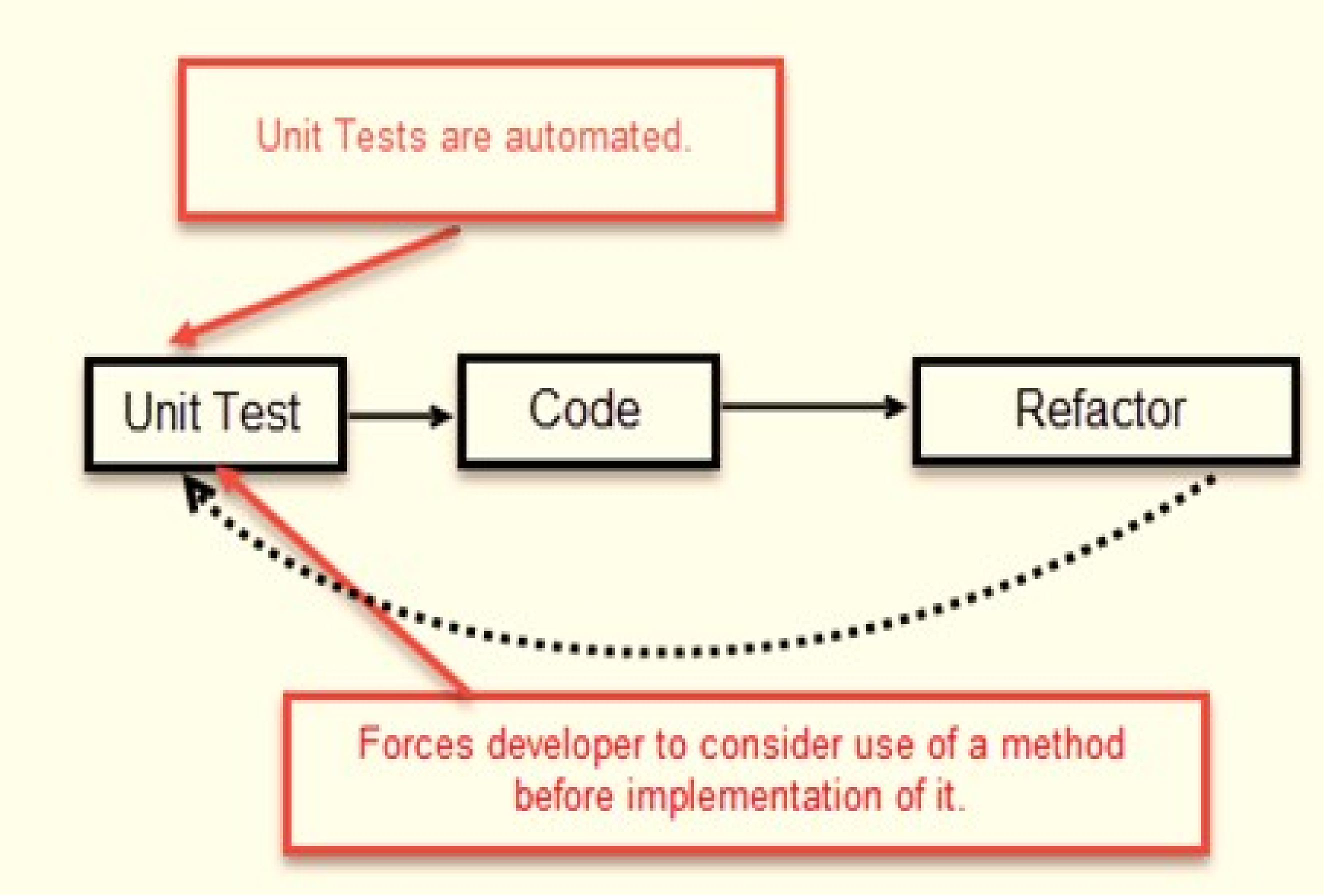
Five steps of test-driven development

- The unit tests that come out of TDD are also an integral part of the continuous integration/continuous delivery (CI/CD) process.
- TDD relates specifically to unit tests and continuous integration/continuous delivery pipelines like CircleCI, GoCD, or Travis CI which run all the unit tests at commit time.
- The tests are run in the deployment pipeline.
- If all tests pass, integration and deployment will happen.
- On the other hand, if any tests fail, the process is halted, thus ensuring the build is not broken.

Test Driven Development (TDD)

- Test Driven Development (TDD) is software development approach in which test cases are developed to specify and validate what the code will do.
- In simple terms, test cases for each functionality are created and tested first and if the test fails then the new code is written in order to pass the test and making code simple and bug-free.
- Test-Driven Development starts with designing and developing tests for every small functionality of an application.
- TDD framework instructs developers to write new code only if an automated test has failed.
- This avoids duplication of code.

Test Driven Development (TDD)



Unit Tests are automated.

Forces developer to consider use of a method before implementation of it.

Test Driven Development (TDD)

- The simple concept of TDD is to write and correct the failed tests before writing new code (before development).
- This helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests. (Tests are nothing but requirement conditions that we need to test to fulfill them).
- Test-Driven development is a process of developing and running automated test before actual development of the application.
- Hence, TDD sometimes also called as Test First Development.

Key aspects of BDD in DevOps:

- Shared Understanding and Communication:

- BDD emphasizes creating a common language and understanding of system behavior through concrete examples and scenarios, typically using the Gherkin "Given-When-Then“ format.

- This bridges the communication gap between business analysts, developers, testers, and operations teams, leading to clearer requirements and reduced misunderstandings.

"Given-When-Then" (GWT) is a structured way to express acceptance criteria for software features, particularly in the context of Behavior-Driven Development (BDD)

Benefits of using Given-When-Then:

- Clarity and Communication:

GWT makes acceptance criteria easier to understand for both technical and non-technical stakeholders.

- Testability:

The structured format makes it straightforward to create test cases that verify the specified behavior.

- Collaboration:

It encourages collaboration between developers, testers, and business analysts during the requirements gathering and testing process.

- Reduced Ambiguity:

GWT helps minimize ambiguity in requirements by clearly defining the expected behavior in a specific context.

Benefits of using Given-When-Then:

- Example:
- Let's say you're testing a feature that allows users to deposit money into their bank account:
- **Given**: the user has a positive balance in their account.
- **When**: the user deposits $100 into their account.
- **Then**: the account balance should increase by $100.
- By using GWT, you can clearly communicate the expected behavior of the deposit feature and create tests to ensure it functions correctly.

Key aspects of BDD in DevOps:

- Executable Specifications and Automation:

- BDD scenarios serve as executable specifications that can be directly translated into automated tests.

- This enables continuous testing throughout the DevOps pipeline, providing rapid feedback on whether the software meets the expected behaviors.

Key aspects of BDD in DevOps:

- Early Feedback and Continuous Improvement:

- By integrating BDD into the DevOps pipeline, teams can identify and address issues early in the development cycle, reducing the cost and effort of fixing defects later.

- This continuous feedback loop supports iterative development and continuous improvement.

- Alignment with Business Value:

- BDD scenarios are written from the perspective of the end-user or business, ensuring that development efforts are aligned with delivering features that provide real business value.

.

Key aspects of BDD in DevOps:

- Living Documentation:

- The BDD scenarios and automated tests act as living documentation, always reflecting the current state of the system's behavior.

- This provides a reliable source of truth for all teams and reduces the effort required for maintaining separate documentation.

How BDD enhances DevOps:

- Improved Collaboration:

- Fosters cross-functional collaboration by bringing together business, development, testing, and operations teams to define and validate system behaviors.

- Faster Feedback Loops:

- Automates acceptance tests, providing quick and continuous feedback on the impact of changes.

- Reduced Rework:

- Clarifies requirements upfront, minimizing misunderstandings and subsequent rework.

How BDD enhances DevOps:

- Higher Quality Software:

- Ensures that the developed software consistently meets the desired behaviors and acceptance criteria.

- Enhanced Traceability:

- Links user stories and requirements directly to automated tests and code, improving traceability throughout the development lifecycle.

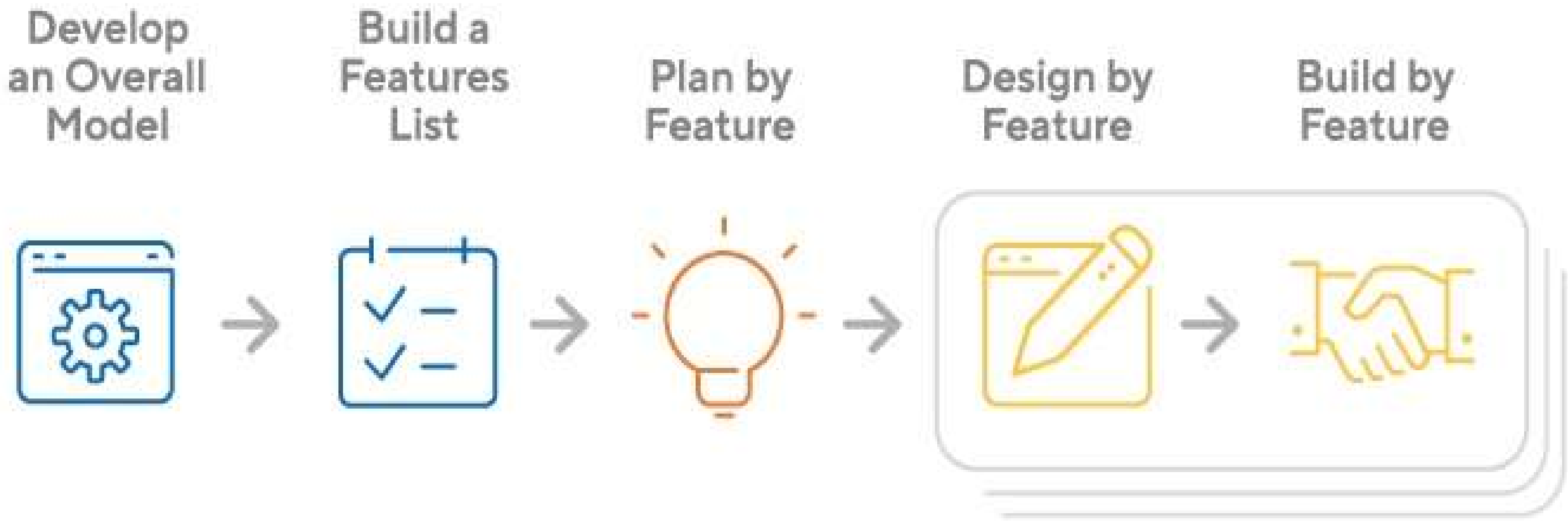
How FDD integrates with DevOps:

- FDD was designed to follow a five-step development process, built largely around discrete “feature” projects. That project lifecycle looks like this:

- Develop an overall model
- Build a features list
- Plan by feature
- Design by feature
- Build by feature

- The framework has since gained widespread use particularly in larger organizations, and today there is a thriving Feature Driven Development community.

How FDD integrates with DevOps:



How FDD integrates with DevOps:

- Iterative and Incremental Development:
- FDD's focus on short, iterative cycles for developing "features" (small, client-valued functions) naturally complements DevOps' emphasis on continuous integration and continuous delivery (CI/CD).

Each completed feature can be integrated, tested, and potentially deployed rapidly.

- Collaboration and Communication:
- FDD encourages close collaboration within feature teams and with stakeholders, promoting a shared understanding of requirements and progress.

- This fosters the cross-functional collaboration central to DevOps, breaking down silos between development and operations.

How FDD integrates with DevOps:

- Continuous Integration and Delivery:
- FDD's emphasis on delivering tangible, working software frequently aligns with CI/CD practices.

- Features are developed, integrated, and tested in small increments, allowing for early detection of issues and continuous deployment to various environments.

- Automated Testing and Quality Assurance:
- The iterative nature of FDD encourages continuous testing throughout the development lifecycle, which is crucial for maintaining quality in a fast-paced DevOps environment.

- Automated testing can be implemented to validate each feature as it's developed and integrated.

How FDD integrates with DevOps:

- Monitoring and Feedback:

- FDD's focus on progress tracking and reporting at all levels provides valuable data for monitoring the health and performance of features in production, feeding back into the development process for continuous improvement, a key aspect of DevOps.

- Feature Flags and Progressive Delivery:

- FDD can leverage feature flags within a DevOps setup to control the rollout of new features, allowing for A/B testing, phased releases, and quick rollback if issues arise, minimizing risk and enhancing user experience.

How FDD integrates with DevOps:

- Feature Flags and Progressive Delivery:

- By combining the structured, feature-centric approach of FDD with the automation, collaboration, and continuous delivery practices of DevOps, organizations can achieve faster delivery cycles, improved software quality, and enhanced responsiveness to client needs.

How FDD integrates with DevOps:

- FDD’s strengths include:
- Simple five-step process allows for more rapid development
- Allows larger teams to move products forward with continuous success
- Leverages pre-defined development standards, so teams are able to move quickly

How FDD integrates with DevOps:

- FDD’s weaknesses include:
- Does not work efficiently for smaller projects
- Less written documentation, which can lead to confusion
- Highly dependent on lead developers or programmers

Cloud as a catalyst for DevOps

Essential Characteristics

On-demand Self-service

Resource pooling

Broad network access

Rapid Elasticity

Measured Service

Scalability:

- Ability to system to handle increasing workload by increasing in proportion the amount of resource capacity.
- Architecture allows on-demand resources for increasing workload

Elasticity:

- Dynamically Commissioning and decommissioning of resources
- Speed at which resources are provisioned on demand and usage of resources

Cloud as a catalyst for DevOps

Private Cloud

Private Cloud: The infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on or off the premises of the organization.

Public Cloud

Public Cloud: The infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

Hybrid Cloud

Hybrid Cloud: The infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability.

DevOps– Cloud as a catalyst

- Cloud is key enabler and has affected DevOps. It helps DevOps to mature.

- In traditional organizations that use physical servers, requesting a new server is a complex, time-consuming process.

- It may take days or even weeks to requisition a new physical server, install and set up the operating system (OS), and configure all the software required to deploy the application.

- Pervasiveness of cloud technology, private, public, and hybrid, has enabled organizations to provision environments on demand.

DevOps– Cloud as a catalyst

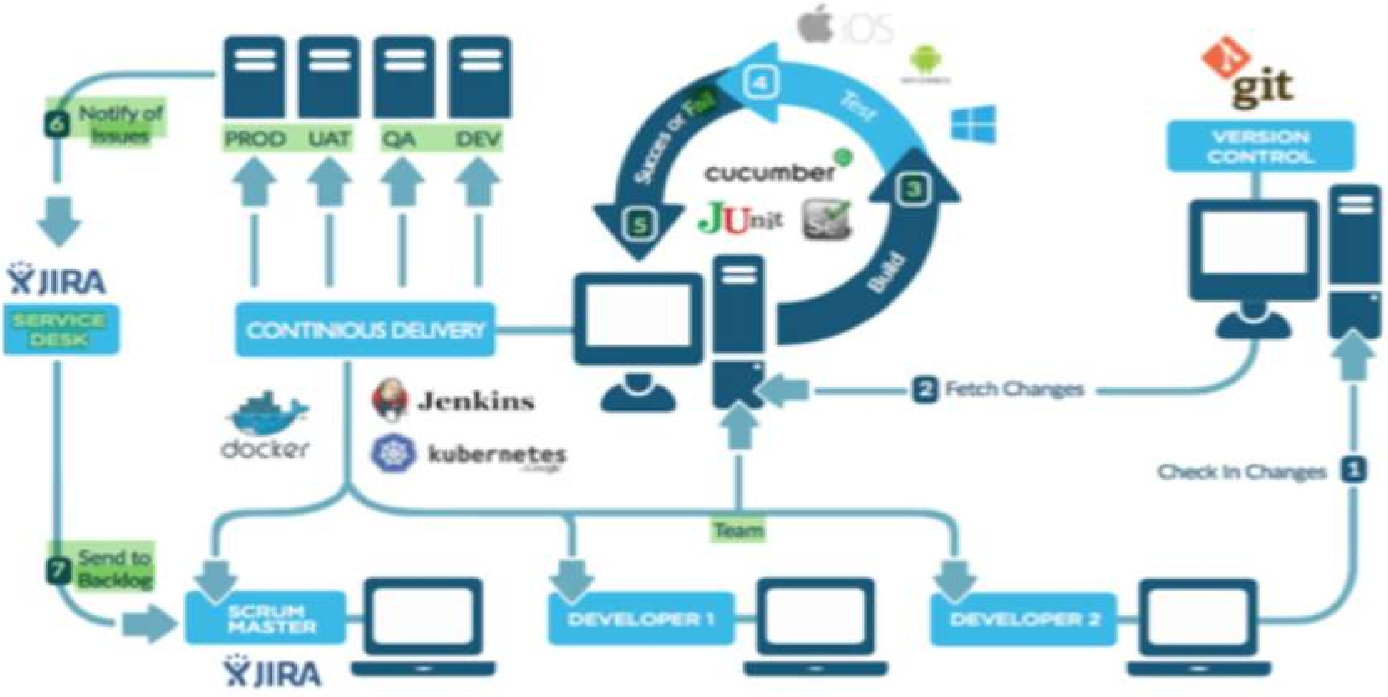
- Virtualization in Cloud solve that problem to a large extent.
- This has led to tremendous growth in the scale by which the environments to which applications are deployed are used.
- This scale has necessitated automation on software deployment tasks, which DevOps addresses
- Ability to create and switch environments simply, the ability to create VMs easily, and the management of databases etc..

DevOps– Cloud as a catalyst

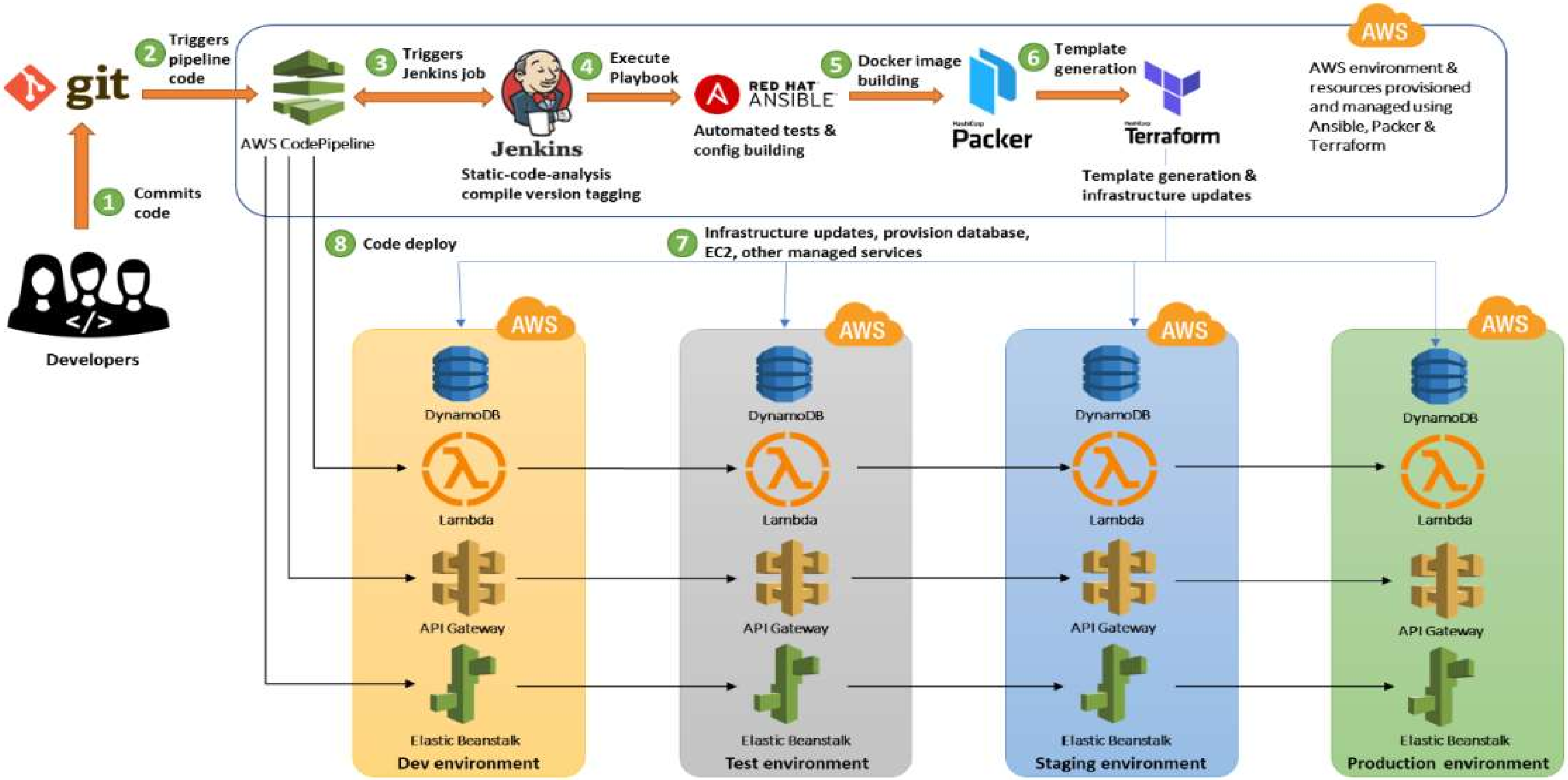
- Advent of cloud actually facilitates continuous deployment and provides easier access to production like environments during the development and testing stages of the delivery pipeline.

- Developer can not only invoke an automated build, but also request the provisioning and configuration of a production like environment in the cloud and then deploy the application being built to it without any direct involvement by the operations team.

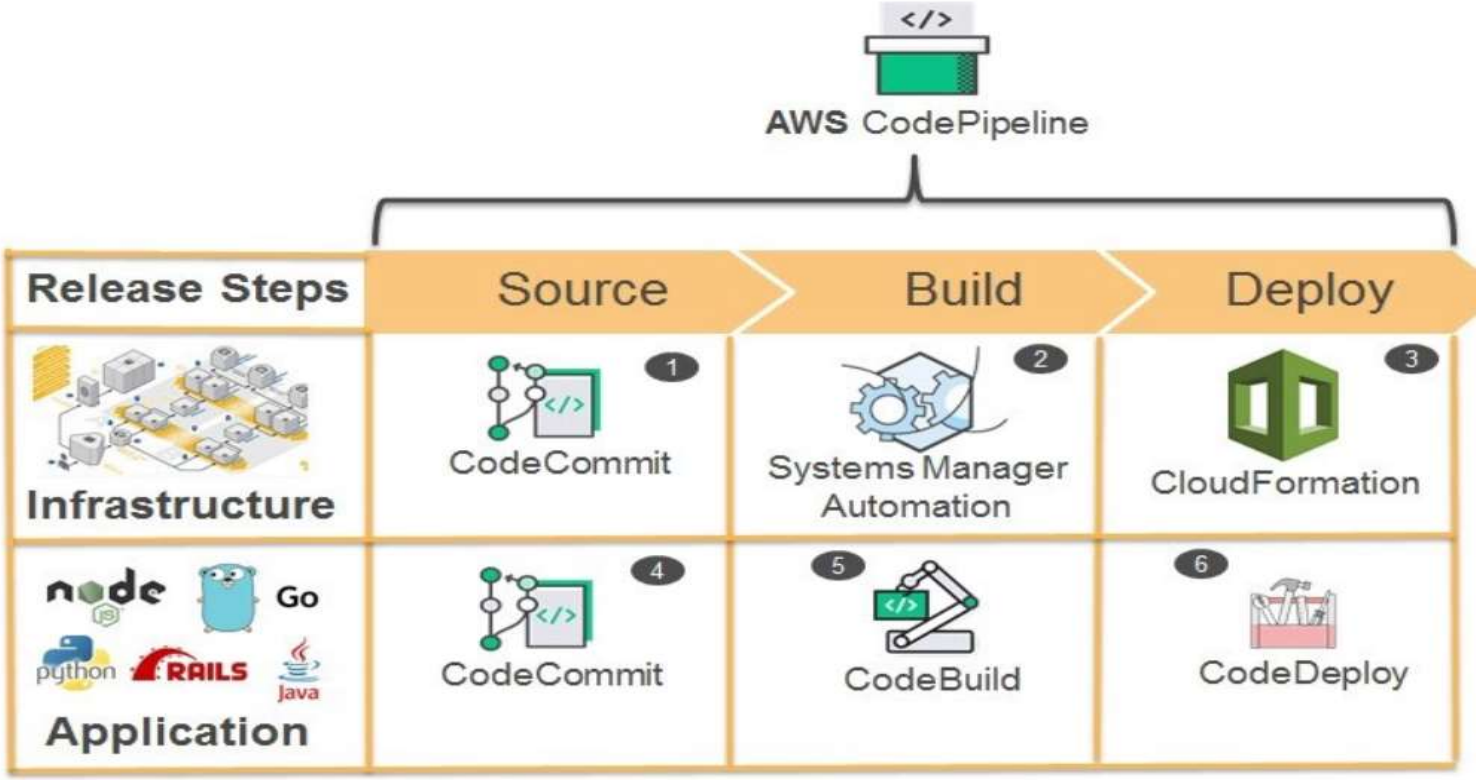
DevOps– Cloud as a catalyst



DevOps -AWS


git
2 Triggers pipeline code
AWS CodePipeline
3 Triggers Jenkins job
Jenkins
Static-code-analysis compile version tagging
4 Execute Playbook
RED HAT ANSIBLE
Automated tests & config building
5 Docker image building
Packer
6 Template generation
Terraform
Template generation & infrastructure updates
AWS
AWS environment & resources provisioned and managed using Ansible, Packer & Terraform
1 Commits code
Developers
8 Code deploy
7 Infrastructure updates, provision database, EC2, other managed services
AWS
DynamoDB
Lambda
API Gateway
Elastic Beanstalk
Dev environment
Test environment
Staging environment
Production environment

Amazon AWS and DevOps



AWS CodePipeline

Release Steps	Source	Build	Deploy
Infrastructure	1 CodeCommit	2 Systems Manager Automation	3 CloudFormation
node Go python RAILS Java Application	4 CodeCommit	5 CodeBuild	6 CodeDeploy

QUESTIONS AND DISCUSSION

THANK YOU