Continuous Delivery

Webinar DevOps Institute & DevOn

About me

Ashwin Shankarananda

*"I help organizations become a **Responsive Enterprise**"*

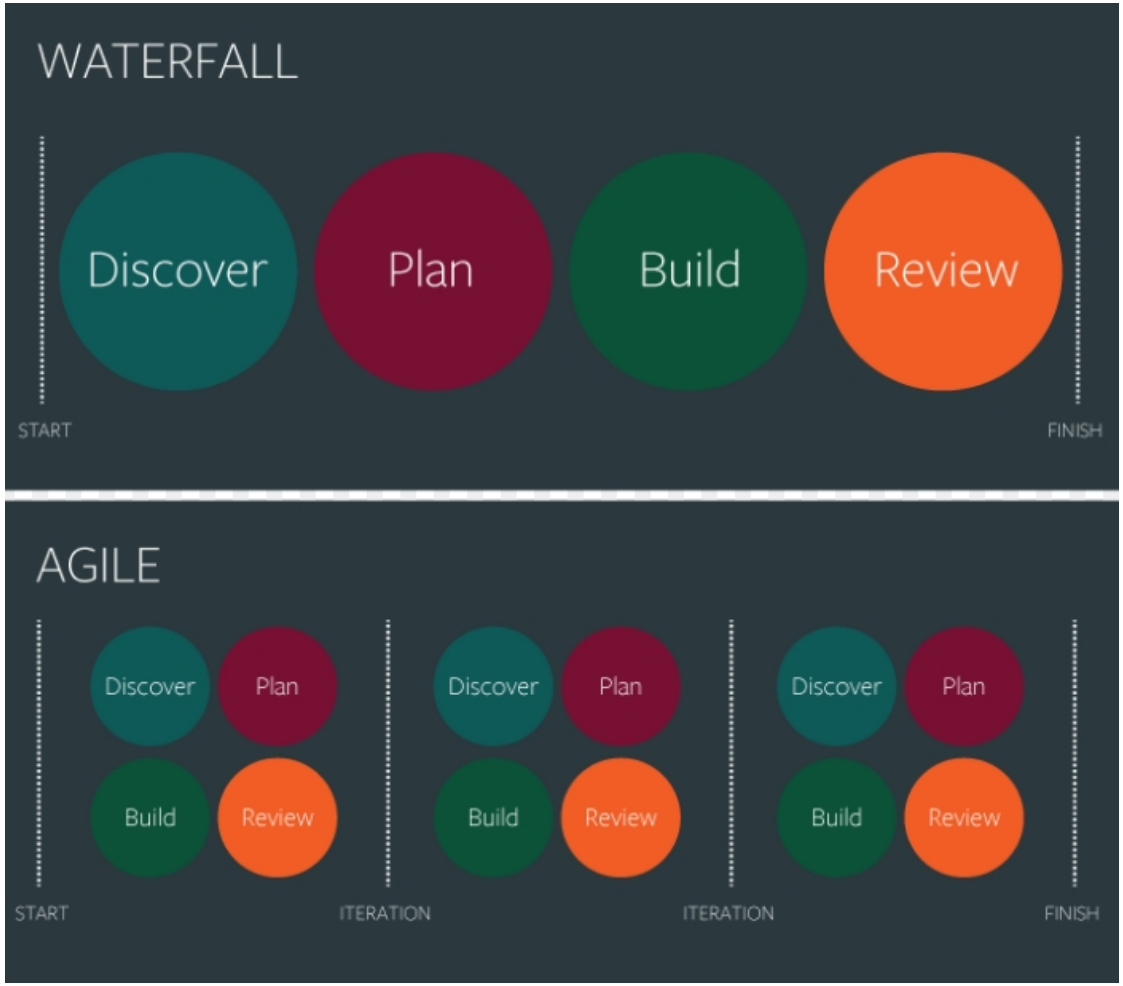
- Principal consultant
- DevOn
- 11 years of IT experience
- Certified trainer
- Business Intelligence and Business Analytics
- ITIL V3

Agenda

1. Need for Continuous Delivery
2. What is Continuous Delivery
3. Design & Architect Continuous Delivery Systems
4. Continuous Delivery Pipeline
5. Demo

Waterfall vs. Agile

Waterfall vs. Agile



Benefits of Agile over Waterfall

Advantages of the Agile model

- It is focused on client process, so it makes sure that the client is continuously involved during every stage.
- Agile teams are extremely motivated and self-organized so it's likely they provide a better result from the development projects.
- Agile software development methods assure that quality of development is maintained.
- The process is completel based on the incremental progress. Therefore, the client and team know exactly what is complete and what is not. This reduces risk in the development process.

Limitations of the Waterfall model

- It is not an ideal model for a large-size project.
- If the requirement is not clear at the beginning, it is a less effective method.
- Very difficult to move back to make changes in the previous phases.
- The testing process starts once development is over. Hence it has high chances of bugs to be found later in development where they are expensive to fix.

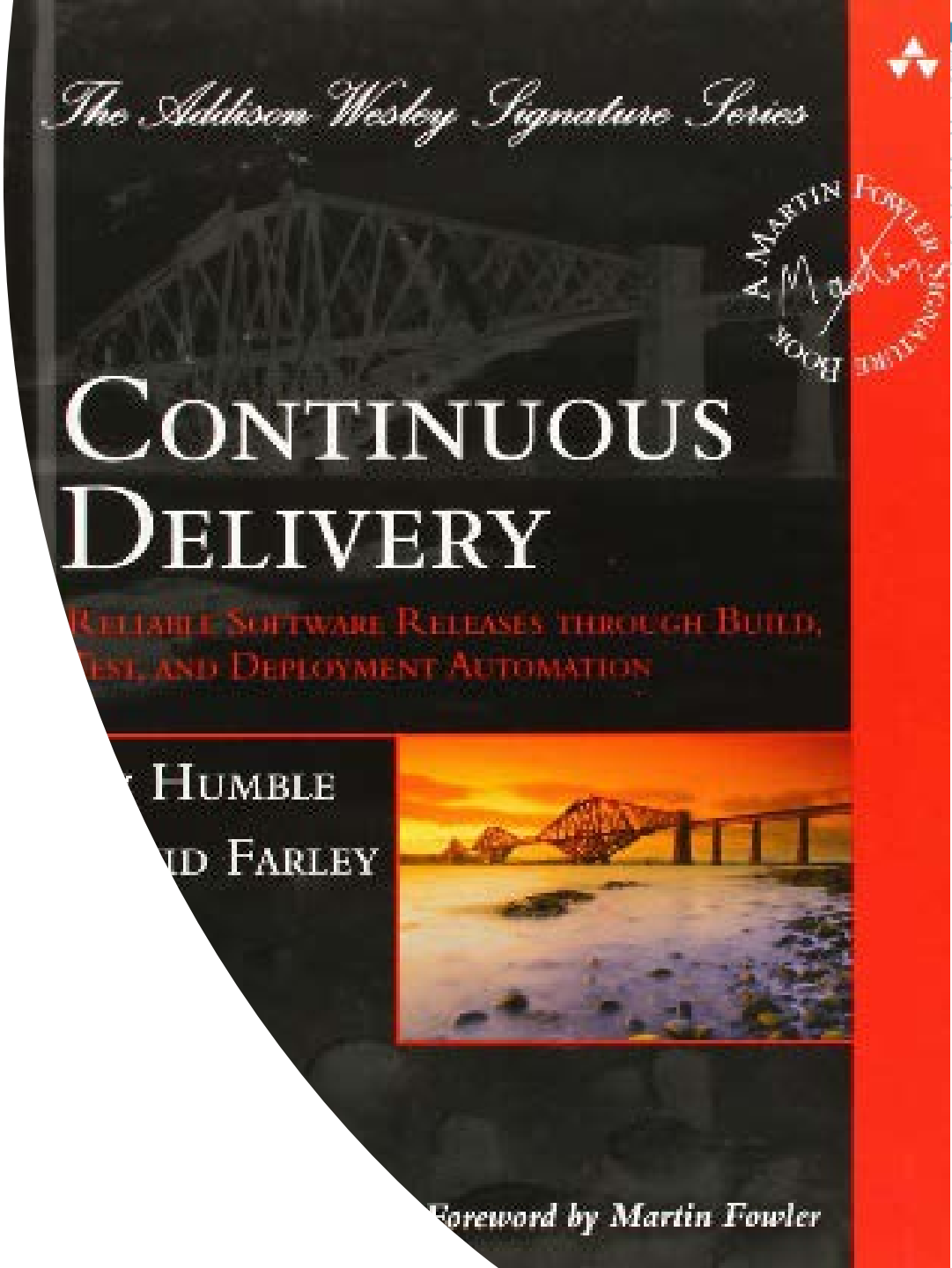
Agile to Continuous Delivery

Continuous Delivery (CD) is the application development discipline that takes Agile to its logical conclusion, creating software that is always ready to release. It does this by building upon and extending Agile, CI and DevOps practices and tools to transform the way software is delivered.

What is Continuous Delivery?

"Continuous Delivery is the ability to get changes of all types – including new features, configuration changes, bug fixes and experiments – into production, or into the hands of users, *safely* and *quickly* in a *sustainable way.*"

- Jez Humble



Continuous Delivery Principles

- Build quality in
- Work in smaller batches
- Automation mindset
- Strive for Continuous Improvement
- Delivery is everyone's responsibility

Pillars of Continuous Delivery

Configuration management

ANSIBLE CHEF puppet labs SALTSTACK

Continuous Integration

Continuous Testing

Katalon Studio
Unified Functional Testing
TestComplete
TestPlant eggPlant
TRICENTIS
watir
Rational software
Ranorex
ROBOT FRAMEWORK

Configuration Management

CM is a practice of handling changes systematically to maintain system integrity over time. It implements the policies, procedures, techniques and tools to:

- Manage & Evaluate changes
- Track Status
- Maintain inventory and document

Why is it important?

- Reproducibility
- Increase uptime
- Improve performance
- Ensure Compliance
- Prevent Errors
- Reduce Costs
- Traceability

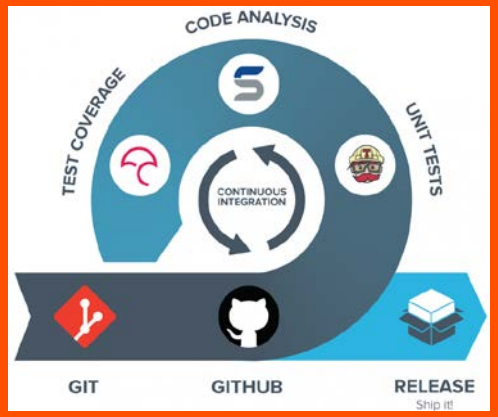
ANSIBLE CHEF puppet labs SALTSTACK

Continuous Integration

Continuous Integration is a development practice that requires software changes to be integrated into a shared repository as many times in a day as possible, followed by automated builds, automated testing and quality checks to catch integration errors as fast as possible it work on fail fast fix fast approach.

What are the benefits of CI?

- Risk Mitigation
- Confidence
- Team communication
- Reduced overhead
- Consistency of Build Process

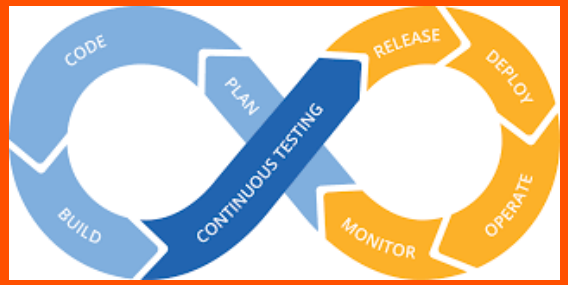


Continuous Testing

Continuous Testing is the process of executing automated tests as part of the software delivery pipeline to obtain feedback on the business risks associated with a software release candidate as rapidly as possible.

What are the benefits of CI?

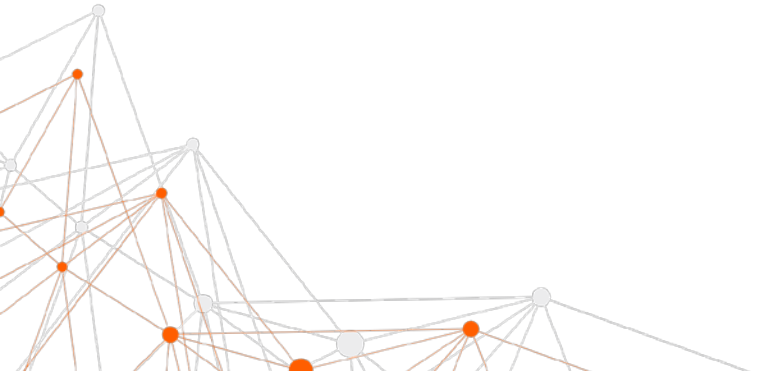
- Reduce application-bound risks.
- Brings consistency.
- Enables Faster Release.
- Improves Test Coverage.
- Enables more transparency.



Katalon Studio
Unified Functional Testing
TestComplete
TestPlant eggPlant
TRICENTIS
watir
Rational software
Ranorex
ROBOT FRAMEWORK

Architecting for CD

- Availability
- Security
- Performance
- Usability
- Testability
- Deployability
- Scalability



Architecting for Deployability

Deployability is a non-functional requirement that addresses how reliably and easily software can be deployed from development into production.

- Minimize differences between environments, by effectively using CM practices.
- Parameterize and look up differences by environment.
- Deployable systems can typically be upgraded or reconfigured with zero or minimal downtime.
- Designing for testability and deployability starts with ensuring our products and services are composed of loosely-coupled, well-encapsulated components or modules.
- To aid the independent deployment of components, we should also invest in creating versioned APIs which have backwards compatibility
- Use binary versioning tools to manage artifacts to avoid rebuilds for promotions.
- Deployment process in pre production should be identical to production, containerization will help bring this consistency.

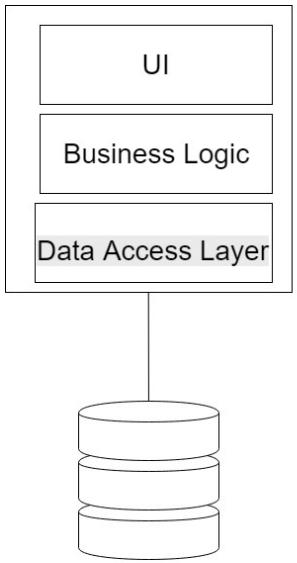
Architecting for Testability

When we talk about Design for Testability, we are talking about the architectural and design decisions in order to enable us to easily and effectively test our system.

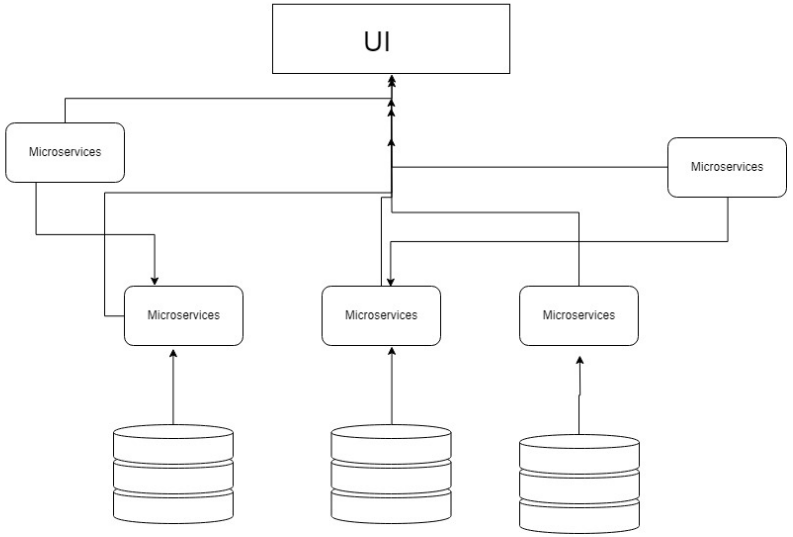
- Software libraries, frameworks, repositories and services should also support testability.
- Having too much logic at DB side makes testing virtually impossible.
- Flexibility to have stubs, test doubles for Unit and Component Testing.
- Efficient logging for diagnoses and maintenance.
- Flexible and simple configuration to set up a test environment.

Microservices

Microservices - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The microservice architecture enables the continuous delivery/deployment of large, complex applications.



Monolithic Architecture



Microservices Architecture

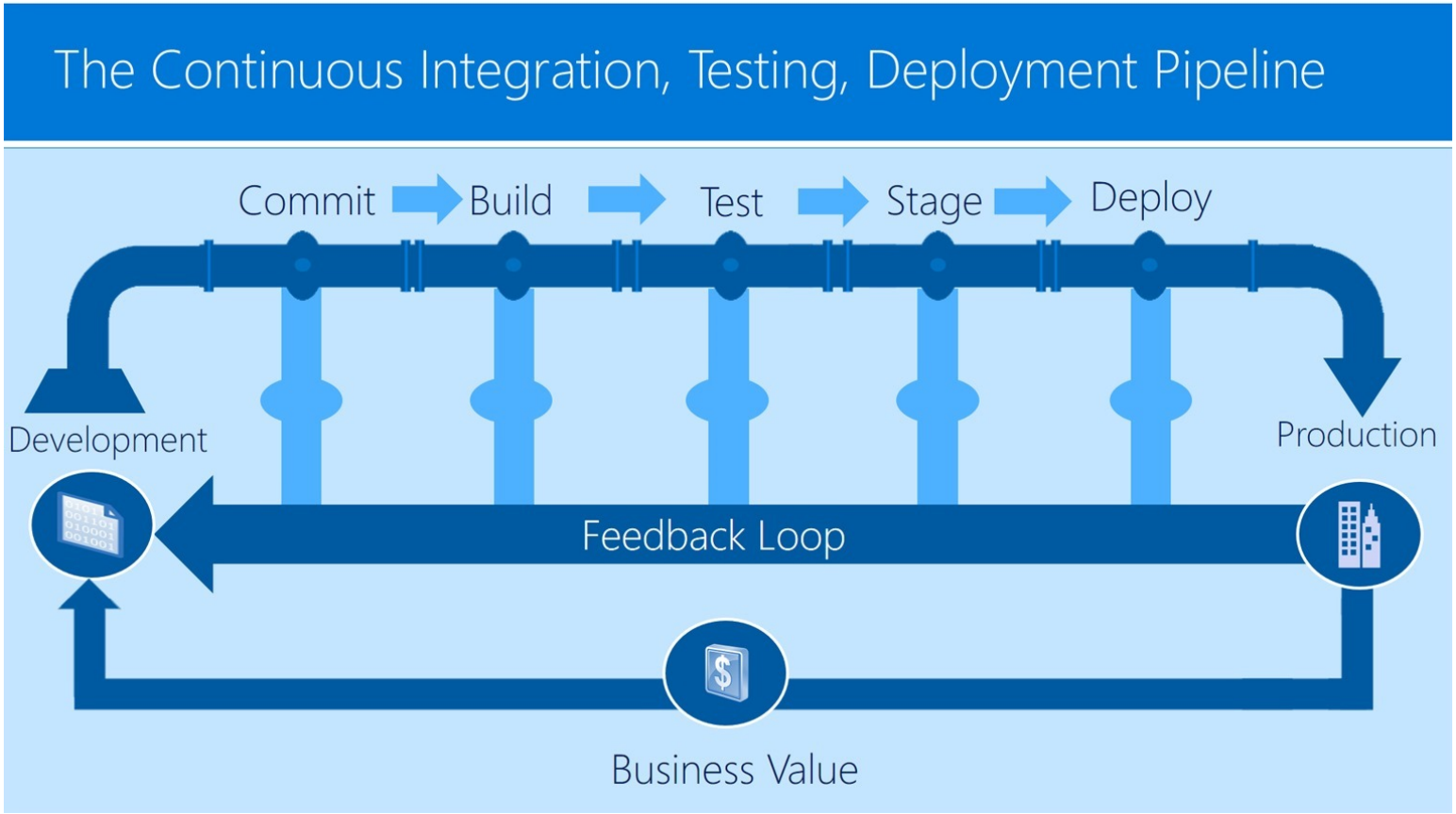
Microservices

Advantages	Disadvantages
Freedom to use different technologies	Increases troubleshooting challenges
Each Microservice focuses on a single business capability	Increases delay due to remote calls
Supports individual deployable units	Increases efforts for configuration and other operations
Allows frequent software releases	Difficult to maintain transaction safety
Ensures security of each service	Tough to track data across various service boundaries
Multiple services are parallelly developed and deployed	Difficult to move code between services

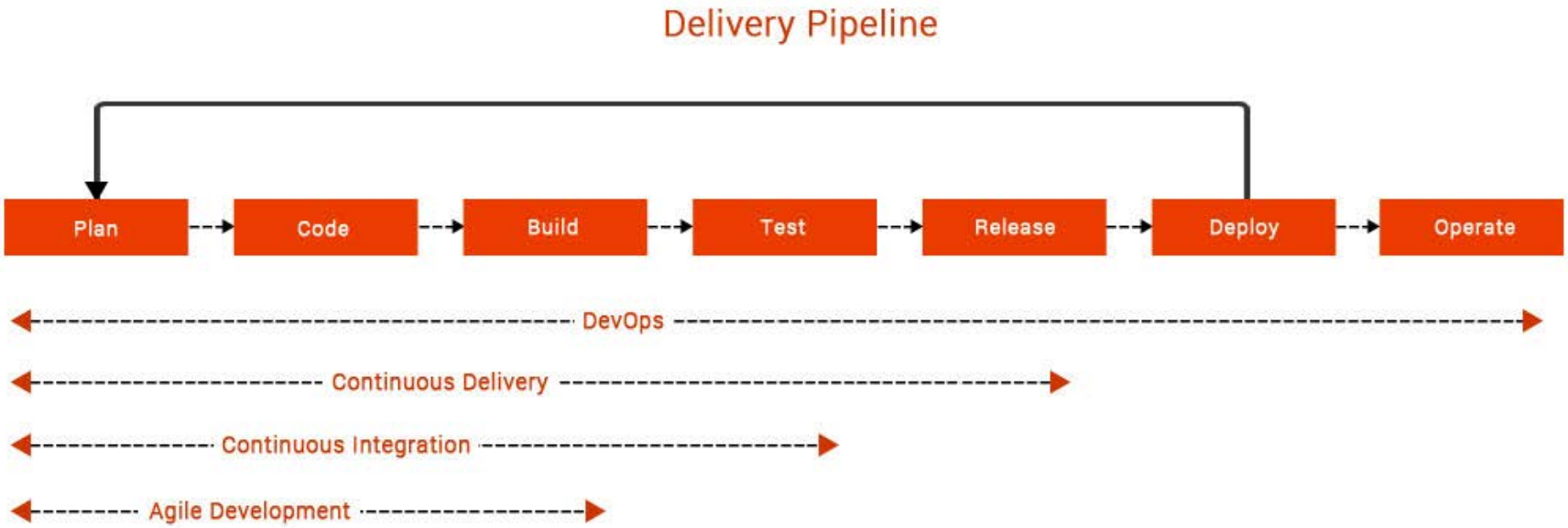
Pipeline

Continuous Delivery (CD) is a software strategy that enables organizations to deliver new features to users as fast and efficiently as possible. The core idea of CD is to create a repeatable, reliable and incrementally improving process for taking software from concept to customer. The goal of Continuous Delivery is to enable a constant flow of changes into production via an automated software production line. The Continuous Delivery pipeline is what makes it all happen.

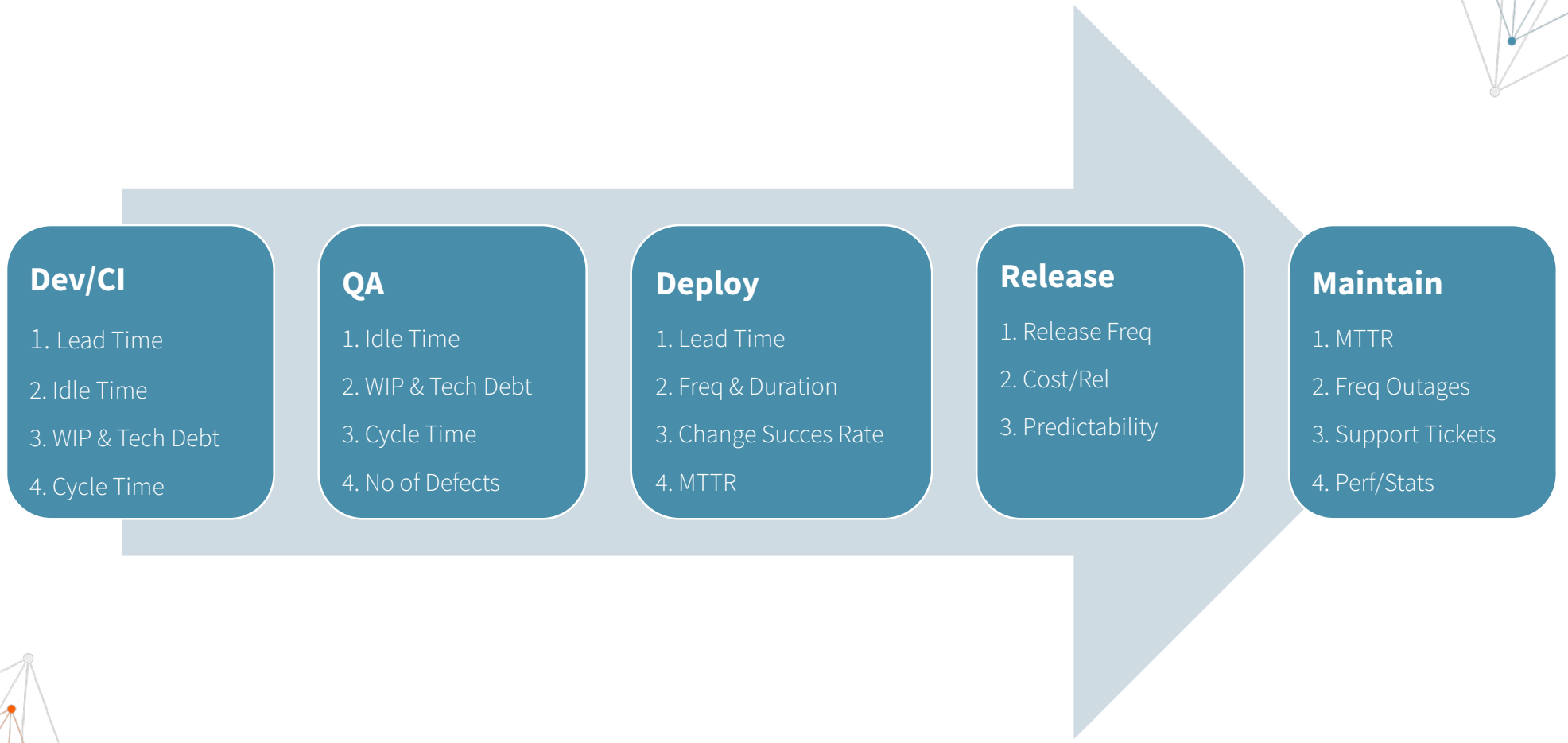
Pipeline


The Continuous Integration, Testing, Deployment Pipeline
Commit
Build
Test
Stage
Deploy
Development
Production
Feedback Loop
Business Value

Pipeline


Delivery Pipeline
Plan
Code
Build
Test
Release
Deploy
Operate
DevOps
Continuous Delivery
Continuous Integration
Agile Development

Metrics to measure



Demo

Thank you!

Next webinar: DevOps Leadership - 2nd October 4pm CEST

Link in your mail!