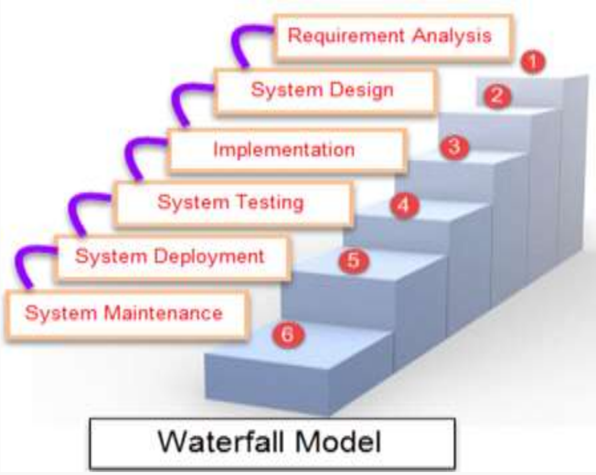
DevOps Overview

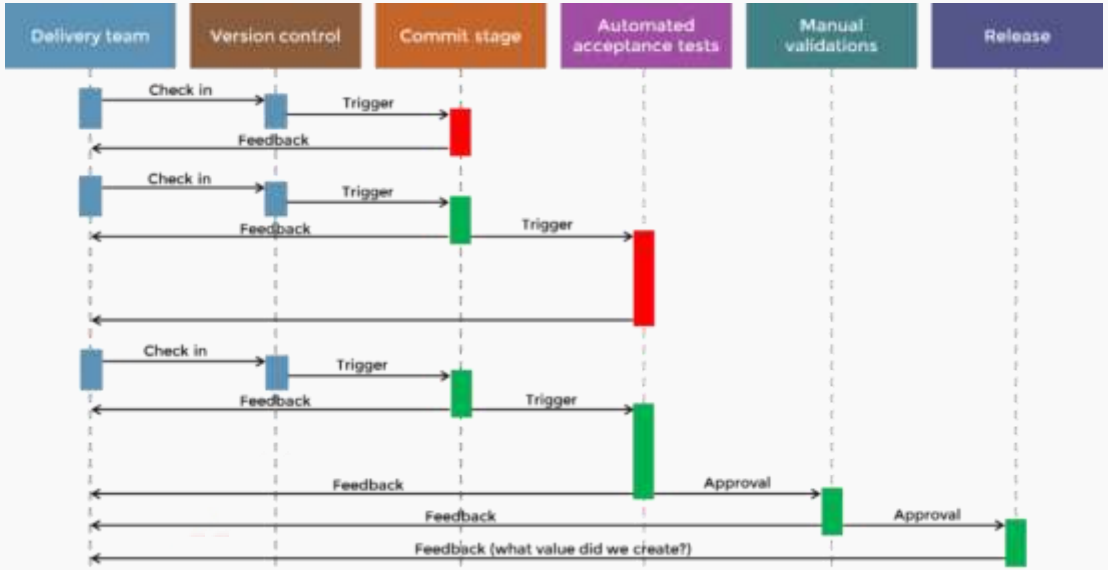
Steve Robinson 2018

Roots of DevOps

Lean and Kaizen



Waterfall Model



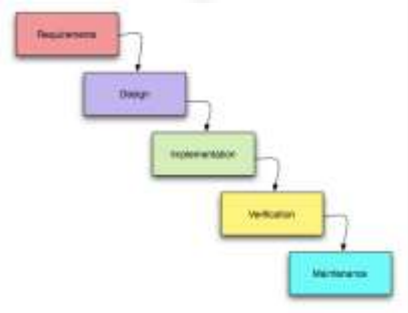
Roots of DevOps (Waterfall-Agile-DevOps)

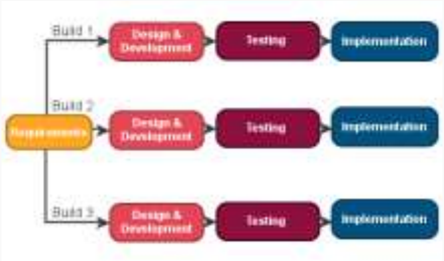
Waterfall

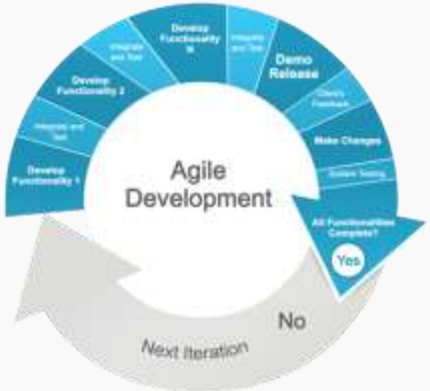
Iterative Agile

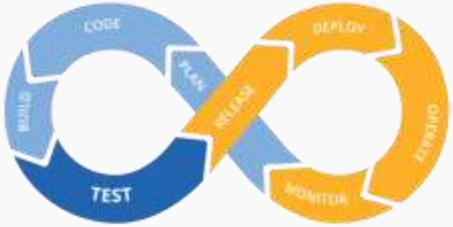
Disciplined Agile

DevOps

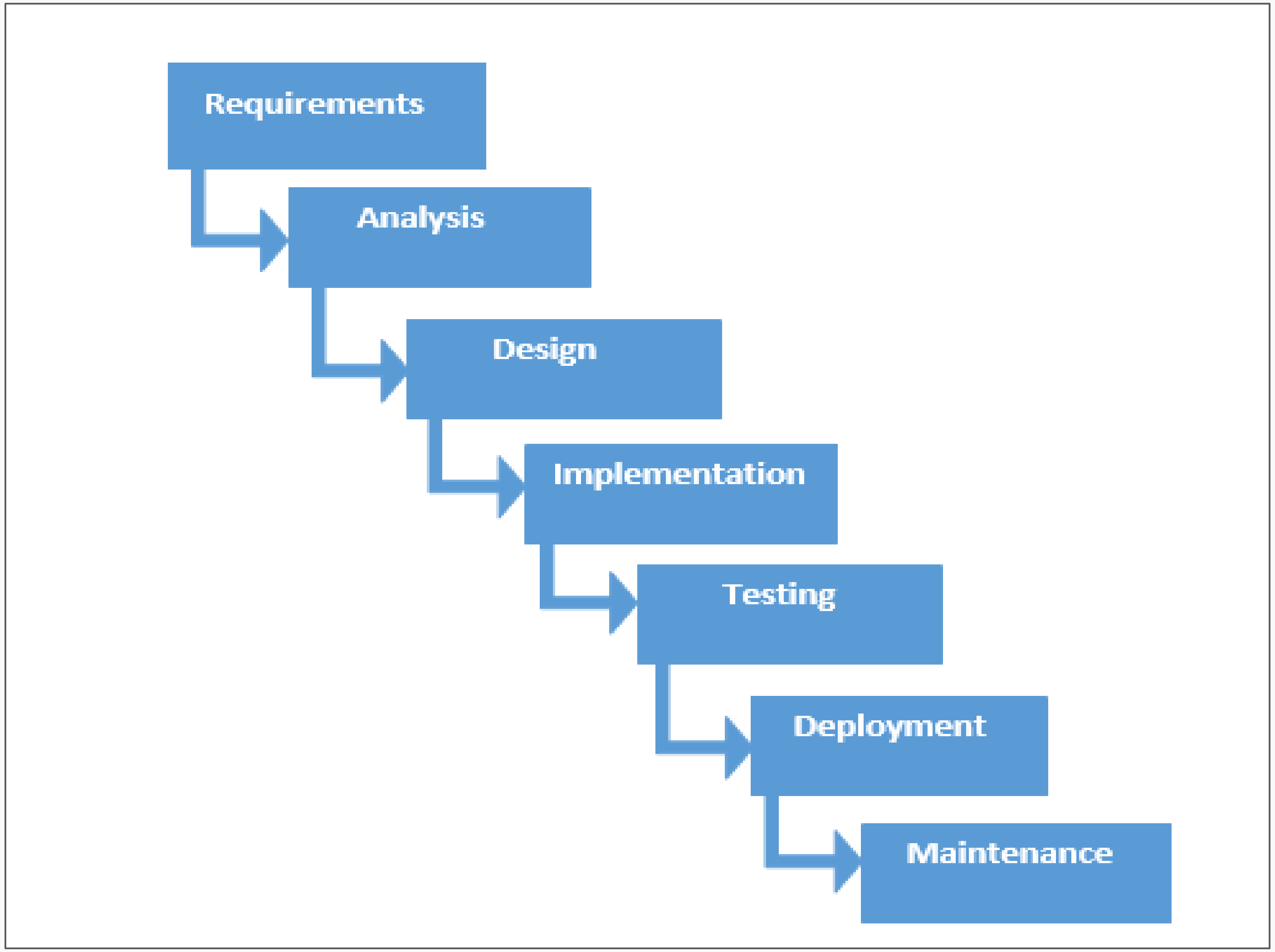








Waterfall Model

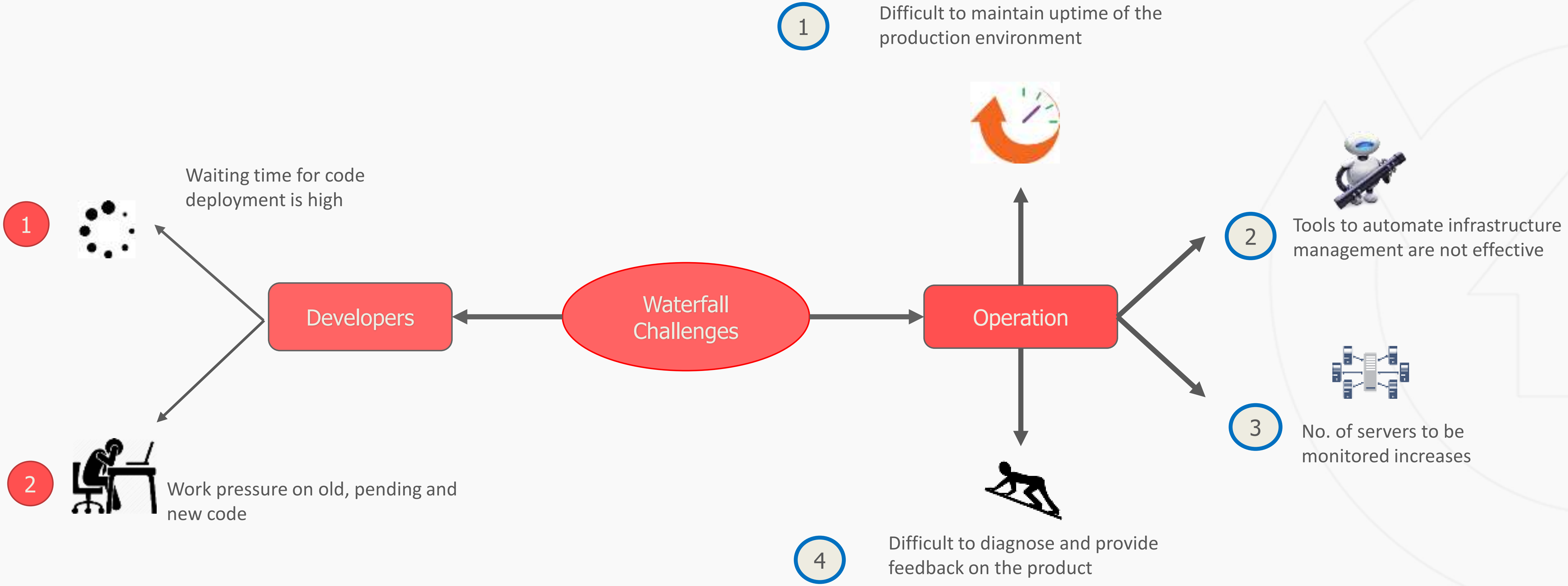


Challenges in Waterfall Model

testhouse

- From Developers side:
  - After development, the code deployment time was huge
  - Pressure of job on "old items" and "pending items" was high, because of the huge development time and deployment time
- From Operations side:
  - It was difficult to maintain 100% uptime of the production environment
  - Automation of the infrastructure set up was not effective
  - Number of servers to be monitored kept increasing and thus resulted in complexity
  - Difficulty in providing feedback and diagnose issue in the product

Challenges in waterfall model



Waterfall Challenges

Developers

1 Waiting time for code deployment is high

2 Work pressure on old, pending and new code

Operation

1 Difficult to maintain uptime of the production environment

2 Tools to automate infrastructure management are not effective

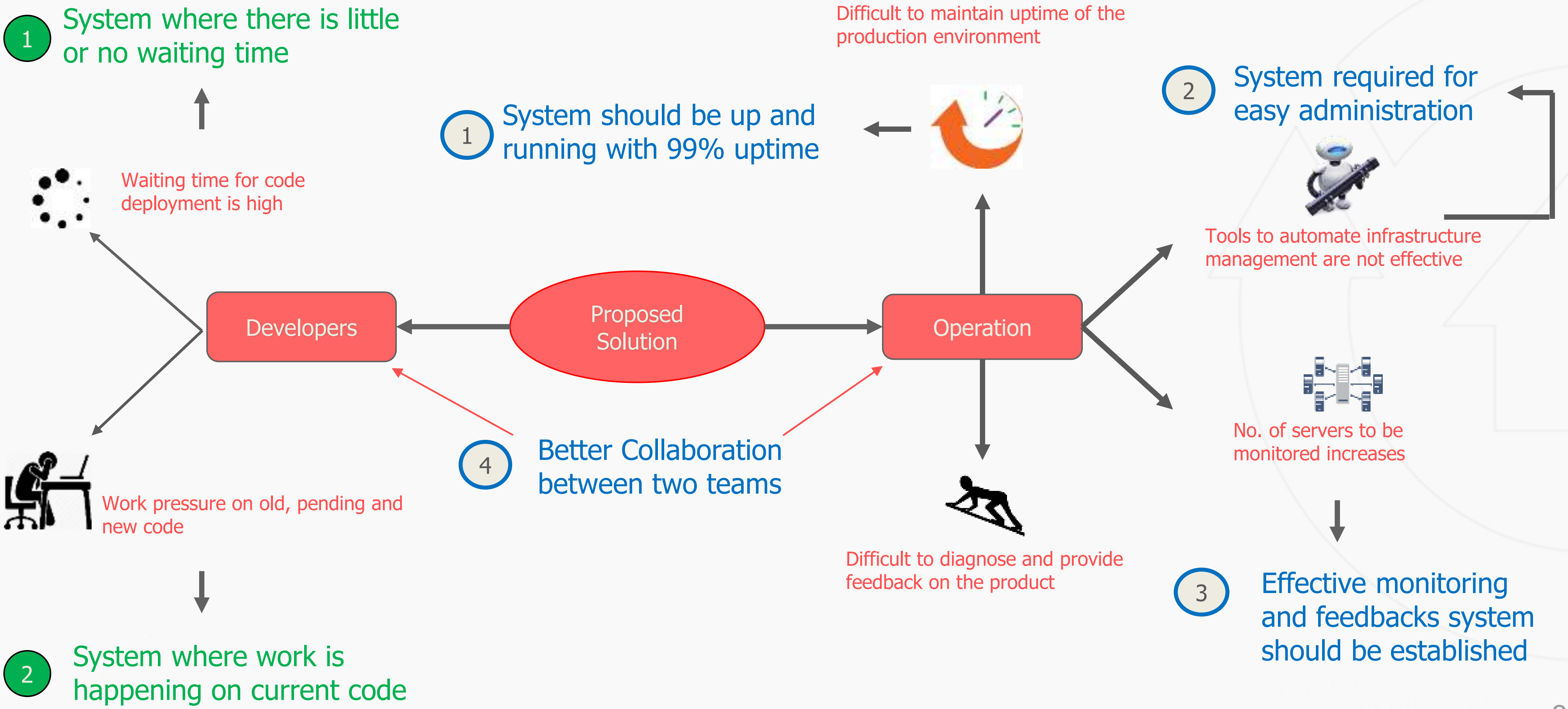
3 No. of servers to be monitored increases

4 Difficult to diagnose and provide feedback on the product

Solutions

- From Developers side:
  - Enablement of an efficient system for code deployment and thus achieve less delay and waiting time
  - A system where work happens on the current code itself i.e. development sprints are short and well planned
- From Operations side:
  - System should have at-least 99% uptime
  - Tools and systems for effective administration
  - Effective monitoring and feedback system
  - Improved collaboration between development and operations team

Solutions



1 System where there is little or no waiting time

Waiting time for code deployment is high

Work pressure on old, pending and new code

2 System where work is happening on current code

Developers

Proposed Solution

Operation

1 System should be up and running with 99% uptime

Difficult to maintain uptime of the production environment

2 System required for easy administration

Tools to automate infrastructure management are not effective

No. of servers to be monitored increases

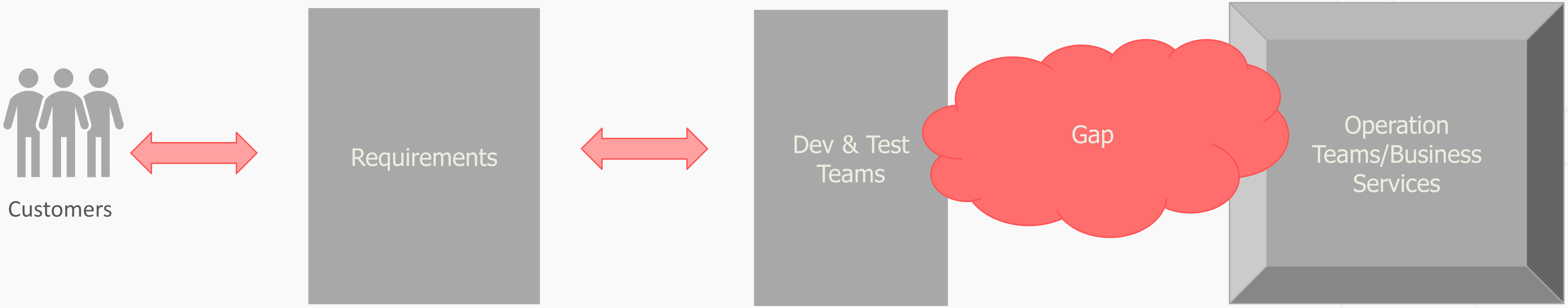
3 Effective monitoring and feedbacks system should be established

Difficult to diagnose and provide feedback on the product

4 Better Collaboration between two teams

Why to go for DevOps?

- To overcome the delivery challenges and gaps



Reasons for Gap

- In Dev environment:
  - Features are delivered after testing in development systems.
  - Dev systems need not be equivalent to production systems.
  - Turn around time of developers will be faster with respect to features.
  - Not much focus on infrastructural and deployment impact because of code changes.

- In Ops environment:
  - Turn around time will be reduced with respect to feature deployment and testing due to large number of builds
  - Concerns about infrastructural and deployment impact because of code changes.

Agile Development

- Agile Development is an umbrella term for several iterative and incremental software development methodologies
- The most popular agile methodologies include Scrum, Kanban, Scaled Agile Framework , Lean Development and Extreme Programming (XP)
- Whilst each of the agile methodologies is unique in its specific approach, they all share a common vision and core values. They all fundamentally incorporate iteration and the continuous feedback that it provides to successively refine and deliver a software system
- They all involve continuous planning, continuous testing, continuous integration, and other forms of continuous evolution of both the project and the software
- In the beginning, agile teams were primarily made up of developers. As these agile teams became more effective and efficient at producing software, it became clear that having Quality Assurance (QA) and Dev as separate teams was inefficient. Agile grew to encompass QA in order to increase the velocity of delivering software and now agile is once again growing to encompass the delivery and support members to extend agility from ideation to delivery.

"Imagine a world where product owners, Development, QA, IT Operations, and Infosec **work together**, not only to help each other, but also to ensure that the overall organisation succeeds.

By working towards a **common goal**, they enable the fast flow of **planned work** into production (e.g. performing tens, hundreds, or even thousands of code deploys per day), while achieving world-class stability, reliability, availability, and security."

The DevOps Handbook

There's a common refrain from IT managers who are leading successful DevOps transformations within their organizations:

DevOps is about continual learning and improvement rather than an end state.

Your specific process will ultimately take its own shape in your organisation as you learn from successes and failures along the way.

"Taking great people and putting them into a pathological or **bureaucratic culture** does not change the culture - It **breaks** the **people**.

The **solution** is to do all the hard work to **transform** the **culture** of the organisation and **grow** effective **leadership** and management appropriate to each horizon – which will, incidentally, remove the need to continually hire or acquire innovators."

LEAN Enterprise

Drivers for Change

- Any Initiative to centralise the development of an Organisation or Business Unit requires the establishment of a defined Software Engineering function supported by Frameworks & Blueprints which define I.T. capability.

- The objective should be to drive significant re-use of code with continuous higher code quality all the while continuously reducing time to delivery/market.

- In addition, a common **Tooling Framework** is required in order to support all teams within the Application Development Service/Lifecycle, fostering collaboration and common goals.

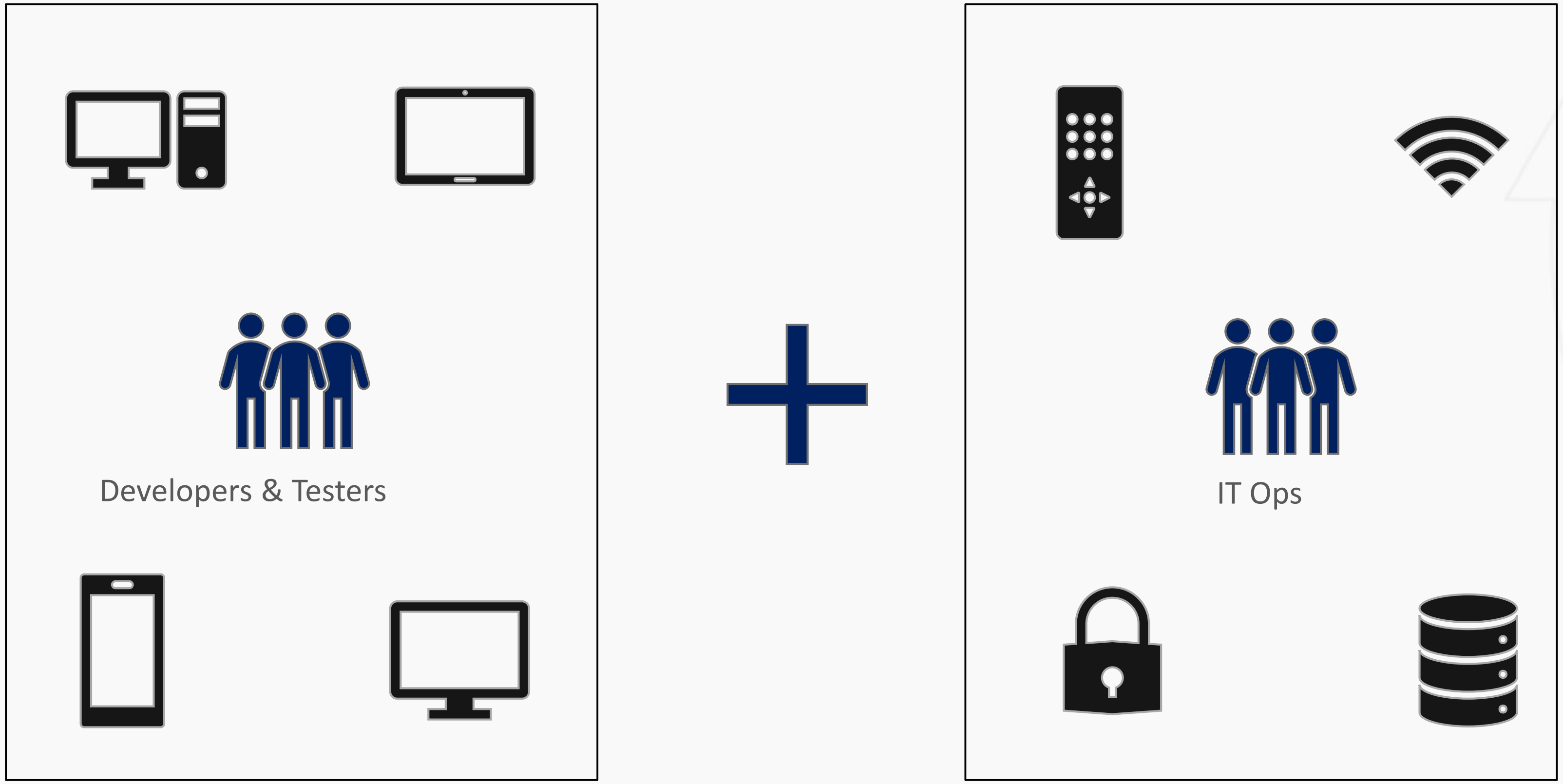
- The tooling frameworks and software engineering working practises need to align to **global DevOps objectives** and **Agile at scale**.

Required for Change

- Centralised Portfolio Management.
- Full Business Case, Requirements tracking.
- Frameworks to outline both the process, the architecture and the tools required.
- Continual Process mapping and Process improvement.
- Governance to manage any process defined.
- Define a story, evangelise the process.
- Self audit, and independent audit.
  - **Constant** and **Never-Ending Continuous Improvement**.

What is DevOps?

A culture & mindset for collaborating between developers and operations



What is DevOps?

- DevOps is a combination of development and operations functions
- The DevOps ideals extend agile development practices by further streamlining the movement of software change through the build, validate, and deploy and delivery stages, while empowering cross-functional teams with full ownership of software applications – from design through production support
- DevOps is not a technology, a process or a standard; rather, it is an IT culture or movement that emphasizes ways in which development, testing and operations can collaborate more effectively. DevOps is more about trust, people and teamwork than about process, and creating software as an ongoing service, not a static product
- Advantages are:
  - Quick evolution of products & services
  - Less risk
  - Better quality
  - Less coast
- DevOps teams focus on standardizing development environments and automating delivery processes to improve delivery predictability, efficiency, security and maintainability. The DevOps ideals provide developers more control of the production environment and a better understanding of the production infrastructure

4 pillars of the DevOps ecosystem

DevOps adoption requires changes to 4 primary aspects of the Enterprise:

● Culture ● Process ● People ● Technology.

- Changes to each pillar tends to influence the changes required in other pillars.
- To realise benefits, enterprise **Processes** must adapt from manual work by administrators and developers in favour of repeatable, automated processes.
  - The effect is the altering of skillsets required of team members.
- The **People** in Development, QA and Operations teams must possess core competencies in automated infrastructure creation/management, automated software build/test and automated application deployment.
- Building these core competencies requires continuous adoption of new **Technologies** within the enterprise to support the work of our teams
- Without efficient communication paths between teams, the **Culture** of the organisation can present an obstacle to progress.

DevOps means continuously innovate your SLDC

testhouse

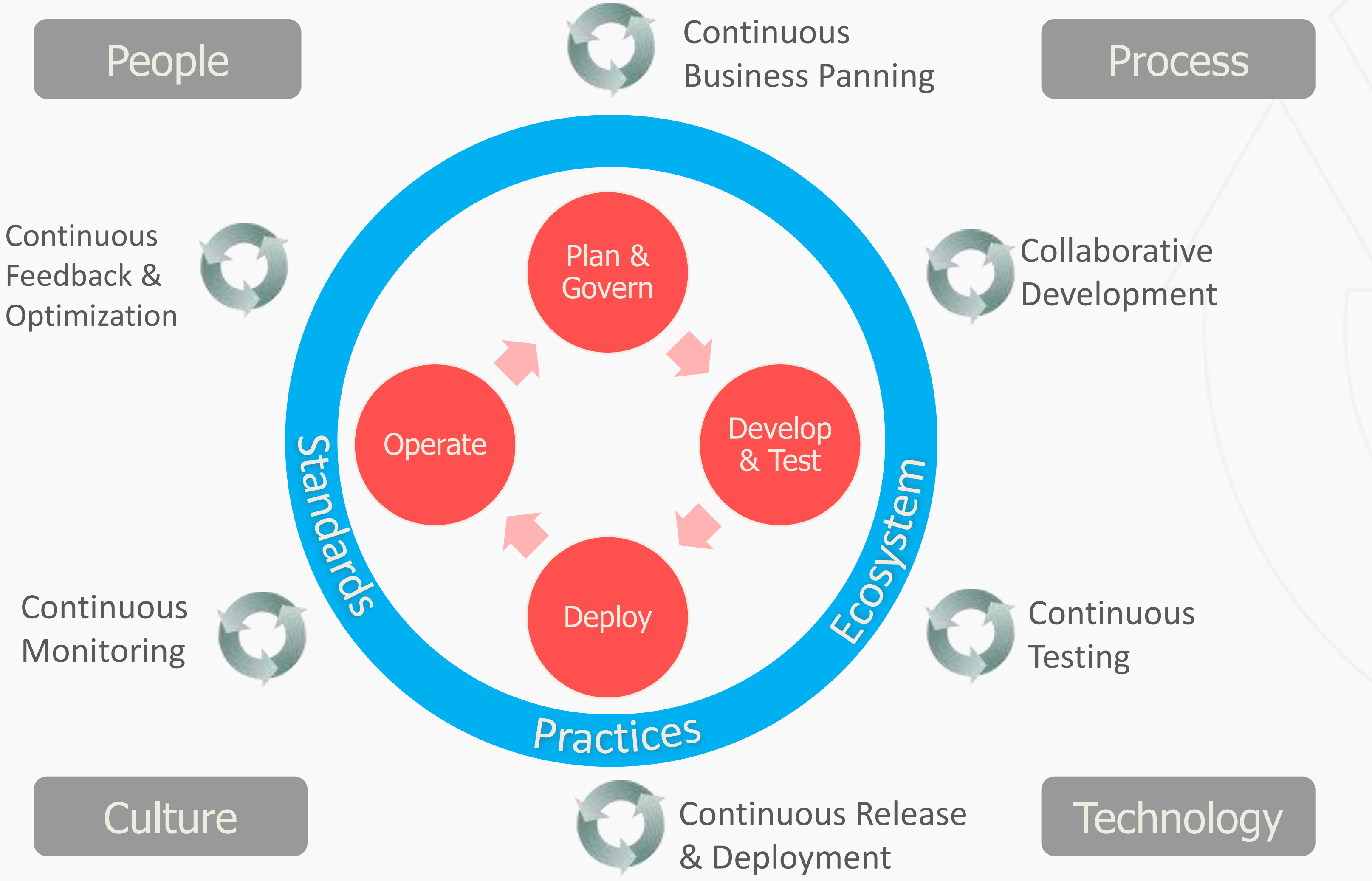
DevOps:

- Addresses all aspects of the Software Delivery Lifecycle (SDLC)
- Does no equal Cloud/Tools, but frequently leverages cloud & technology enablers.
- Has become a strategic imperative largely due to the fundamentally different SDLC process requirements needed to support Systems of Engagement and Delivery in an ever improving cycle.

Speed: More frequent releases and functionality refreshing with decreased time to market.

Quality: Brand and business objectives being represented and evaluated constantly.

Cost: On-going focus of reduction / ROI, but delivered as a by-product.



Unite Disconnected Software Delivery Disciplines

testhouse

The organisation must unite disconnected software delivery disciplines if they are to increase **quality** and **productivity** and benefit from the latest methodology innovations, at scale for:

● Agile ● Lean Software ● DevOps

All of the above initiatives require automated integration of tools to eliminate friction points, enabling teams to work with the same information in near real time.

Additionally, emerging platforms, such as Mobile & Cloud need to adhere to a joined up feedback loop for both PAAS & SAAS.

Integration among these tools is a must.

Plan for traceability at onset

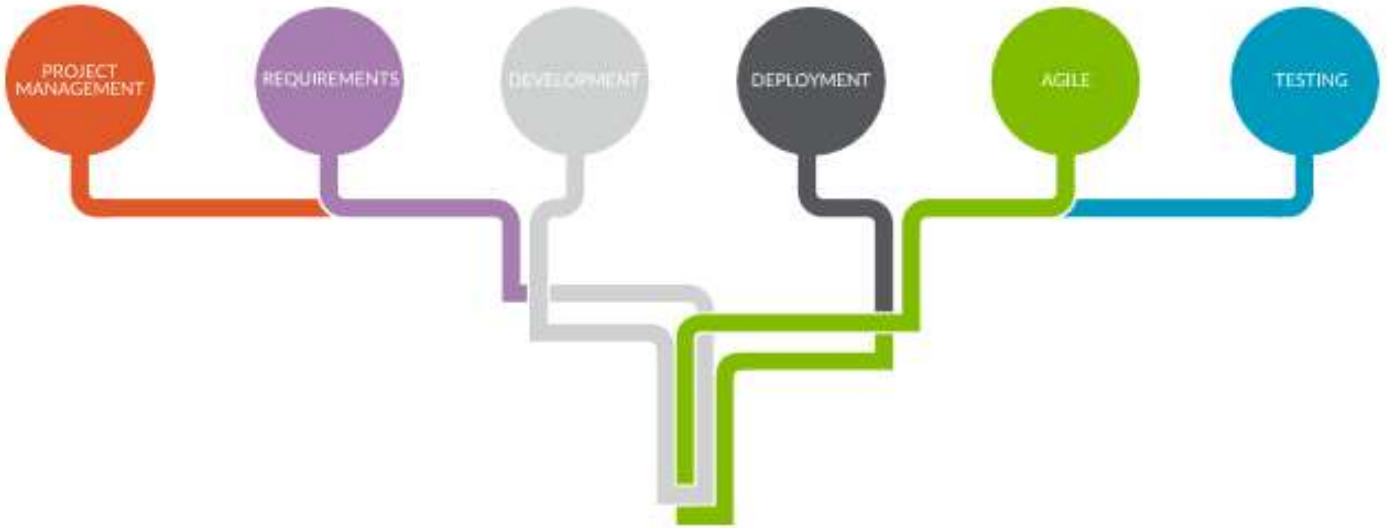
- Reporting is important for the purposes of delivering the project to **ensure** that the project conforms to **corporate requirements.**

- Supporting **compliance** requires cross-tool **traceability** and reporting.

- The cross-organizational infrastructure necessary to connect the practice of software delivery should include:
  - Integrated architecture
  - Process monitoring
  - Measurement
  - Reporting
  - Dashboarding

Braid the delivery disciplines

Unlike piecemeal integration solutions, true software lifecycle integration solutions should..



- **Connect** all software development and delivery tools.
- Synchronize information across disciplines.
- Unlock cross-discipline value through shared information.
- Give practitioners real-time access to the changes other team members are making in their tools.
- Improve visibility across **the entire project team** for **traceability** and **compliance**.
- Enable **end-to-end reporting**.

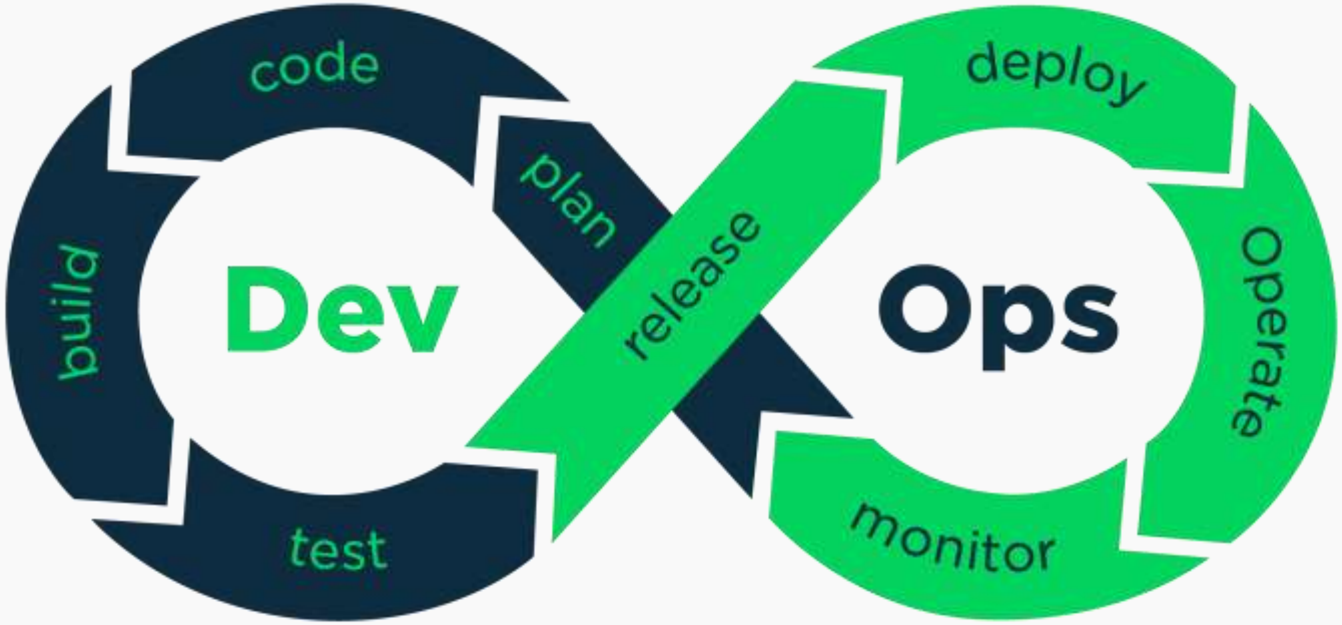
Develop a foundational DevOps methodology

The traditional waterfall-based solution development lifecycle should be modified to:

- Involve business, security, development and production teams from the beginning to properly define functional, technical and infrastructure requirements.
- Inform the production team of development progress so that issues are defined early on, and the production environment is ready to go once development and testing is completed.
- Establish a process for continual feedback.

DevOps

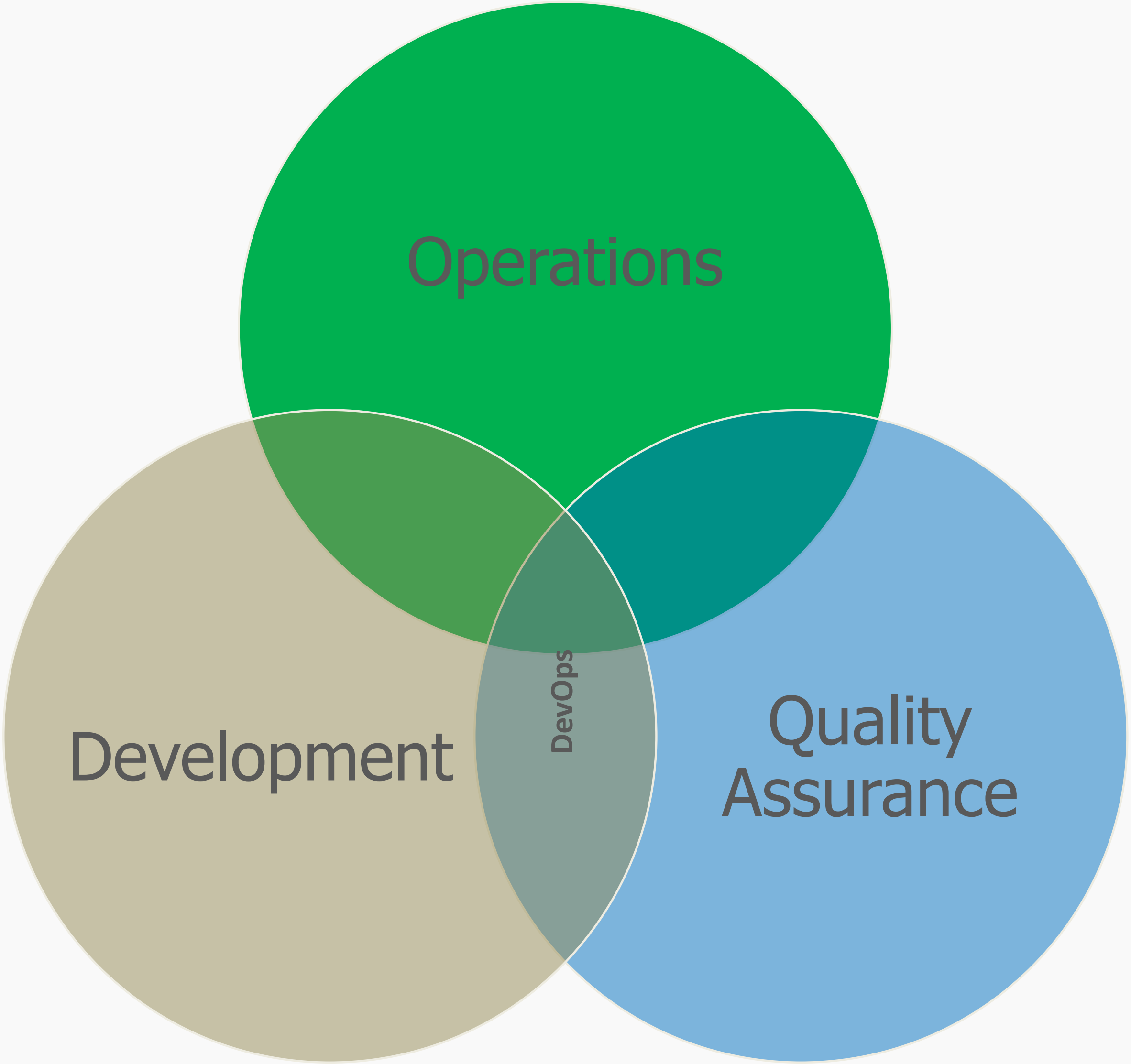
- According to the DevOps culture, a single group of Engineers (developers, system admins, QA's. Testers etc turned into DevOps Engineers) has end to end responsibility of the Application (Software) right from gathering the requirement to development, to testing, to infrastructure deployment, to application deployment and finally monitoring & gathering feedback from the end users, then again implementing the changes



Principles of DevOps?

- Effective Collaboration
- Increased operational agility
- Quicker release of software to market
- Automating the design, development, QA and deployment of new systems and applications
- Develop and test the product in an environment which is very much similar to production environment
- Frequently deploy the builds
- Continuously validate the quality of operation

DevOps is an Intersection



How DevOps Solved Dev Challenges

Ops Challenges	DevOps Solution
Waiting time for code deployment	• Continuous Integration ensures that there is quick deployment of code, faster testing and speedy feedback mechanism.
Work pressure on old, pending and new code	• No waiting time for code deployment. Hence the developer focuses on building the current code.

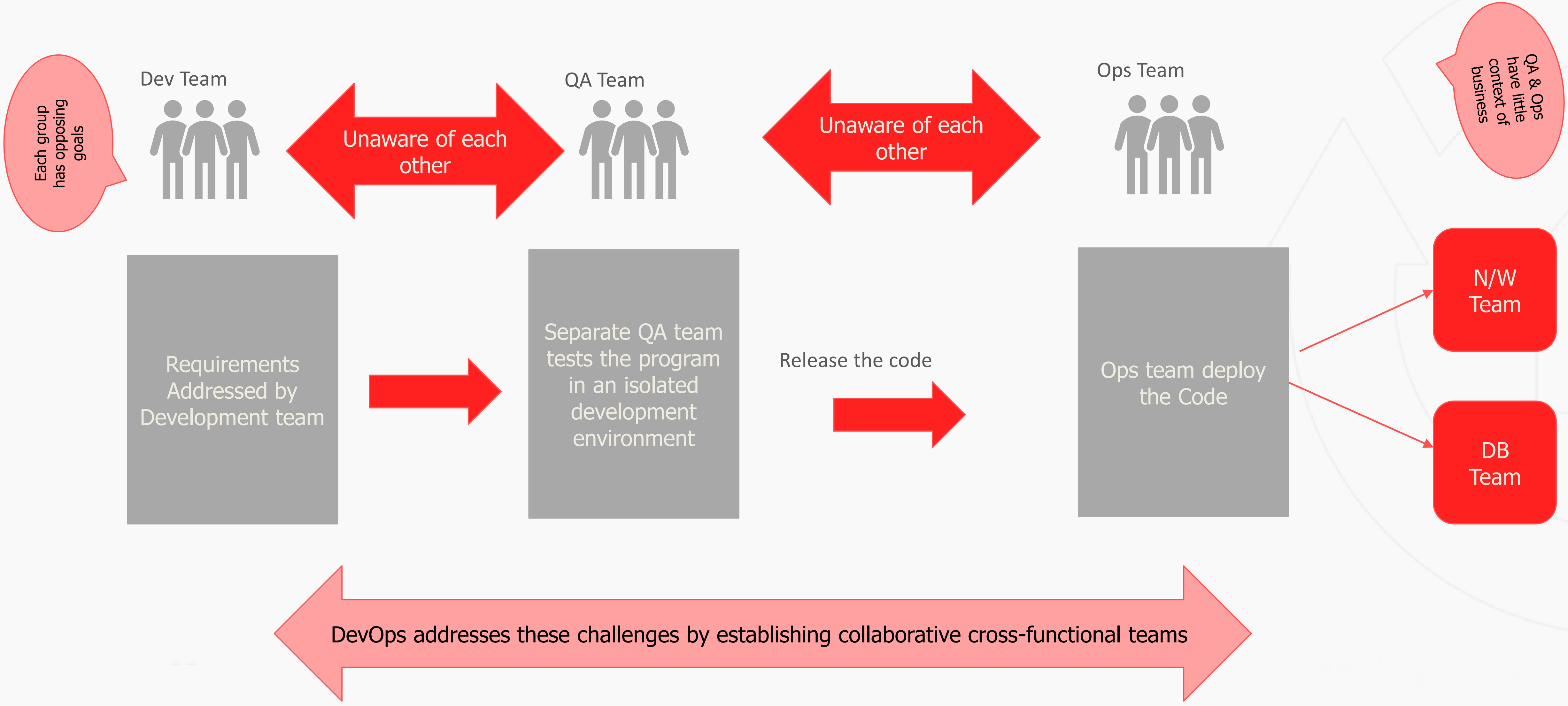
How DevOps Solved Ops Challenges

Ops Challenges	DevOps Solution
Difficult to maintain uptime of the production environment	**Containerization/virtualization** ensures there is a simulated environment created to run the software as containers offer great reliability for service uptime
Tools to automate infrastructure management are not effective	**Configuration management** helps you to organize and execute configuration plans, consistently provisioning the system, and proactively manage their infrastructure.
No. of servers to be monitored increases	**Continuous monitoring**: Effective monitoring and feedback system is established through Nagios. Thus effective administration is assured
Difficult to diagnose and provide feedback on the product	

Challenges Solved by DevOps

- Prior to DevOps application development, teams were in charge of gathering business requirements for a software program and writing code
- A separate QA team tested the program in an isolated development environment (if requirements were met) and releases the code for operations to deploy
- When teams work separately:
  - Dev is unaware of QA and Ops. So product may not work anticipated
    - QA and Ops have little context of the business purpose and value of the software
    - Different goals of the team result in inefficiency and criticism when software damages happen
- DevOps addresses above issues by establishing collaborative cross-functional teams that share responsibility for maintaining the system that runs the software and preparing the software to run on that system with increased quality feedback and automation issues

Challenges Solved by DevOps



Each group has opposing goals

Dev Team

Unaware of each other

QA Team

Unaware of each other

Ops Team

QA & Ops have little context of business

Requirements Addressed by Development team

Separate QA team tests the program in an isolated development environment

Release the code

Ops team deploy the Code

N/W Team

DB Team

DevOps addresses these challenges by establishing collaborative cross-functional teams

Pre-DevOps Scenario

- The Dev team kicks a new release “over the wall” to QA
- When the testers bring their findings to Dev, the developers become defensive and blame the testers that are testing the environment for the bugs
- The testers respond that it isn’t their testing environment, but the developer’s code that is the problem
- Eventually QA kicks the debugged new release “over the wall” to Ops
- The Ops team’s goal is to limit changes to their system, but they apprehensively release the code and the system crashes. Ops says that Dev provided them faulty artefacts
- Dev says everything worked fine in the test environment
- The fire drills begin to debug the system and get production stable. The production environment isn’t Dev’s and QA’s responsibility, so they keep hands off while Ops spends all night fixing the production issues

POST-DevOps Scenario

- The software team meets prior to starting a new software project. The team includes developers, testers, operations and support professionals. This team plans how to create working software that is ready for deployment
- Each day new code is deployed as the developers complete it. Automated testing ensures the code is ready to be deployed
- After the code passes all the automated testing it is deployed to a small number of users. The new code is monitored for a short period to ensure there are no unforeseen problems and it is stable
- The new code is then proliferated to the remaining users once the monitoring shows that it is stable

DevOps can be a blend of culture, tools and maturity that make sense for your organization

Reasons for DevOps

- The existence of traditional team structures that did not scale to meet the varied needs of modern enterprises

- The disconnect between development, testing and operations teams, which resulted in poor communication, collaboration and integration

- The accelerating progression of digital technologies, which has evolved faster than the underlying processes used to deploy, extend and manage them

DevOps

- Dev team sits with Ops team to understand the impact of code changes and works more closely with systems which are equivalent to production environment
- Dev team give more attention to metrics used by Ops team
- Ops will have more understanding on infrastructure needs and apply more automation for deployment
- Development-Test-Production tunnel could be closely monitored for each deployment
- Better collaboration and communication

Agile Development

- Agile Development is an umbrella term for several iterative and incremental software development methodologies
- The most popular agile methodologies include: Scrum, Kanban, Scaled Agile Framework , Lean Development and Extreme Programming (XP)
- While each of the agile methodologies is unique in its specific approach, they all share a common vision and core values. They all fundamentally incorporate iteration and the continuous feedback that it provides to successively refine and deliver a software system
- They all involve continuous planning, continuous testing, continuous integration, and other forms of continuous evolution of both the project and the software
- In the beginning, agile teams were primarily made up of developers. As these agile teams became more effective and efficient at producing software, it became clear that having Quality Assurance (QA) and Dev as separate teams was inefficient. Agile grew to encompass QA in order to increase the velocity of delivering software and now agile is once again growing to encompass the delivery and support members to extend agility from ideation to delivery.

Agile and DevOps

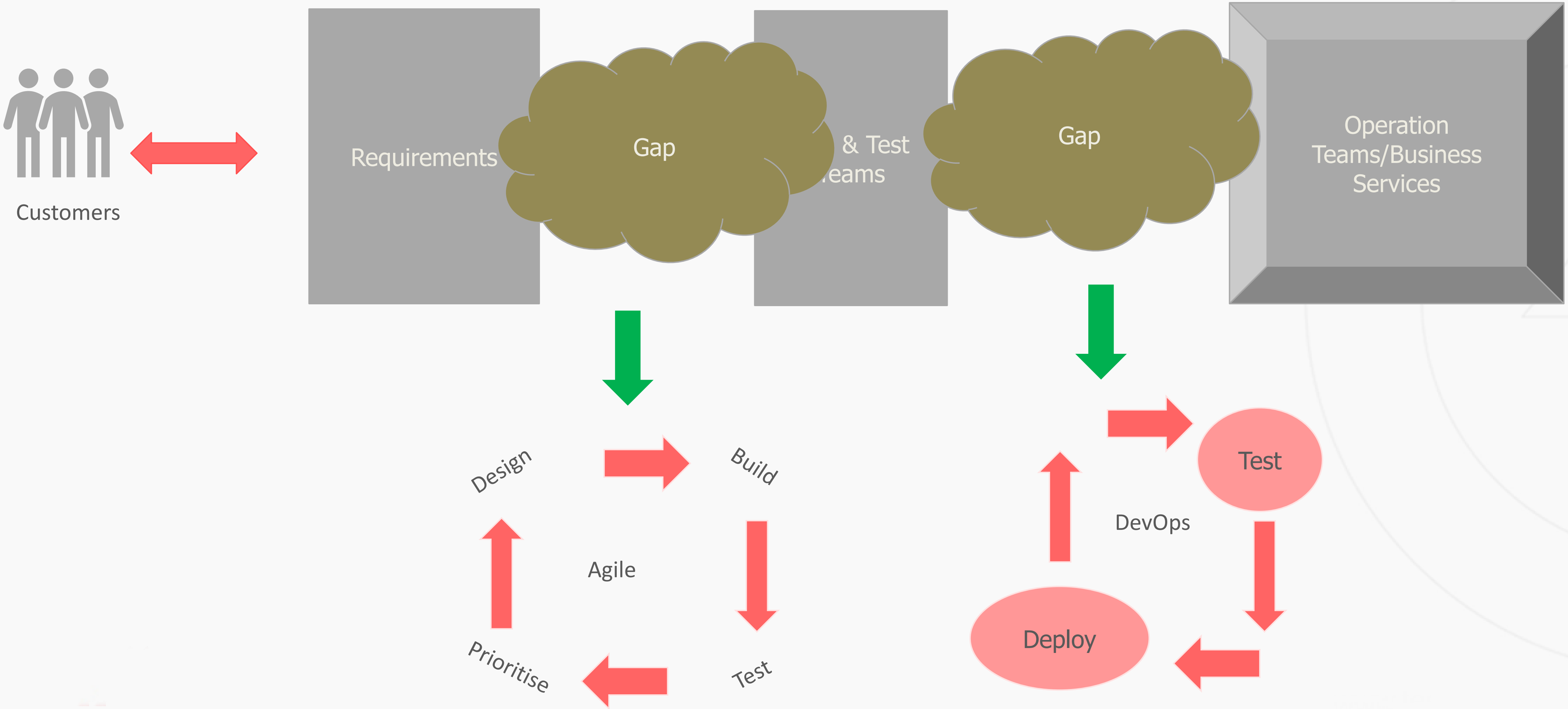
Agile Development

- Focuses on the gap between customer requirements and dev + testing team
- Consists of cross-functional team who performs design, development and test based on customer preferences
- Focuses on functional and non-functional readiness

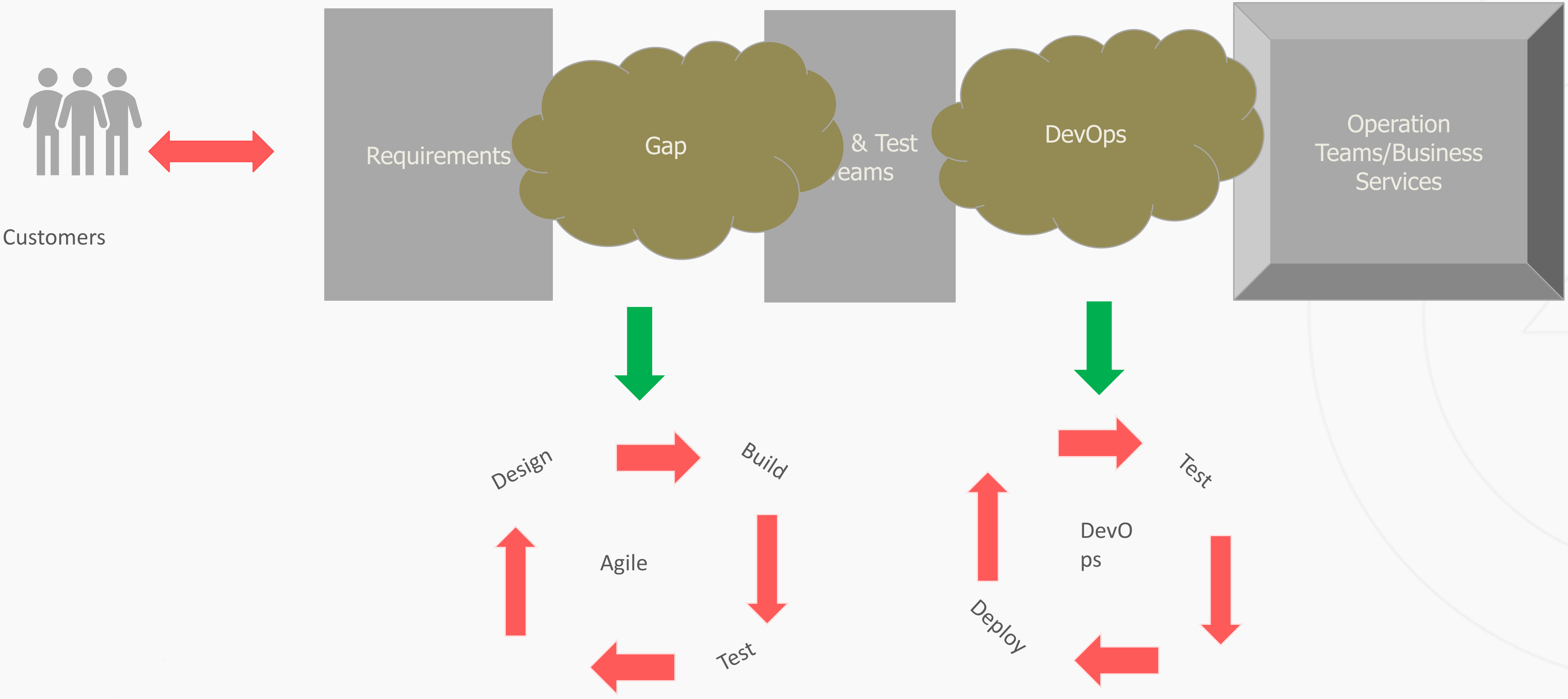
DevOps

- Focuses on the gap between dev + testing team and operational team
- Automated release management
- Focuses on functional, non-functional, operational and business readiness

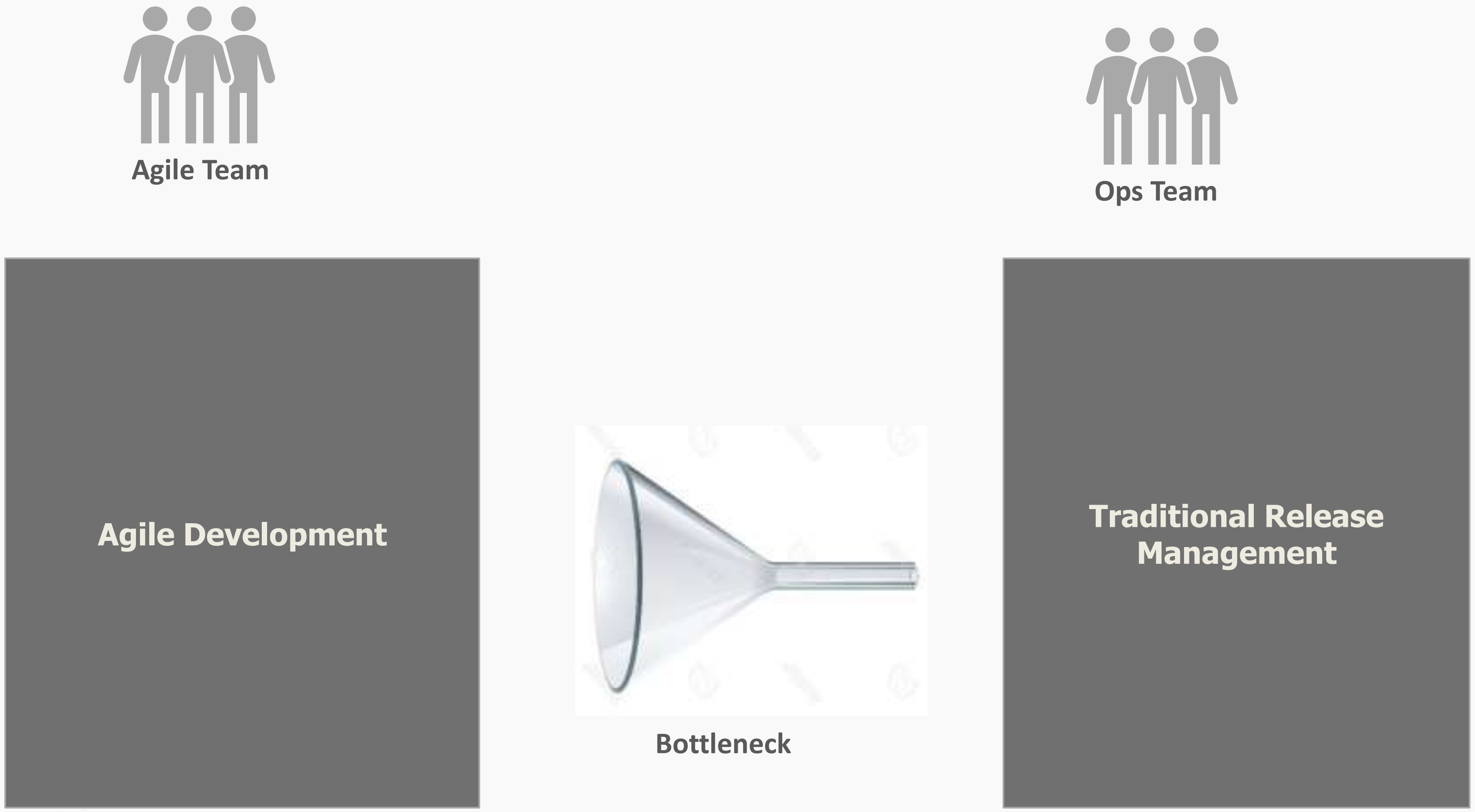
Agile & DevOps – Addressing Delivery Challenges



DevOps Vs Release Management – Addressing Delivery Challenges



Agile + Release Management



Agile Team

Ops Team

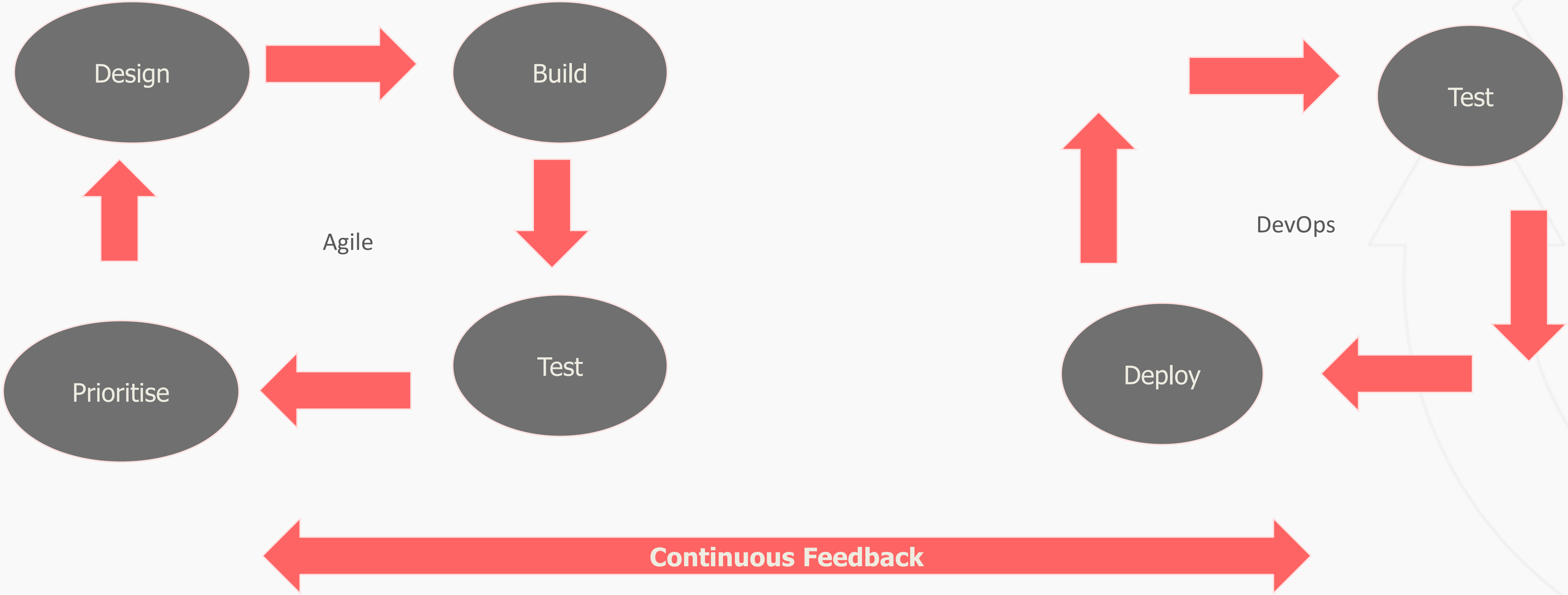
Agile Development

Bottleneck

Traditional Release Management

Agile + DevOps

Continuous Integration extended as Continuous Delivery



Faster delivery reduces risk

DevOps Practices

- One fundamental practice is to perform very frequent but small updates. This is how organizations innovate faster for their customers. These updates are usually more incremental in nature than the occasional updates performed under traditional release practices. Frequent but small updates make each deployment less risky. They help teams address bugs faster because teams can identify the last deployment that caused the error. Organizations using a DevOps model deploy updates much more often than organizations using traditional software development practices.
- Organizations might also use a microservices architecture to make their applications more flexible and enable quicker innovation. The microservices architecture decouples large, complex systems into simple, independent projects. Applications are broken into many individual components (services) with each service scoped to a single purpose or function and operated independently of its peer services and the application as a whole. This architecture reduces the coordination overhead of updating applications, and when each service is paired with small, agile teams who take ownership of each service, organizations can move more quickly.

DevOps Practices

- The combination of microservices and increased release frequency leads to significantly more deployments which can present operational challenges. Thus, DevOps practices like continuous integration and continuous delivery solve these issues and let organizations deliver rapidly in a safe and reliable manner. Infrastructure automation practices, like infrastructure as code and configuration management, help to keep computing resources elastic and responsive to frequent changes. In addition, the use of monitoring and logging helps engineers track the performance of applications and infrastructure so they can react quickly to problems.
- Together, these practices help organizations deliver faster, more reliable updates to their customers.

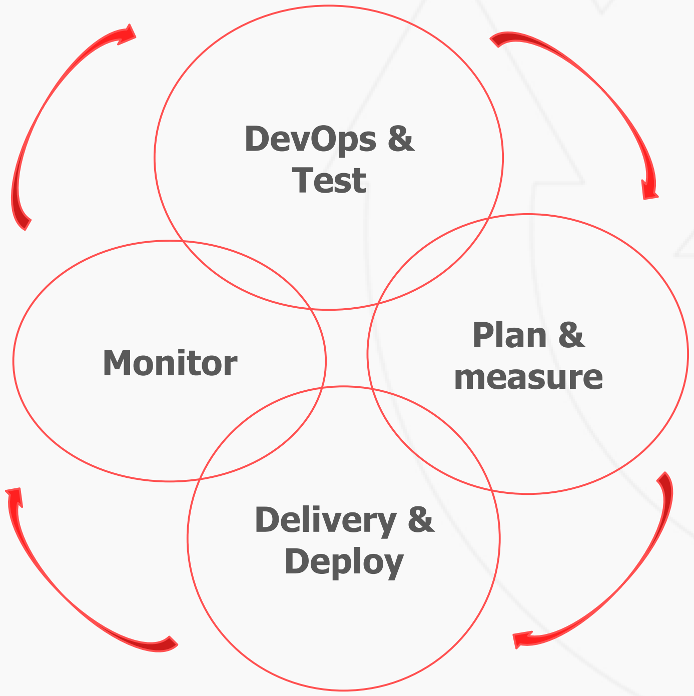
4 key Activities of DevOps Practice

Collaborative Development:
Increased collaboration between teams

Continuous Integration & Continuous Testing:
Integration of software testing with deployment and operations

Continuous Delivery and Deployment:
Increased delivery speed and frequency

Continuous Monitoring:
Improved quality by monitoring production performance



DevOps & Test

Plan & measure

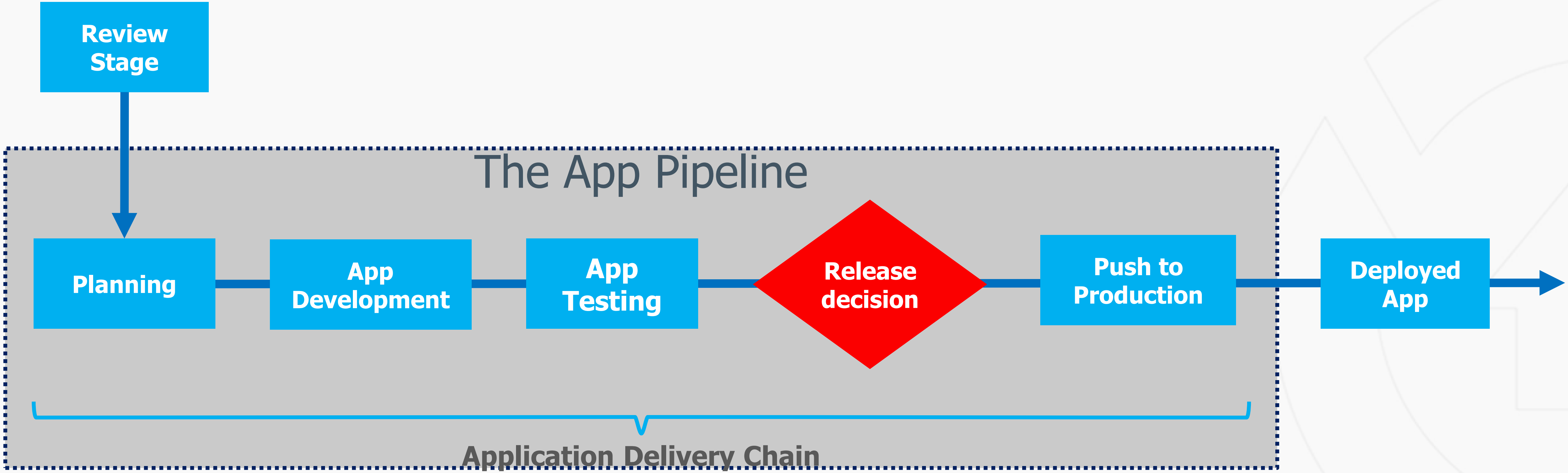
Delivery & Deploy

Monitor

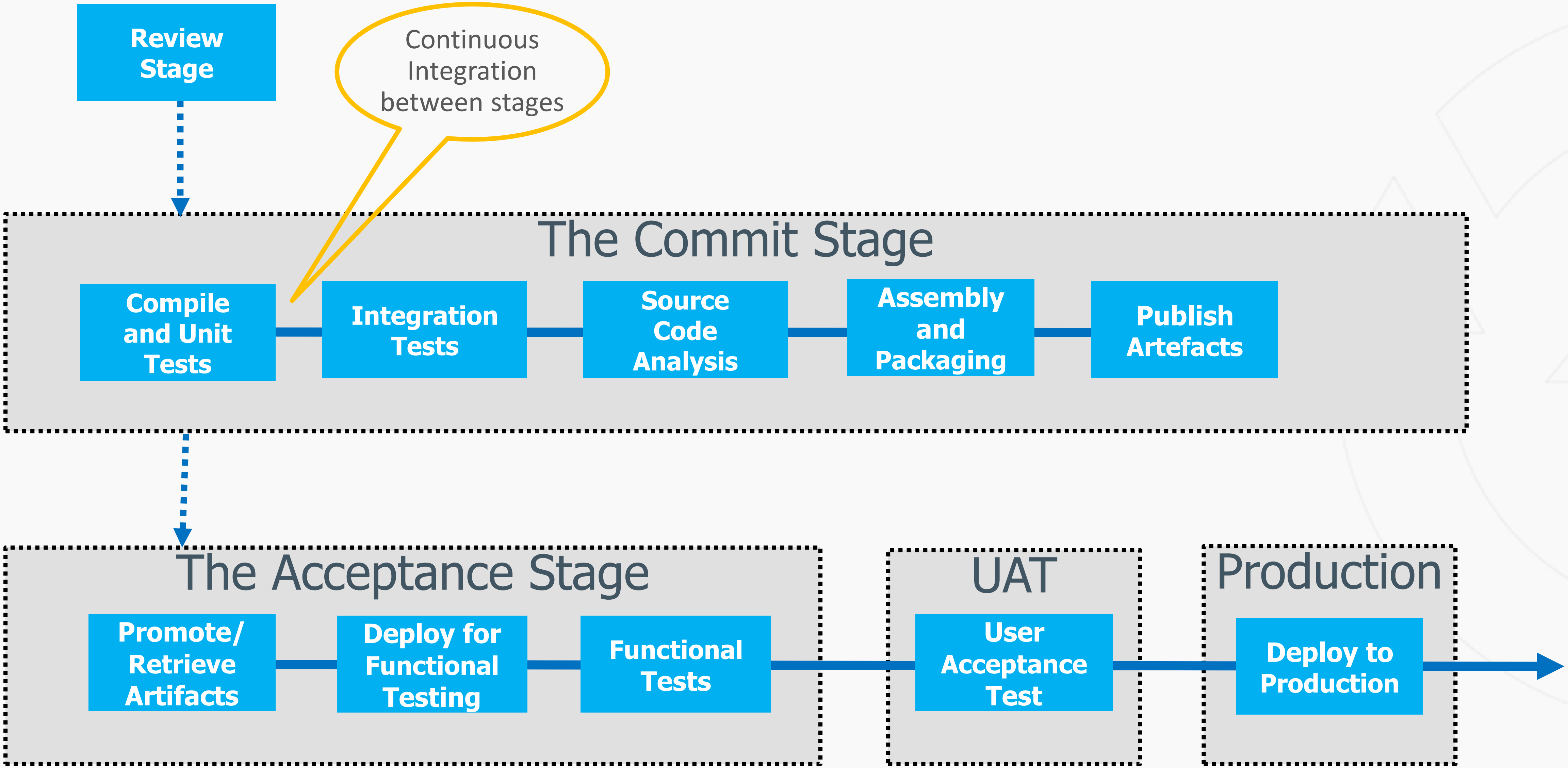
Communication & Collaboration

- Increased communication and collaboration in an organization is one of the key cultural aspects of DevOps
- The use of DevOps tooling and automation of the software delivery process establishes collaboration by physically bringing together the workflows and responsibilities of development and operations
- Building on top of that, these teams set strong cultural norms around information sharing and facilitating communication through the use of chat applications, issue or project tracking systems, and wikis
- This helps speed up communication across developers, operations, and even other teams like marketing or sales, allowing all parts of the organization to align more closely on goals and projects

Essentials steps to Deployment



Continuous Delivery Pipeline is achieved via Continuous Integration

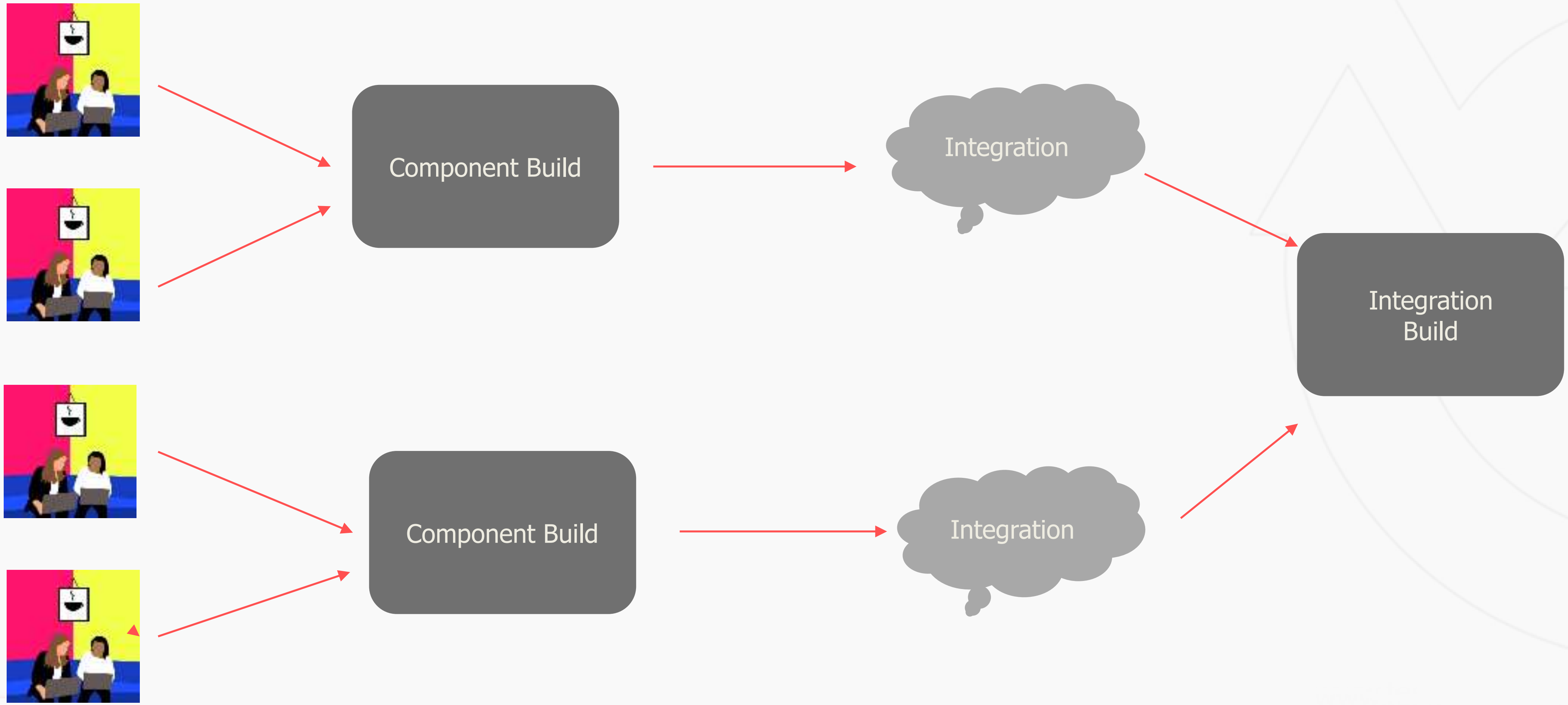


Continuous Integration

- Continuous integration is the practice of quickly integrating newly developed code with the main body of code that is to be released
- Continuous integration saves a lot of time when the team is ready to release the code.
- Automation is required to successfully execute continuous integration
- Continuous integration is often the first step down the path toward DevOps maturity
- The continuous integration process from a DevOps perspective involves checking your code in, compiling it into usable (often binary executable) code and running some basic validation testing
- The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous Integration

The code changes by each developer in integrated to ensure the main branch is up to date



Dinner – Day 1

Continuous Testing

- Continuous testing is the first step in the right direction when embarking on a DevOps journey
- Continuous testing is a metaphor for a continuous feedback mechanism that drives software delivery through the SDLC tunnel
- Automated feedback at each checkpoint is an auto-trigger for the next process in the delivery chain if the feedback is to move forward, or green
- If the feedback is to not move forward, the process immediately is stopped, and corrective measures are taken

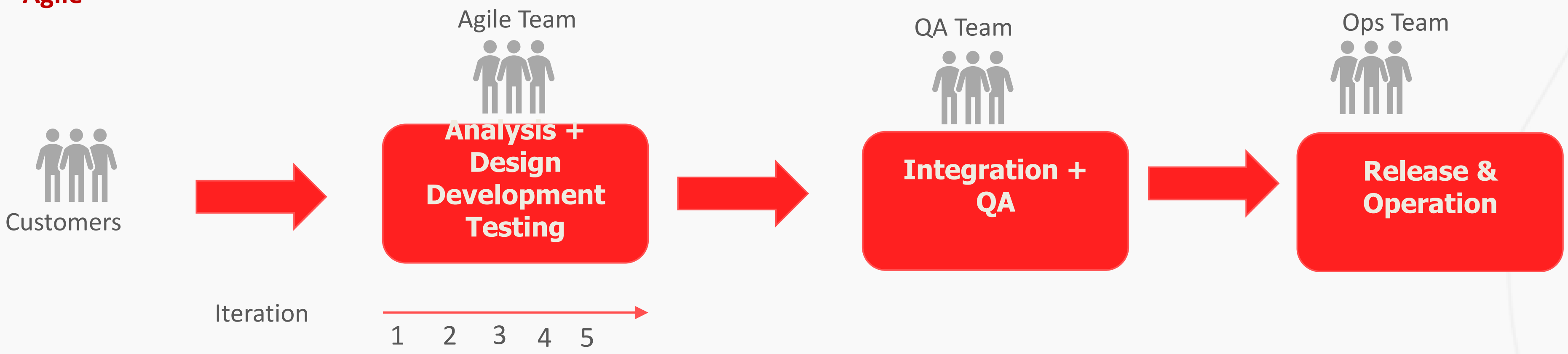
Continuous Delivery

- Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared, for a release to production
- Continuous delivery is an extension of continuous integration. It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. It sits on top of continuous integration
- When executing continuous delivery, you add additional automation and testing so that you don't just merge the code with the main code line frequently, but you get the code nearly ready to deploy with almost no human intervention
- When continuous delivery is implemented properly, developers will always have a deployment-ready build artefact that has passed through a standardized test process
- It's the practice of having the code base continuously in a ready-to-deploy state

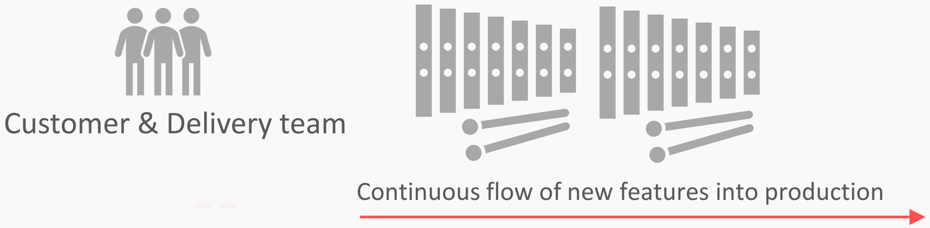
Continuous Delivery

Run each CI build through deployment procedures on production

Agile



Continuous Delivery



Automated test driven development process
Collaboration between business and delivery teams
Cross functional teams including Q and operations
Continuous integration and incremental development in the mainstream
Software is production ready always
Release always mapped to business needs, not to operational constraints

Continuous Deployment

testhouse

- Continuous deployment is the most advanced evolution of continuous delivery. It’s the practice of deploying all the way into production without any human intervention
- Teams that utilize continuous delivery don’t deploy untested code; instead, newly created code runs through automated testing before it gets pushed out to production
- The code release typically only goes to a small percentage of users and there’s an automated feedback loop that monitors quality and usage before the code is propagated further

Continuous Monitoring & Logging

- Organizations monitor metrics and logs to see how application and infrastructure performance impacts the experience of their product's end user
- By capturing, categorizing, and then analysing data and logs generated by applications and infrastructure, organizations understand how changes or updates impact users, shedding insights into the root causes of problems or unexpected changes
- Active monitoring becomes increasingly important as services must be available 24/7 and as application and infrastructure update frequency increases
- Creating alerts or performing real-time analysis of this data also helps organizations more proactively monitor their services

Other Practices

- Microservices
- Infrastructure as Code
- Configuration Management

Microservices

- The microservices architecture is a design approach to build a single application as a set of small services
- Each service runs in its own process and communicates with other services through a well-defined interface using a lightweight mechanism, typically an HTTP-based application programming interface (API)
- Microservices are built around business capabilities; each service is scoped to a single purpose
- You can use different frameworks or programming languages to write microservices and deploy them independently, as a single service, or as a group of services

Infrastructure as Code

- Infrastructure as code is a practice in which infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration
- The cloud's API-driven model enables developers and system administrators to interact with infrastructure programmatically, and at scale, instead of needing to manually set up and configure resources
- Engineers can interface with infrastructure using code-based tools and treat infrastructure in a manner similar to how they treat application code
- Because they are defined by code, infrastructure and servers can quickly be deployed using standardized patterns, updated with the latest patches and versions, or duplicated in repeatable ways

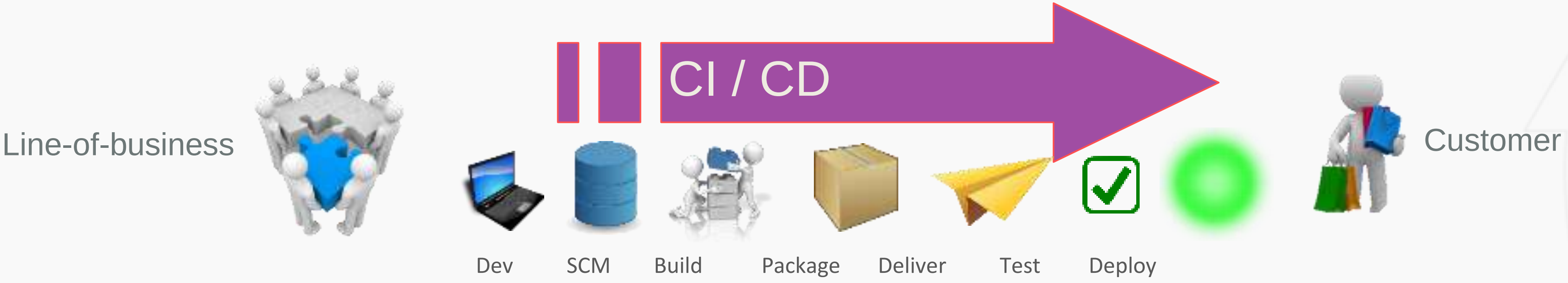
Configuration Management

- Developers and system administrators use code to automate operating system and host configuration, operational tasks, and more
- The use of code makes configuration changes repeatable and standardized
- It frees developers and systems administrators from manually configuring operating systems, system applications, or server software

DevOps vs Agile

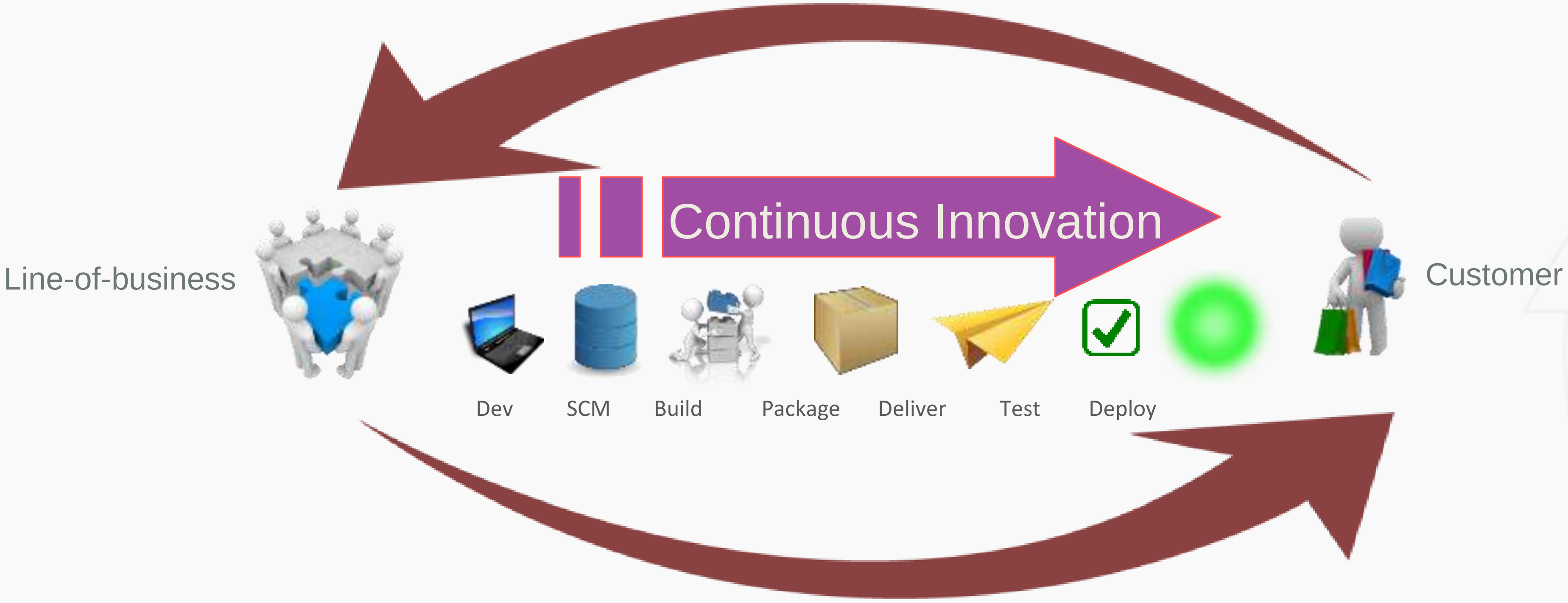
Testing in Agile	Testing in DevOps
Test as early and as often as possible	Test continuously
Automate testing as much as possible	Automate almost everything
Continuous integration and testing is a step forward	Continuous integration and testing is mandatory
Potentially shippable code at the end of a sprint	Potentially shippable code following every integration

Three Ways of DevOps
Systems Thinking



Getting ideas into production quickly, in small batches, Automating the process

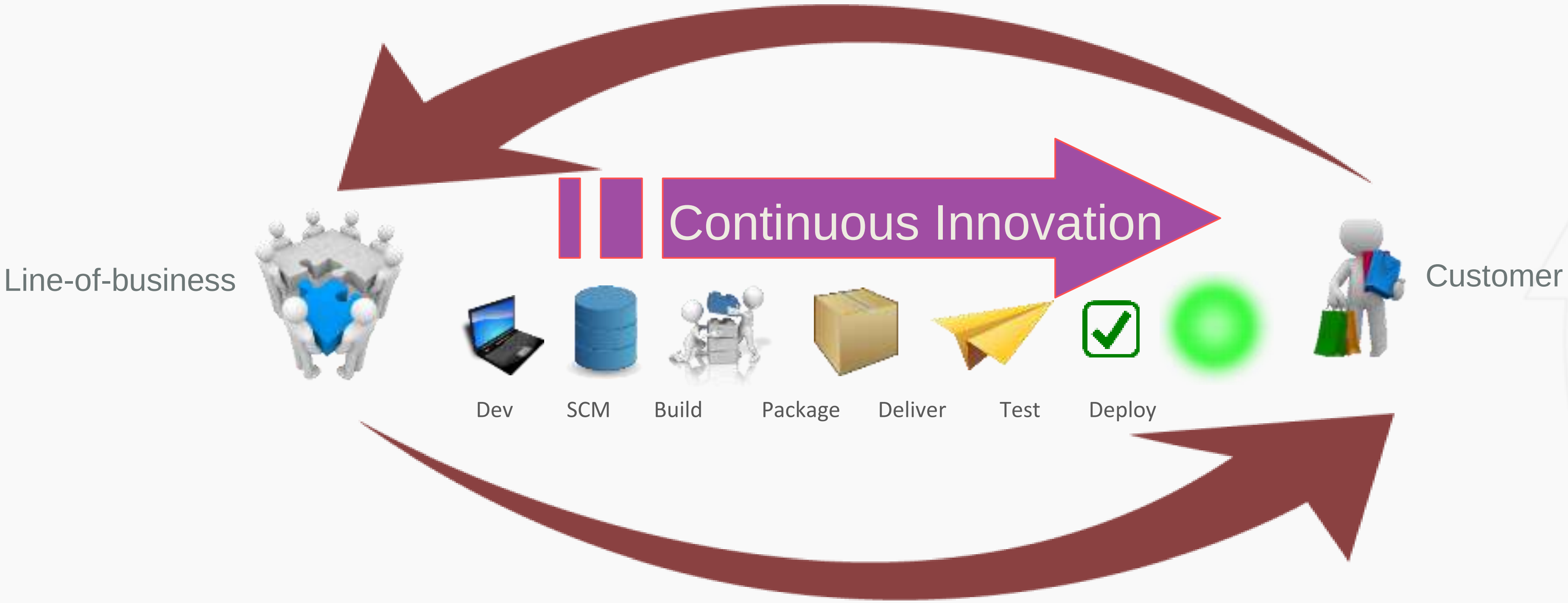
Three Ways of DevOps
Feedback Loops



- Getting ideas into production quickly, in small batches
- Getting people to use it
- Getting their feedback

- Measure and Monitor
- Are we building the right product?
- Understanding and responding to all customers
- Introduce faults to the system

Three Ways of DevOps
Culture of Experimentation



- Getting ideas into production quickly, in small batches
- Getting people to use it
- Getting their feedback
- Experiment and reduce MTTR

- Experiment and introduce failure practice makes perfect
- Allocate time to experiment
- Allow people to take risks
- Introduce faults to the system

The CAMs Model

CAMS is the migration from the CALMS Model which seems to have dropped the L for Lean

- Culture
  - Absolutely essential in making the DevOps implementation a success
  - Bringing the whole team together business, development, test and operations
- Automation
  - Automate all manual and repeatable processes
- Measurement
  - The glue to bringing it all together
- Sharing
  - Collaboration
  - Visibility
  - Transparency

DEVOPS BENEFITS

STRONG IT PERFORMANCE IS A COMPETITIVE ADVANTAGE

Firms with high-performing IT organizations were 2x as likely to exceed their profitability, market share, and productivity goals

DEVOPS PRACTICES IMPROVE IT PERFORMANCE

DEPLOY CODE 30X FASTER

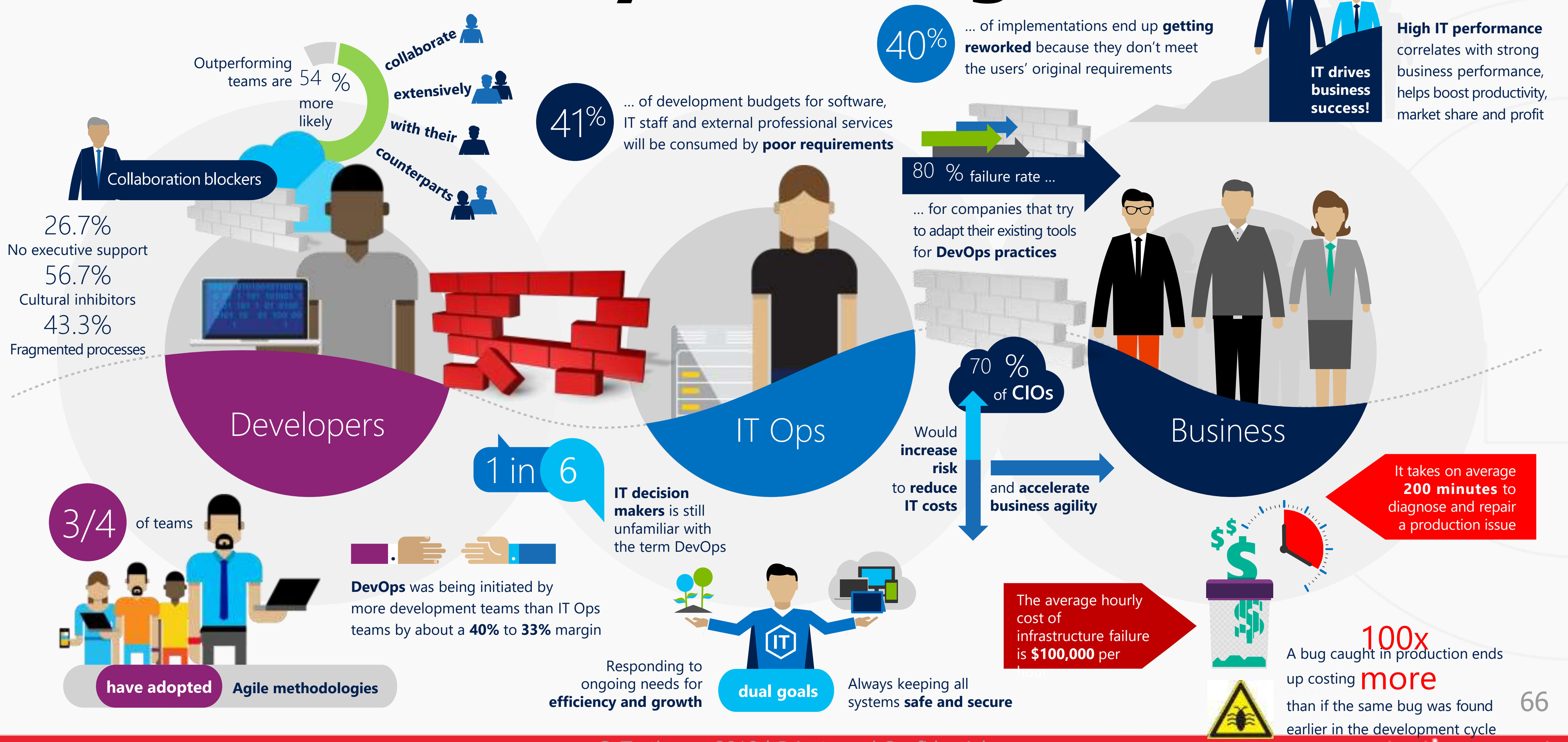
… and with 200x shorter lead time when compared to their lower-performing peers

HAVE 60X FEWER FAILURES

… and recover from failure 168X faster when compared to their lower-performing peers

Value Delivery Challenges

testhouse



Outperforming teams are 54 % more likely collaborate extensively with their counterparts

Collaboration blockers

26.7%
No executive support

56.7%
Cultural inhibitors

43.3%
Fragmented processes

41% … of development budgets for software, IT staff and external professional services will be consumed by **poor requirements**

40% … of implementations end up **getting reworked** because they don't meet the users' original requirements

80 % failure rate …

… for companies that try to adapt their existing tools for **DevOps practices**

IT drives business success!

High IT performance correlates with strong business performance, helps boost productivity, market share and profit

1 in 6 **IT decision makers** is still unfamiliar with the term DevOps

70 % of **CIOs**

Would **increase risk** to reduce **IT costs** and accelerate **business agility**

3/4 of teams **have adopted** **Agile methodologies**

DevOps was being initiated by more development teams than IT Ops teams by about a **40%** to **33%** margin

Responding to ongoing needs for **efficiency and growth**

dual goals

Always keeping all systems **safe and secure**

The average hourly cost of infrastructure failure is **$100,000** per hour

It takes on average **200 minutes** to diagnose and repair a production issue

A bug caught in production ends up costing 100x more than if the same bug was found earlier in the development cycle

Value of DevOps

DevOps focuses heavily on establishing a collaborative culture and improving efficiency through automation with DevOps tools.

DevOps Culture

DevOps culture is characterized by increased collaboration, decreasing silos, shared responsibility, autonomous teams, improving quality, valuing feedback and increasing automation. Many of the DevOps values are agile values as DevOps is an extension of agile.

Agile methods are a more holistic way of delivering software. Agile development teams measure progress in terms of working software. Product owners, developers, testers and UX people work closely together with the same goals.

Value of DevOps

DevOps Culture

- DevOps is just adding the operations' mindset and maybe a team member with some of those responsibilities into the agile team. Whereas before DevOps progress is measured in terms of working software, with DevOps progress is measured in terms of working software in the customer's hands.

- To achieve this, Dev and Ops must break down the silos and collaborate with one another, share responsibility for maintaining the system that runs the software, and prepare the software to run on the system with increased quality feedback and delivery automation.

Value of DevOps

- Transitioning to DevOps requires a change in culture and mindset.
- At its simplest, DevOps is about removing the barriers between two traditionally siloed teams, development and operations.
- With DevOps, the two teams work together to optimize both the productivity of developers and the reliability of operations.
- They strive to communicate frequently, increase efficiencies, and improve the quality of services they provide to customers.
- They take full ownership for their services, often beyond where their stated roles or titles have traditionally been scoped by thinking about the end customer’s needs and how they can contribute to solving those needs.
- Quality assurance and security teams may also become tightly integrated with these teams.
- Organizations using a DevOps model, regardless of their organizational structure, have teams that view the entire development and infrastructure lifecycle as part of their responsibilities.

Implementing DevOps

testhouse

DevOps Tools

- DevOps tools consist of configuration management, test and build systems, application deployment, version control and monitoring tools
- Continuous integration, continuous delivery and continuous deployment require different tools

- To expedite and actualize DevOps process apart from culturally accepting it, one also needs various DevOps tools like Puppet, Jenkins, GIT, Chef, Docker, Selenium, AWS etc to achieve automation at various stages which helps in achieving Continuous Development, Continuous Integration, Continuous Testing, Continuous Deployment, Continuous Monitoring to deliver a quality software to the customer at a very fast pace

Tools for DevOps

testhouse

Source Code Repository

- A source code repository is a place where developers check in and change code. The source code repository manages the various versions of code that are checked in, so developers don’t write over each other’s work.

- Source control has probably been around for forty years, but it’s a major component of continuous integration. Popular source code repository tools are Git, Subversion, Cloudforce, Bitbucket and TFS

Build Server

- The build server is an automation tool that compiles the code in the source code repository into executable code base. Popular tools are Jenkins, SonarQube and Artifactory.

Configuration Management

- Configuration management defines the configuration of a server or an environment. Popular configuration management tools are Puppet and Chef.

Tools for DevOps

Virtual Infrastructure

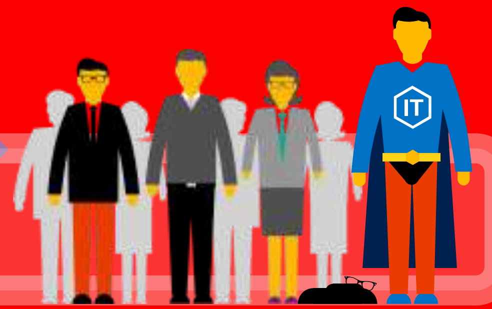
- Amazon Web Services and Microsoft Azure are examples of virtual infrastructures
- Provided by cloud vendors that sell infrastructure or platform as a service (PaaS)
- APIs to allow you to programmatically create new machines with configuration management tools such as Puppet and Chef
- There are also private clouds. For example, VMware has vCloud.
- Virtual infrastructures combined with automation tools to empower organizations practicing DevOps with the ability to configure a server without any fingers on the keyboard. If you want to test your brand-new code, you can automatically send it to your cloud infrastructure, build the environment and then run all of the tests without human intervention

Test Automation

- DevOps testing focuses on automated testing within your build pipeline to ensure that by the time that you have a deployable build, you are confident it is ready to be deployed
- Popular tools are Selenium and Water

Microsoft Tooling

People | Process | Tools



Develop

Developer IDE

Team Collaboration

- Visual Studio Team Services
- Visual Studio Team Foundation Server

Build + Test

Build/CI

- Visual Studio Team Services
- Visual Studio Team Foundation Server

Test

- Visual Studio Team Services
- Visual Studio Team Foundation Server

Microsoft Test Manager

Deploy

Release/CD

Microsoft System Center

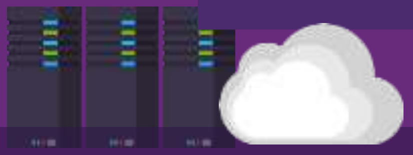
Release Management for Visual Studio

Automation Service

PowerShell | WAML

Azure Resource Management

xPlat Command Line



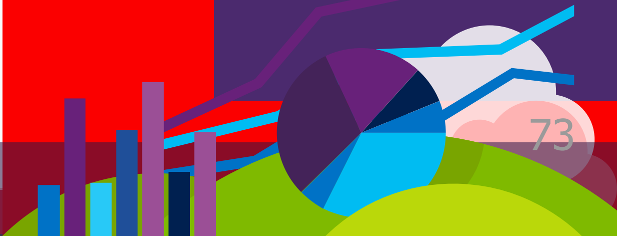
Monitor + Learn

Monitor

Microsoft System Center

- Visual Studio Team Services
- Visual Studio Team Foundation Server

Application Insights



On-Premises | Hybrid | Cloud

OSS Tooling

People | Process | Tools

Develop

Developer IDE

Team Collaboration

GitHub

CodePlex

Build + Test

Build/CI

gradle

GRUNT

Jenkins

Hudson

Test

gradle

GRUNT

Deploy

Configuration

puppet labs

CHEF

Release

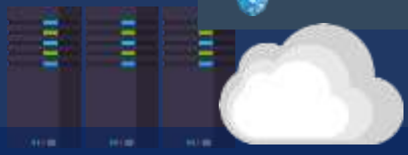
gradle

GRUNT

Jenkins

Hudson

VAGRANT



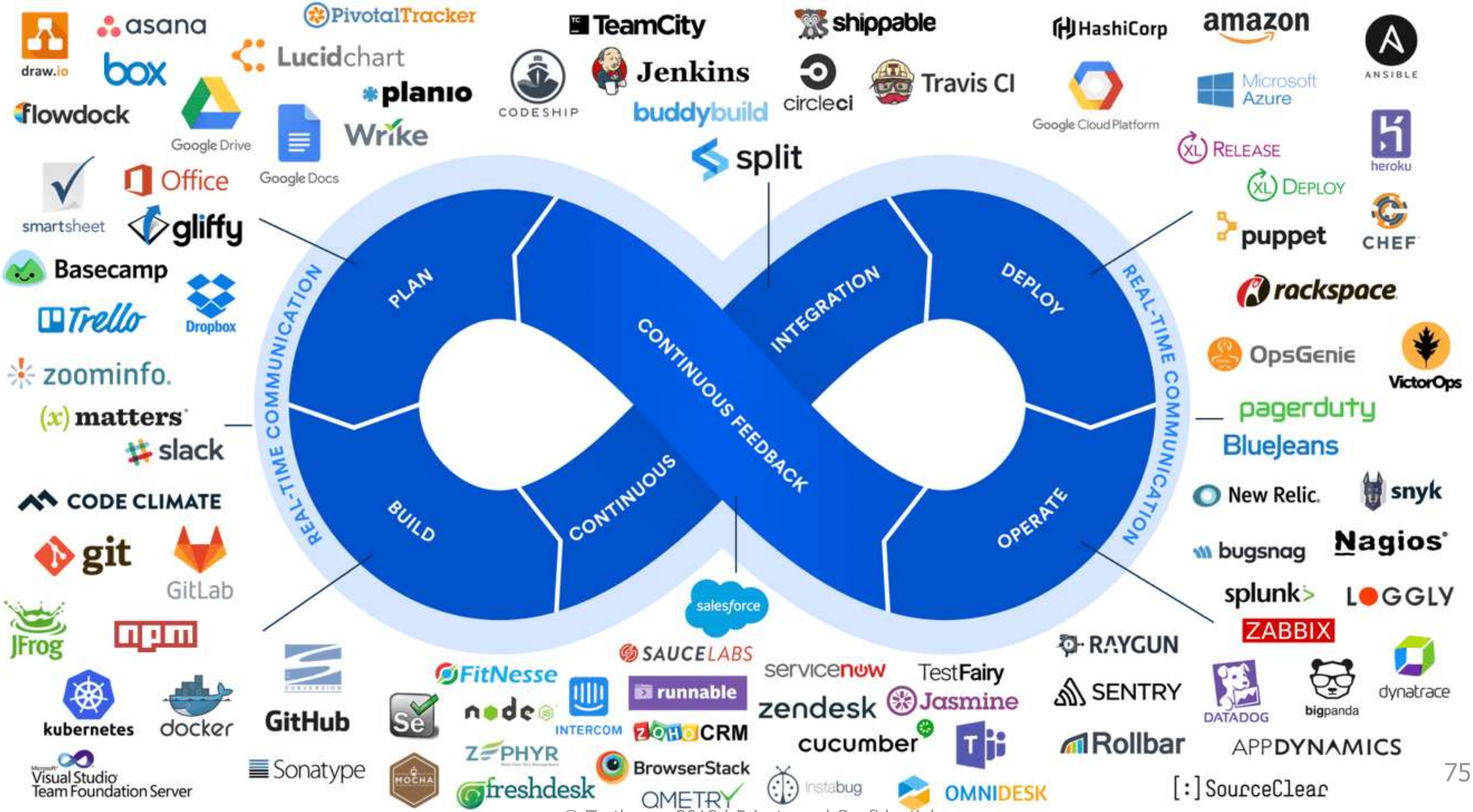
Monitor + Learn

Monitor

Nagios

ZABBIX

On-Premises | Hybrid | Cloud


PLAN
BUILD
CONTINUOUS
INTEGRATION
DEPLOY
OPERATE
CONTINUOUS FEEDBACK
REAL-TIME COMMUNICATION
REAL-TIME COMMUNICATION
draw.io
asana
box
flowdock
PivotalTracker
Lucidchart
planio
Google Drive
Wrike
Google Docs
smartsheet
Office
gliffy
Basecamp
Trello
Dropbox
zoominfo.
(x) matters
slack
CODE CLIMATE
git
GitLab
JFrog
npm
kubernetes
docker
Visual Studio Team Foundation Server
GitHub
Sonatype
Se
MOCHA
node
ZEPHYR
freshdesk
INTERCOM
FitNesse
SAUCELABS
runnable
ZOHO CRM
BrowserStack
QMETRY
salesforce
servicenow
TestFairy
zendesk
Jasmine
cucumber
Instabug
OMNIDESK
TeamCity
CODESHIP
Jenkins
buddybuild
shippable
circleci
Travis CI
split
HashiCorp
Google Cloud Platform
amazon
Microsoft Azure
ANSIBLE
RELEASE
DEPLOY
heroku
puppet
CHEF
rackspace
OpsGenie
VictorOps
pagerduty
BlueJeans
New Relic.
snyk
bugsnag
Nagios
splunk>
LOGGLY
ZABBIX
RAYGUN
SENTRY
DATADOG
bigpanda
dynatrace
Rollbar
APPDYNAMICS
[:] SourceClear

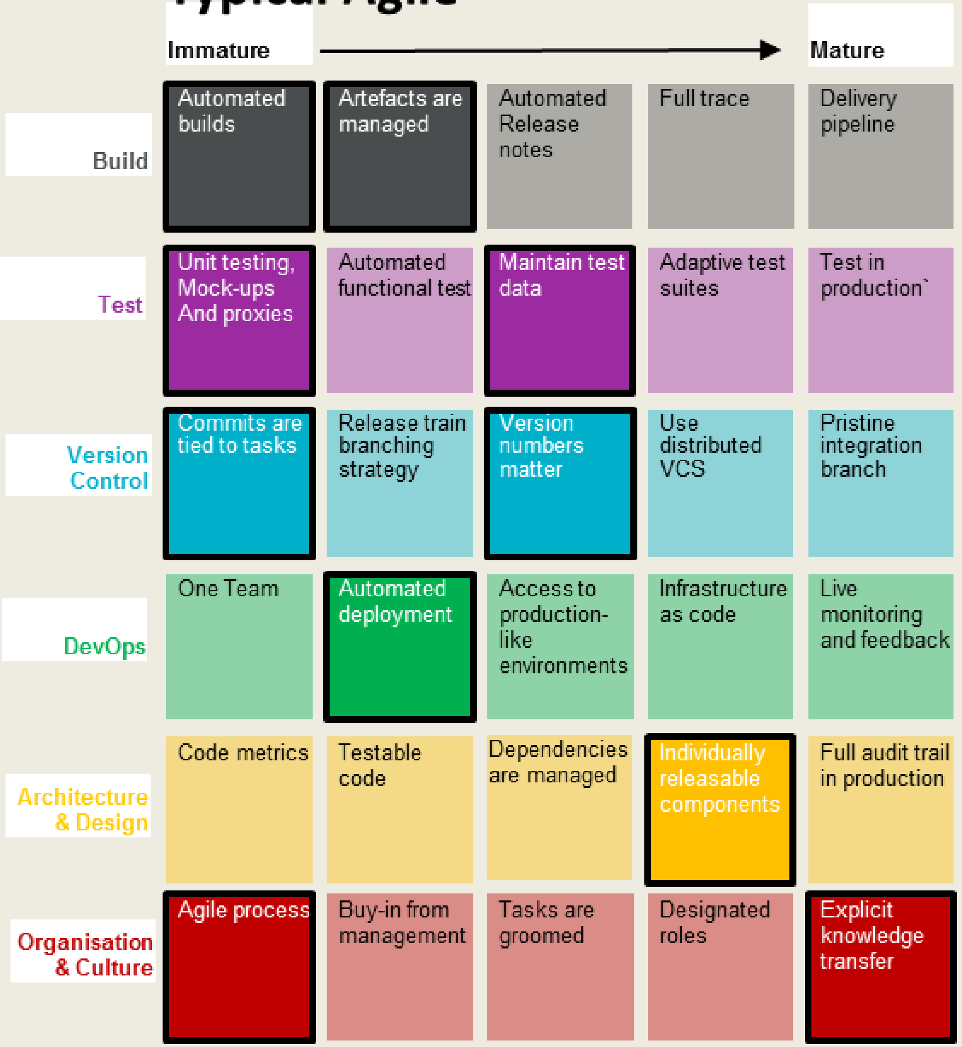
DevOps Maturity & Capabilities…

	Culture	Process	People	Technology
Level 1 Initial Novice	• Uncommunicated Vision • Restricted communication • Sub innovating	• Manual Processes • Inconsistent Project Management • Inconsistent Deliveries • AdHoc Development • AdHoc Testing	• Working Tickets • Teams Organised Around Skillset • Tribal Knowledge	• No Collaboration Tools • No SCM Artefact Management • Manual Monitoring • No Automation • Minimal Testing • Manual Environment Build
Level 2 Managed Beginner	• Clear delivery requirements • Rapid intra-team communication • Innovation via necessity	• Simple Scripts • Project & Requirement Management • Scheduled Deliveries • Scrum Development • Requirements Based Testing • Tactical Reporting	• Working Tickets & Creating Scripts • Teams organised around Delivery • In-Level Technology Skills • Written Knowledge	• Project Planning Tool • Standardised CSM/Artefact • Core Monitoring • Build Automation • Functional Testing • Standardised Environment Build
Level 3 Defined Intermediate	• Articulated business goals • Clear project requirements • Rapid inter-team communication • Innovation by design	• Simple Orchestrations • Integrated Project Management • Automated Deliveries • Agile Development • Integrated Testing • Consolidated Reporting	• Maintaining Scripts & Creating Orchestrations • Teams Organised Around Projects • In-Level Technology Skills • Automation & Documentation	• Knowledge Management • Team/Toolset Integration • Build/Test Automation • Integrated Monitoring • Non-Production Deployment Automation • Automated Environment Build
Level 4 Quantitatively Managed Advanced	• Articulated Business Vision • Clear Project Requirements • Frequent Collaborative Communication • Strategic innovation	• Complex Orchestrations • Quantitative Project Management • Frequent Deliveries • Lean Development • Qualitative Testing • Strategic Reporting	• Maintaining Orchestrations • Teams Organised Around Products/Services • In-Level Technology Skills • Common Knowledge	• Analytics/Intelligence • Self-Healing • Production Deployment Automation • Qualitative Testing
Level 5 Optimised Expert	• Articulated Business Strategy • Clear Business Requirements • Rapid Feedback • Ownership Mindset	• Distributed Orchestrations • Organised Performance Management • Continuous Deliveries & Testing • Predictive Reporting	• Interdisciplinary Teams Organised Around KPI's • Continuous Education • Common Knowledge Transfer	• Tooling as a Product • Toolset Optimisation • Infrastructure As Code

Continuous Integration, Delivery and Deployment Progression

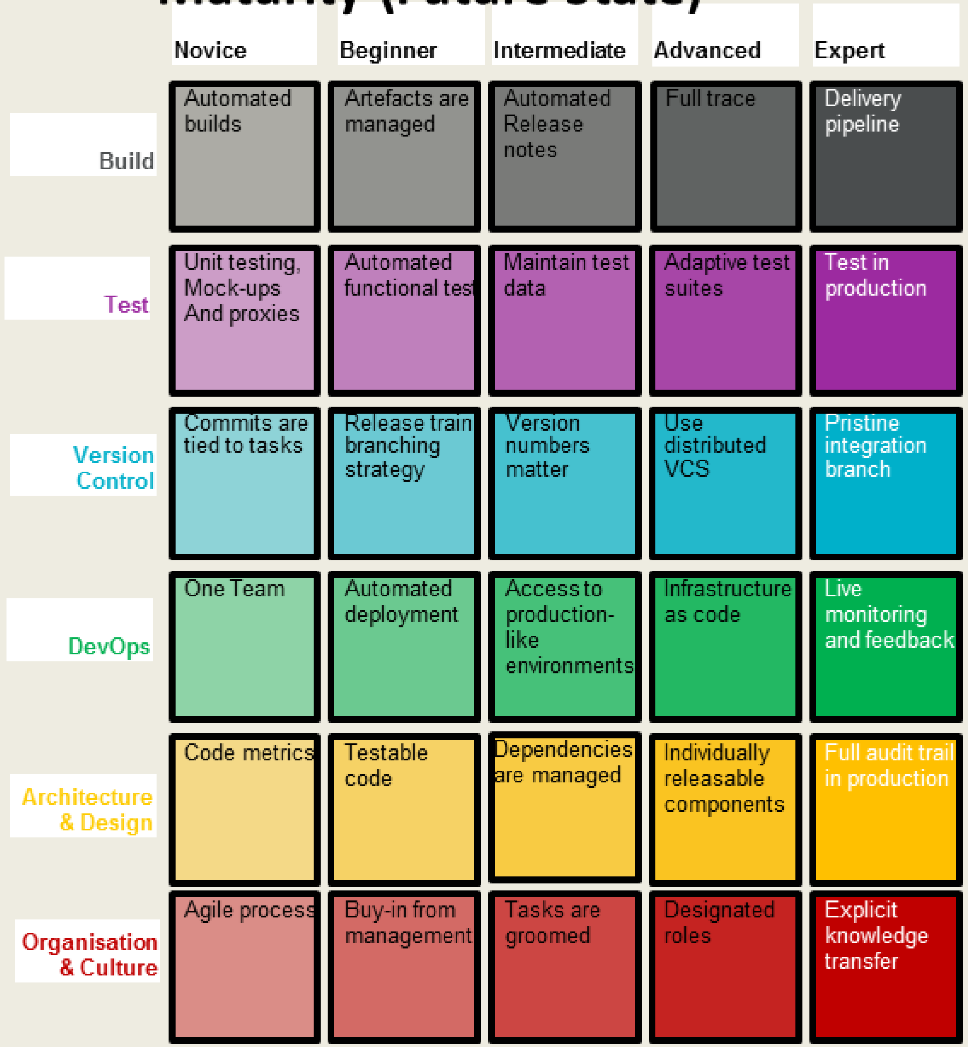
Continuous Never-Ending Improvement to Maturity

Typical Agile



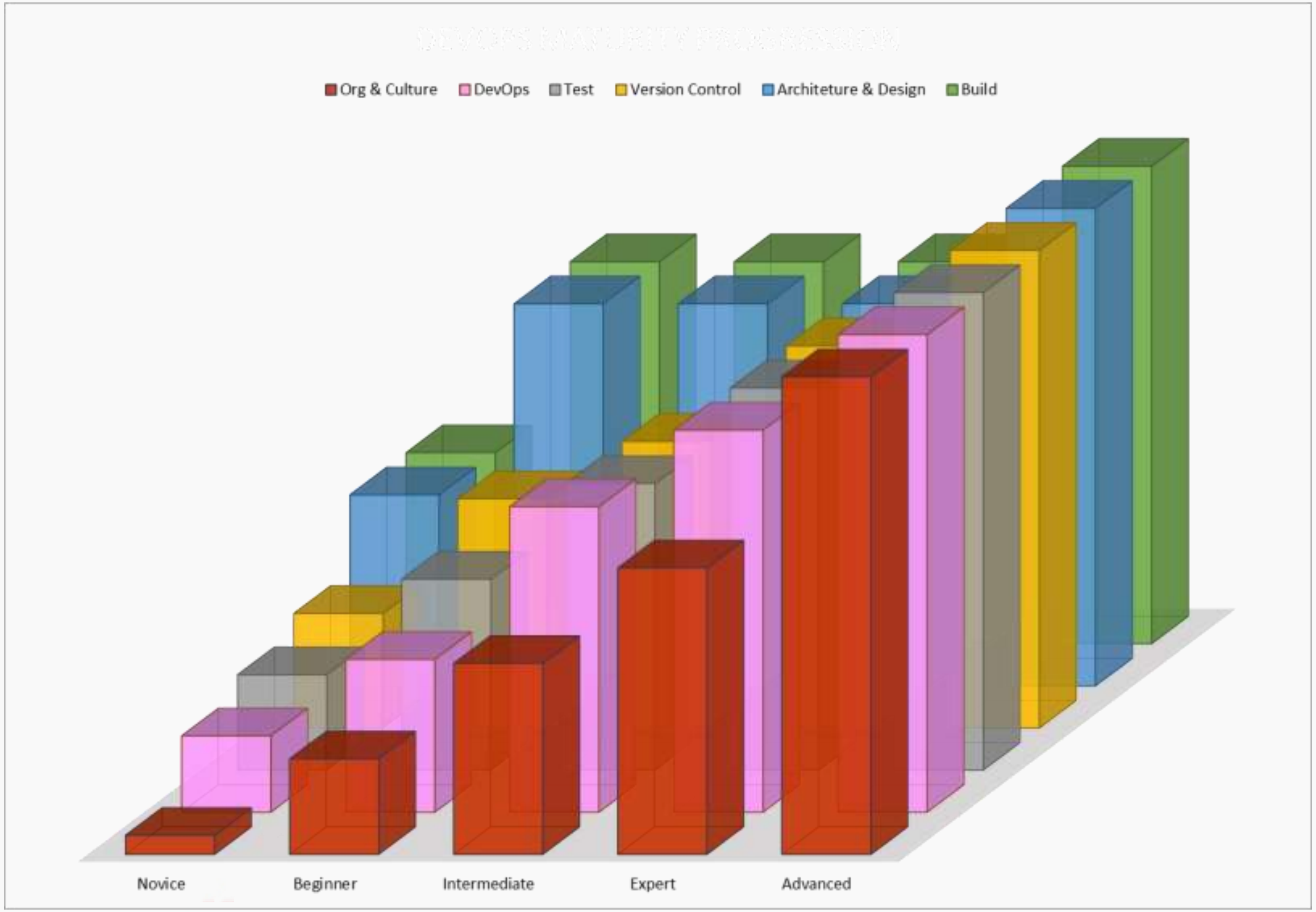
	Immature	→			Mature
Build	Automated builds	Artefacts are managed	Automated Release notes	Full trace	Delivery pipeline
Test	Unit testing, Mock-ups And proxies	Automated functional test	Maintain test data	Adaptive test suites	Test in production`
Version Control	Commits are tied to tasks	Release train branching strategy	Version numbers matter	Use distributed VCS	Pristine integration branch
DevOps	One Team	Automated deployment	Access to production-like environments	Infrastructure as code	Live monitoring and feedback
Architecture & Design	Code metrics	Testable code	Dependencies are managed	Individually releasable components	Full audit trail in production
Organisation & Culture	Agile process	Buy-in from management	Tasks are groomed	Designated roles	Explicit knowledge transfer

Maturity (Future State)



	Novice	Beginner	Intermediate	Advanced	Expert
Build	Automated builds	Artefacts are managed	Automated Release notes	Full trace	Delivery pipeline
Test	Unit testing, Mock-ups And proxies	Automated functional test	Maintain test data	Adaptive test suites	Test in production
Version Control	Commits are tied to tasks	Release train branching strategy	Version numbers matter	Use distributed VCS	Pristine integration branch
DevOps	One Team	Automated deployment	Access to production-like environments	Infrastructure as code	Live monitoring and feedback
Architecture & Design	Code metrics	Testable code	Dependencies are managed	Individually releasable components	Full audit trail in production
Organisation & Culture	Agile process	Buy-in from management	Tasks are groomed	Designated roles	Explicit knowledge transfer

Maturity will be uneven; That’s OK!

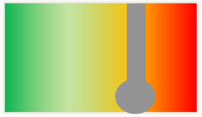
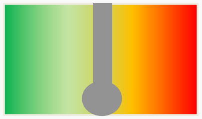
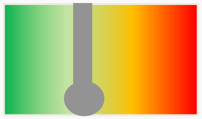
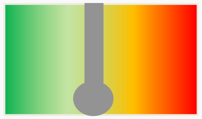
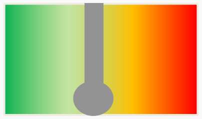
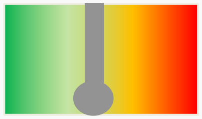
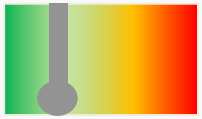
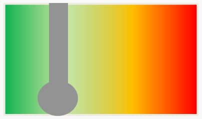


This diagrams strives to show that levels of maturity may vary across disciplines, and so we may have uneven steps.

Because we think often in terms of linear lists, we tend to model maturity progressions in steps, this is not reality.

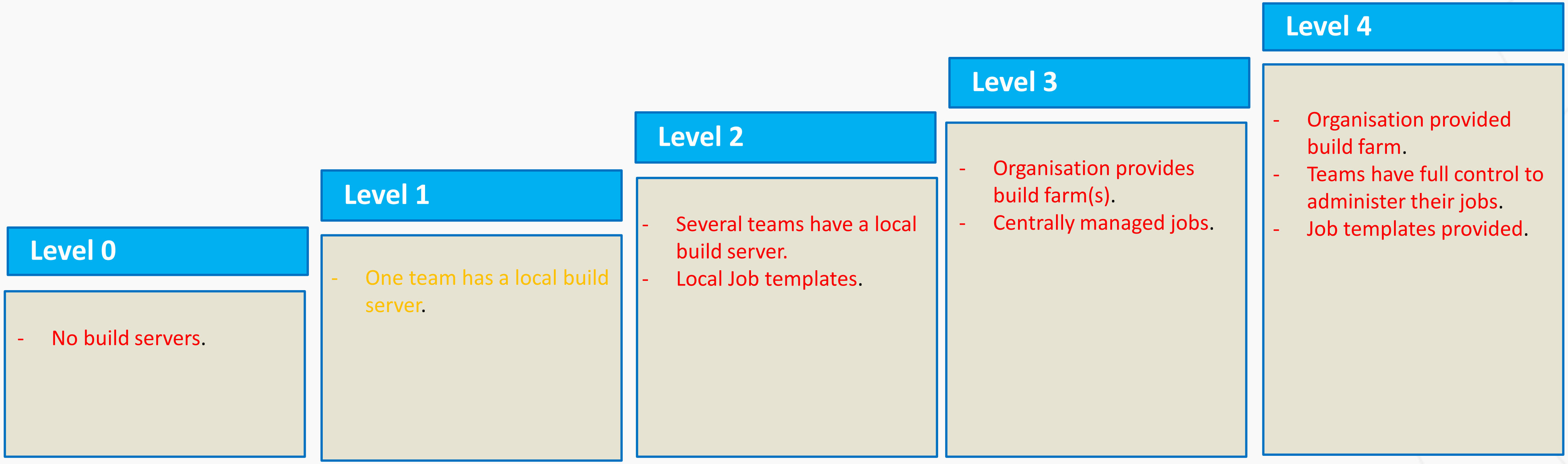
Example:
Intermediate-Level Build may occur before **Intermediate-level Collaboration**, or before **Intermediate traceability**.

Continuous Integration - Core Capabilities

Building Blocks			Complexity	Points of Note
1: Planning	1.1 Requirement Capture 1.2 Portfolio Mgmt 1.3 Collaboration & ChatOps	1.4 Knowledge Management 1.5 Bug/Defect Tracking 1.6 Project Mgmt		1. Requirements traceability is up to and including test cases. 2. Communication channels defined, and Governance forums agreed.
2: Develop & Code	2.1 Integrated Dev Env (IDE) 2.2 Code Generators 2.3 Virtual Desktop (VDI)	2.4 Source Control Mgmt (SCM) 2.5 Code Analysis		1. Incubate process, library/re-use components & automation at onset. 2. Onboarding of resources must be efficient, as well as Information Sharing.
3: Commit & Build	3.1 Build 3.2 Continuous Integration 3.3 Repository Mgmt	3.4 Code Obfuscation 3.5 App Build Distribution 3.6 Unit Testing		1. Build fast, build often. Keep the build green. 2. Integrate early & Test early.
4. Testing	4.1 Functional Testing 4.2 Performance Testing 4.3 UI Testing	4.4 Operational Acceptance Test 4.5 Test Data Mgmt 4.6 Security Testing		1. Test coverage, regression testing if feed from the incubated unit testing. 2. Continually evaluate automated testing (UI, and Test Creation/Test data).
5. Release & Deployment	5.1 App Release Automation (ARA) 5.2 Orchestration	5.3 Configuration Mgmt 5.4 Pipeline Manager 5.5 Cloud Proxy		1. Self service, allow any build to become a candidate release. 2. Environment tracking.
6. Operate	6.1 Monitoring 6.2 Requirements Measurement	6.3 Application Perf Mgmt		1. SLA's are understood. 2. Operational Acceptance Gate (Documentation, Process).
7. Information Management	7.1 Reporting 7.2 Dashboards	7.3 Push Notifications 7.4 BI & Analytics		1. KPI Framework at each stage is a key output. 2. Single notification framework.
8. Common Enablers	8.1 Environment & Lab Mgmt 8.2 Containers 8.3 Service Virtualisation	8.4 Network Virtualisation 8.5 Process Mgmt/Lean Tools		1. Automate as a Mindset, Continuous High Fidelity Feedback. 2. Continuous Improvement is captured via Process Management/Lean tools.

Build Server

Is there a build server up and running, and do teams know about it and why it is there?



Level 0

- No build servers.

Level 1

- One team has a local build server.

Level 2

- Several teams have a local build server.
- Local Job templates.

Level 3

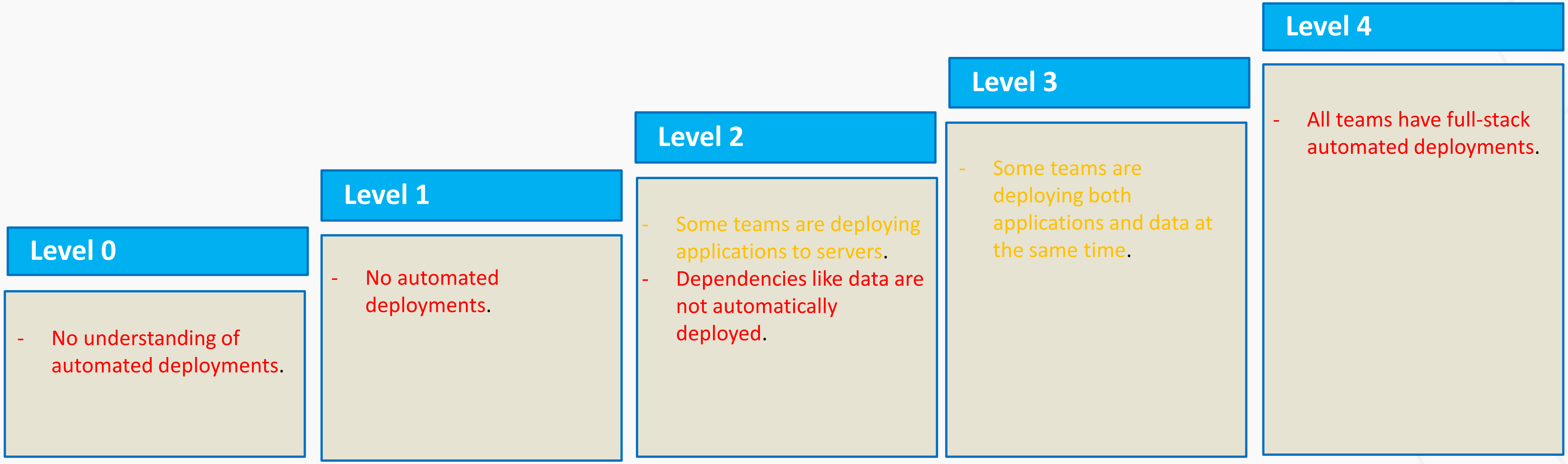
- Organisation provides build farm(s).
- Centrally managed jobs.

Level 4

- Organisation provided build farm.
- Teams have full control to administer their jobs.
- Job templates provided.

Automated Deployments

Are there any automated deployment happening?



Level 0

- No understanding of automated deployments.

Level 1

- No automated deployments.

Level 2

- Some teams are deploying applications to servers.
- Dependencies like data are not automatically deployed.

Level 3

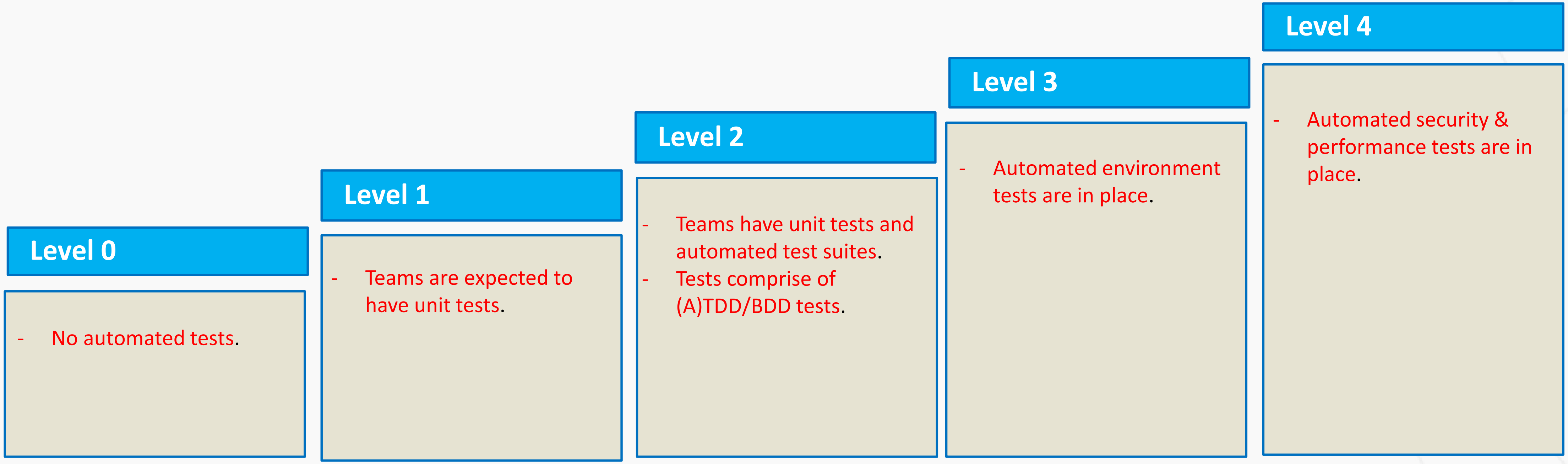
- Some teams are deploying both applications and data at the same time.

Level 4

- All teams have full-stack automated deployments.

Automated Deployments

Is automated testing understood and being used?



Level 0

- No automated tests.

Level 1

- Teams are expected to have unit tests.

Level 2

- Teams have unit tests and automated test suites.
- Tests comprise of (A)TDD/BDD tests.

Level 3

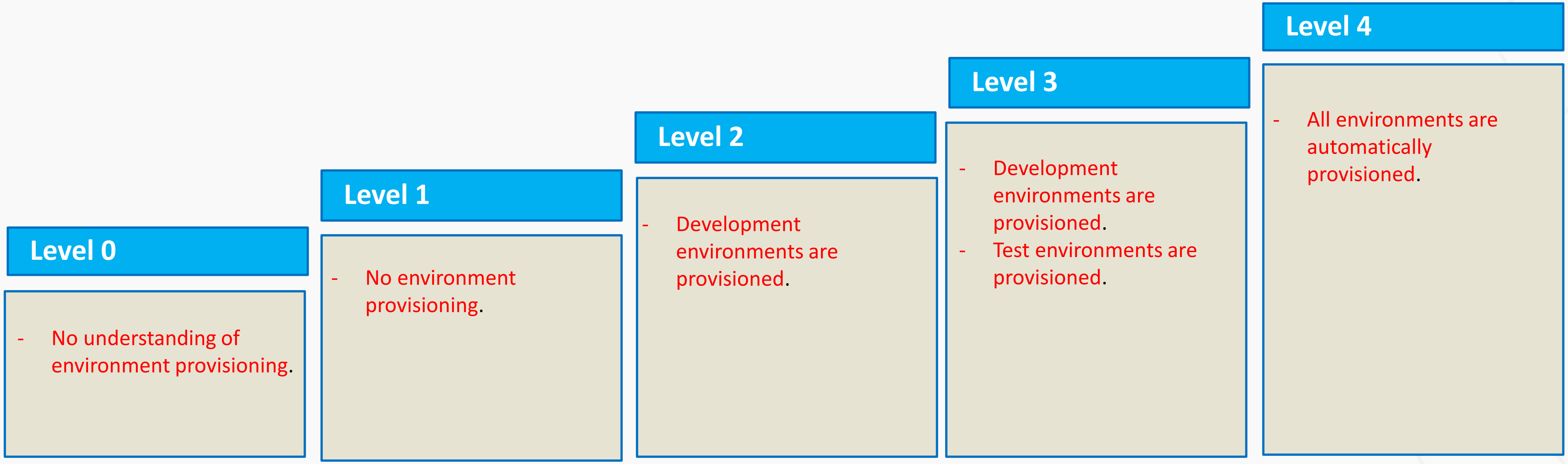
- Automated environment tests are in place.

Level 4

- Automated security & performance tests are in place.

Environment Provisioning

Is automated testing understood and being used?



Level 0

- No understanding of environment provisioning.

Level 1

- No environment provisioning.

Level 2

- Development environments are provisioned.

Level 3

- Development environments are provisioned.
- Test environments are provisioned.

Level 4

- All environments are automatically provisioned.

DevOps Integrations

What integrations to other systems exist?

Level 0

- There is only a Continuous Integration (CI) Server.

Level 1

- CI server is integrated with an artefact management system.

Level 2

- Code metrics tools are integrated.

Level 3

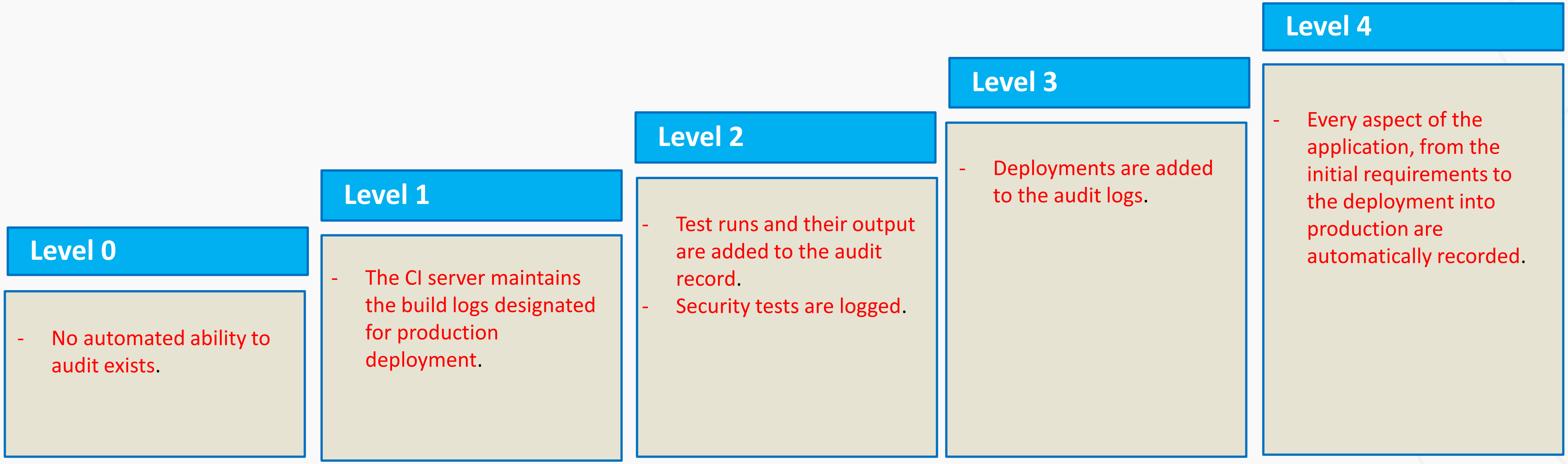
- Requirements systems are integrated.

Level 4

- Integrations are in place for all systems used in build, test and deployment.
- Monitoring software is integrated.
- Log management software is integrated.

Audit-ability

Is the overall pipeline audit-controlled?



Level 0

- No automated ability to audit exists.

Level 1

- The CI server maintains the build logs designated for production deployment.

Level 2

- Test runs and their output are added to the audit record.
- Security tests are logged.

Level 3

- Deployments are added to the audit logs.

Level 4

- Every aspect of the application, from the initial requirements to the deployment into production are automatically recorded.

Culture

Do groups in the end-to-end process operate from the perspective of shared accountability for the delivery of business results?

Level 0

- No obvious culture of collaboration.
- The proverbial silo.

Level 1

- Separate teams exist to build software, test it, deploy it and manage it.
- Little to no interaction across teams.
- Potential silo attitudes.

Level 2

- Delivery teams are responsible for their own quality and builds.
- Handed to a ‘production’ team for deployments and infrastructure / system configuration.
- Disjointed commitment.

Level 3

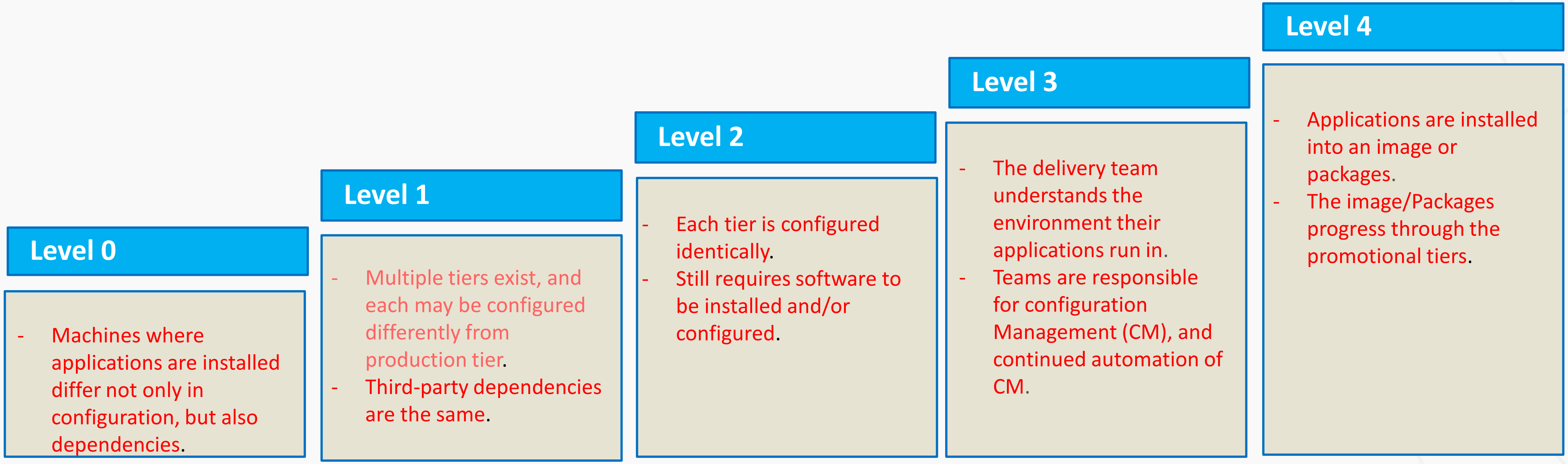
- The delivery team understands the environments their application will run in and are responsible for every thing but deployments.

Level 4

- All individuals in the IT organisation understand all facets of delivery.
- They do their part to make the organisation successful.
- Rapid feedback is automatically available to all to see the health of the systems and practices.

Server Integrity

Do the servers from Development to Production have the exact same configuration and version of OS and other supporting software?



Level 0

- Machines where applications are installed differ not only in configuration, but also dependencies.

Level 1

- Multiple tiers exist, and each may be configured differently from production tier.
- Third-party dependencies are the same.

Level 2

- Each tier is configured identically.
- Still requires software to be installed and/or configured.

Level 3

- The delivery team understands the environment their applications run in.
- Teams are responsible for configuration Management (CM), and continued automation of CM.

Level 4

- Applications are installed into an image or packages.
- The image/Packages progress through the promotional tiers.

Monitoring

Is monitoring in place that provides timely and accurate notification of issues?

testhouse

Level 0

- No monitoring in place.

Level 1

- Production servers have the OS and processes monitored.
- Delivery teams do not have access.

Level 2

- Pre-production environments are monitored, with teams being notified of negative events.

Level 3

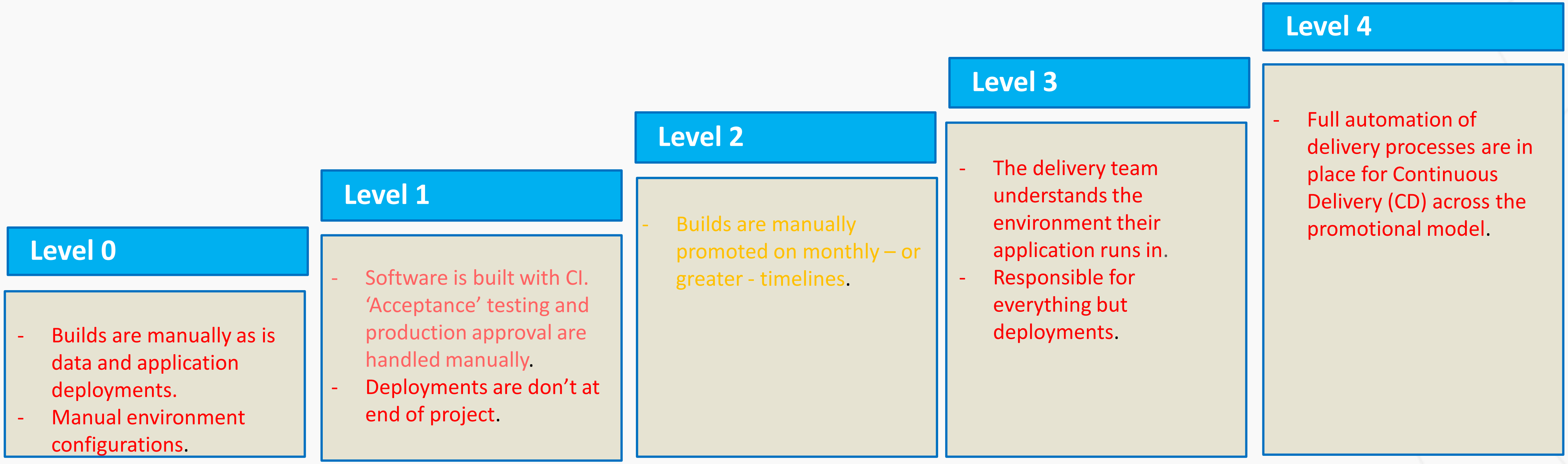
- The delivery team understands the environment that their application runs in.
- Responsible for everything but deployments.

Level 4

- Every system and process involved with delivery and execution are monitored.
- Dashboard exists and are available to all.

Continuous Delivery

Does the organisation have the tools and process in place to enable the regular in frequent delivery of software into production via a promotional model?



Level 0

- Builds are manually as is data and application deployments.
- Manual environment configurations.

Level 1

- Software is built with CI. ‘Acceptance’ testing and production approval are handled manually.
- Deployments are don’t at end of project.

Level 2

- Builds are manually promoted on monthly – or greater - timelines.

Level 3

- The delivery team understands the environment their application runs in.
- Responsible for everything but deployments.

Level 4

- Full automation of delivery processes are in place for Continuous Delivery (CD) across the promotional model.

Managed Artefacts

Are artefacts being managed by an artefact management system?

Level 0

- No understanding of managed artefacts.

Level 1

- Dependent libraries and build-artefacts are being stored in SCM.

Level 2

- Libraries and built artefacts are being managed.
- Configuration Management is not occurring.
- Scripts and test data are not managed.

Level 3

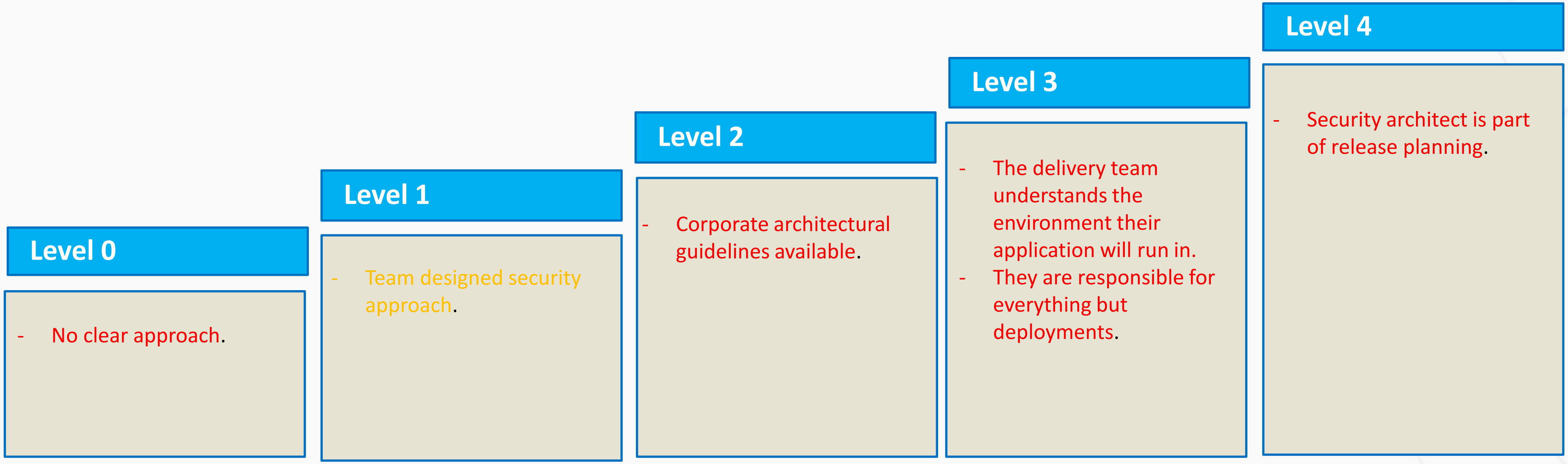
- The delivery team understands the environment their application will run in.
- They are responsible for everything but deployments.

Level 4

- All deployments are done using managed artefacts.

Security

Is the security of software considered and how is it handled?



Level 0

- No clear approach.

Level 1

- Team designed security approach.

Level 2

- Corporate architectural guidelines available.

Level 3

- The delivery team understands the environment their application will run in.
- They are responsible for everything but deployments.

Level 4

- Security architect is part of release planning.

Within the enterprise, where do we begin?...

Getting started – Where do we start?

A common recommendation is to improve day-today stability of trunk:

1. Setup Continuous Integration with some automated tests and start learning to continually keep builds green.

2. If already doing this at component-level using unit-tests, then move to a larger application builds in an operational like environment with automated system test.

3. Culture change will occur naturally because different development groups are continually getting all the code working together and delivered (to the Customer/Market) as opposed to just localised intermediary steps.

4. Stress the test automation to ensure it is stable and maintainable.

4 Steps to successful starting DevOps

testhouse

- Embarking on an organizational transformation with DevOps can be a massive undertaking.
  - In fact, it can be downright overwhelming when you consider the scope and size of the changes it requires.
- DevOps represents a **major cultural change**, so keep your expectations in check.
- Don't expect the organization to change overnight, and don't expect the entire organization to change at the same time.
- Consider following these four steps for a smooth transition.

1

Start at the right place at the right time

1. Start at the right place at the right time

- Decide where to plant the seeds of change and grow.

- Where can we **harvest quick wins** and learn what works?

- Start with the business:
  - Does it demand speed and velocity from IT?
  - Is management desperate to go faster?
  - Are they open to change?

- Is IT receptive or resistant to change?

- Try to choose areas where technology is more cloud and web oriented.
  - It's not that DevOps is only suitable for web and cloud; it's that newer technology with lower legacy debt will make it easier for your first time out the gate.

- Look for areas where the teams will be empowered.

2

Lather (observe and orient)

2. Lather (observe and orient)

- **Observe and measure** how things work today.
  - How long does it take for a new requirement to get to production?
  - Employ rapid scientific methods to assess success/failure.

- Find the areas with the greatest pain, and find the biggest bottlenecks.

- Get some data, create a baseline, and figure out how to continually update it.

- A word of caution:
  - **Don't get stuck measuring for the sake of measurement**. Do just enough, and then get moving.

3

Rinse (decide and act)

3. Rinse (decide and act)

- Do something different.
- The team knows where the problems are, so fix those first.
- If the problem is in coding and builds, work on source control and continuous integration.
- If testing and QA are the biggest issues, automate and implement continuous testing to streamline those processes.
- If the problem area is the deployment and delivery of apps and infrastructure, continuous delivery is the place to start.
- As you start making changes, don't forget to pay attention to the data.
- If the data gets better, then keep going. If things get worse, don't hesitate to go back to the drawing board.

4 Repeat

4. Repeat

- **Don't rush things, and Don't stop.** The improvement loop for DevOps has no end: It's really all about continuous improvement.

- Give the process time to sink in with the team. It may even be a month or more before you can go back to the beginning.

- Teams may need to iterate a process/tools combination several times before they'll feel confident & witness improvement.

- When the team(s) is/are finally ready, start again with the data and search for new bottlenecks and pain points.
  - This is the next opportunity to improve, accelerate, and streamline delivery.

- **Participation** in the DevOps transformation will vary with roles.

- IT **leaders need to empower** the team so they can own the change and the results.
  - The **partnership between IT leadership** and the **team** must be built on **mutual trust** based on a **shared goal** of delivering more value to the customer.

Thought leadership...

Key Cultural Changes

- Getting developers to **own** ensuring **every check-in to trunk** is stable in a production-like environment.

- Getting **Development** and **Operations** teams using **common tools** and environments to align them on a common objective.

- Getting the entire **organisation** to **agree** that the **definition of done** at the release branch i.e. the feature is signed off, defect-free, and the test automation is ready in terms of test coverage and passing rates.
  - KPI/Metrics exist.
  - Process is improved & documented.

- Getting the organisation to **embrace** the unique characteristics of your **software & build practice** then design a **planning process** that takes advantage of it's **flexibility**.

The Improvement Karta - aka Kaizen

1 Repetition is the mother all of learning!

2 Repetition creates habit!

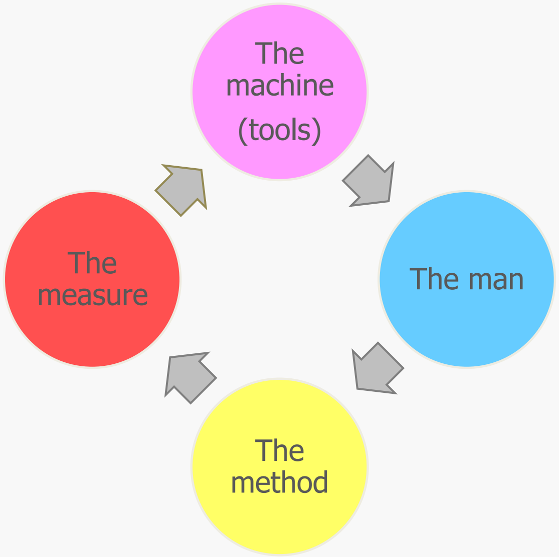
3 Habits are what enable mastery!

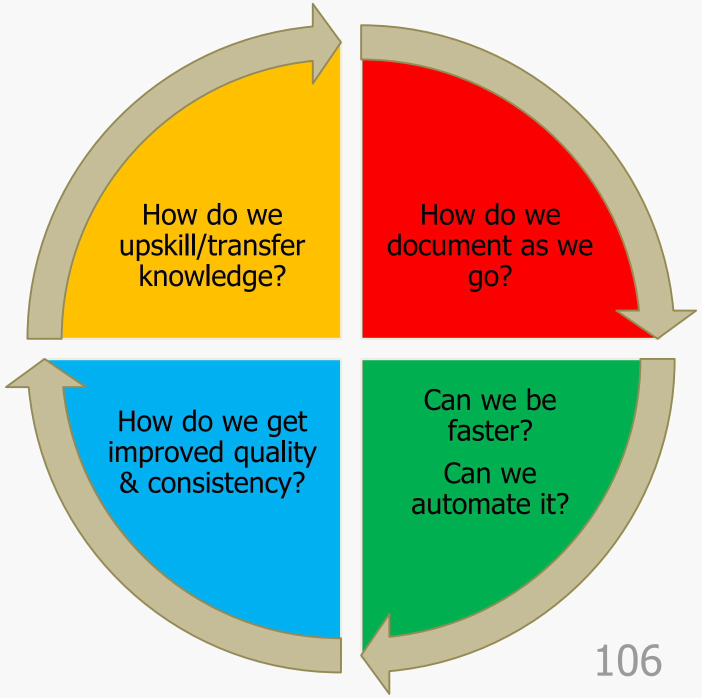
Every two weeks, make one small process improvement

Constant

Never-Ending

Improvement




How do we document as we go?
Can we be faster? Can we automate it?
How do we get improved quality & consistency?
How do we upskill/transfer knowledge?

Are you up for the Challenge?

- **Who do you need** to join you in leading the transformation?
- **What are your plans** for taking the first step?
- **What** are your **business objectives**?
- **What organisational barriers** do you think will need to be addressed?
- What do you think your **organisation can commit to completing** in the first iteration of your enterprise-level continuous improvement process?

Traceability, metrics and KPIs...

Traceability Matrix & Systems

- Traceability" could be factually described as "**the use of tracking and tracing systems and processes to match the incoming product requirements to outgoing product attributes**."

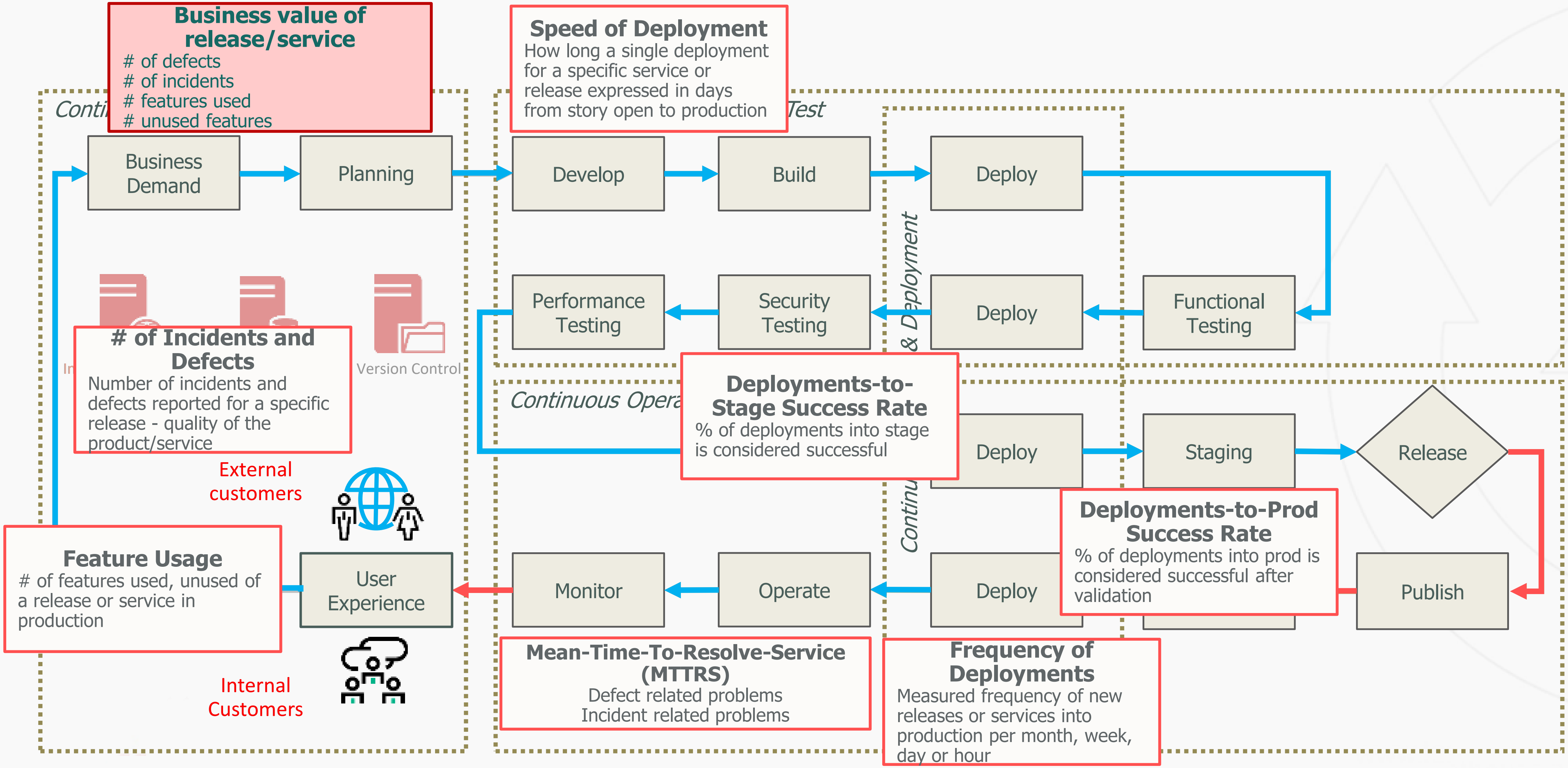
- **Forward traceability**
  - Ensure at any point that everything you require is in the plan somewhere, or, better yet, in the actual software.

- **Backward traceability**:
  - Ensure that everything in the software was developed for some identifiable reason, and it wasn't just developers running amok.

- **Keeping things tidy when requirements change, both during the project and during ongoing maintenance afterwards**
  - If requirements change, you would like to know what the impact of the change will be?
  - Which pieces of the system will be impacted?
  - How many of them are there?

Measuring success: velocity, quality and value

testhouse



Business value of release/service
# of defects
# of incidents
# features used
# unused features

Speed of Deployment
How long a single deployment for a specific service or release expressed in days from story open to production

# of Incidents and Defects
Number of incidents and defects reported for a specific release - quality of the product/service

Deployments-to-Stage Success Rate
% of deployments into stage is considered successful

Deployments-to-Prod Success Rate
% of deployments into prod is considered successful after validation

Feature Usage
# of features used, unused of a release or service in production

Mean-Time-To-Resolve-Service (MTTRS)
Defect related problems
Incident related problems

Frequency of Deployments
Measured frequency of new releases or services into production per month, week, day or hour

Technology Metrics/KPIs

testhouse

- Deployment Frequency
- Lead time for changes
- % Change error rates
- % Failure rates
- Lines of code?
- % Availability
- Recovery time
- Job satisfaction
- Defect %
- # Security tests
- # Broken builds

- Mean-time-to-resolve
- % Test pass rate
- Stories - branch ready
- Stories - not ready for branching
- # Environments
- % Code coverage
- # Unit tests
- # System tests
- % System tests automated
- % Continuous green build

Recommendations for the Enterprise…

Recommendations:
Policy, Planning, and Management

- **Establish a viable governance framework**
  - CI is not only a technical implementation, it is also an organizational and cultural implementation.
- **Establish a "CI Implementation" working group**
  - Management should set up a software development technical leadership working group to lead CI adoption and other software quality and process improvement initiatives through training and technical support.
- **Version control systems must be used across the pipeline**
  - An improperly used source control system can easily derail efforts to implement CI improvement in upstream stages.
- **Enforce proper release management**
  - The CI Implementation working group should take steps to clearly define, document, and enforce policies and guidelines for proper software release practices.

Recommendations: Technology Implementation

- Adopt CI practices incrementally
  - Each team should honestly assess where their maturity level it is at a given point in time, then put in place the changes needed to move to the next level.
  - Continually reviewing the maturity level, will support maturity at scale.
- Implement CI early in new projects
  - Teams should focus their CI improvement efforts on new projects, maintenance releases, defect fixes, and new development and implement new CI practices early.
- Provide fast machines for the CI build and test servers
  - High-performance servers will result in faster builds, removing one of the most important roadblocks to CI improvement.

Recommendations: Technology Implementation (cont…)

- Start with the most willing & knowledgeable teams
  - Initial attempts to improve CI practices should start with the teams that already have some of the key practices in place.
- Write scripts to automate manual processes
  - Teams should decide which manual processes are causing the most problems and automate these first.
  - The benefits of doing this will motivate them to tackle the next problem area, and so on.
- Get serious about automated testing
  - Detecting defects early depends on having all the necessary testing resources in place.
  - No implementation of CI should be attempted without a commitment by all stakeholders to upgrade the department's testing practices and infrastructure.

Recommendations: Technology Implementation (cont…

- Continually adopt the most relevant best practices in CI
  - Development teams should attempt to apply the practices appropriate for the maturity level they are working toward in their individual CI adoption objectives.
- Train and mentor developers and testers together
  - CI improvements create additional demands on the workforce, so the need for additional training, and continuous mentoring and support, is critical, and must be factored into velocity reporting i.e. delivery speed.
- Consider using external consultants and trainers
  - There are outside consulting firms that specialize in providing services to “jump start” an organization’s CI improvements, and it might be both beneficial and cost effective to consider engaging one of these firms.

It is all fine and dandy, but of no use long-term when no governance or LEAN software improvement is applied.

General thoughts

- Process and definition even at varying layers of detail will have no long term impact. The focus has to be in changing people's behaviour.
- Any strategy must encompass the ability to encourage and foster behaviour patterns of the people who participate within.
- Culture is made up of the attitude of choice, thus attitude will define the altitude of success.
- Focus on the culture change objectives, and use process as a conduit then it will become an aqueduct to transport success.
- The ultimate destination is a river of exponential success due to the tributaries of continuous change and improvement.

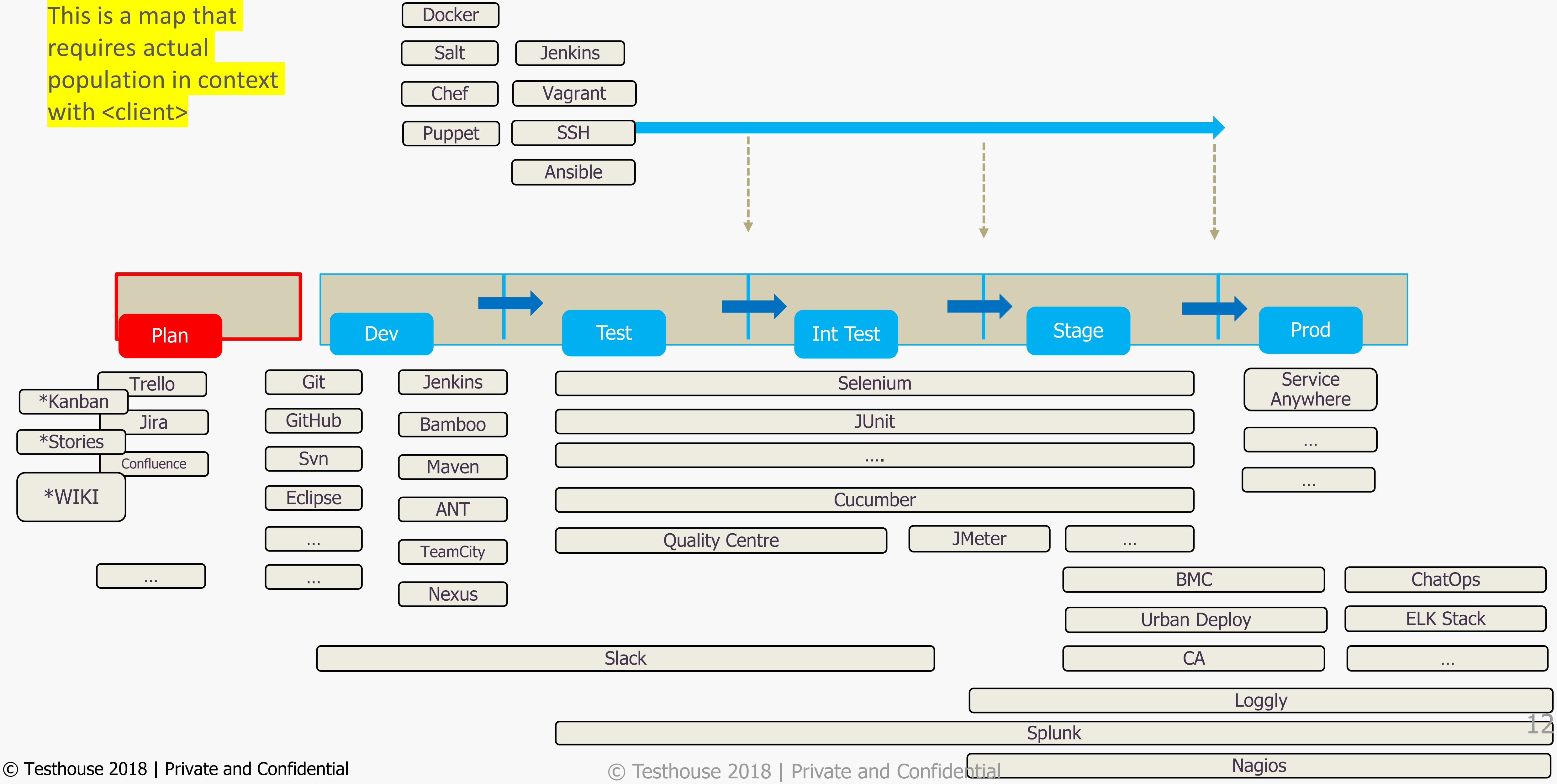
Tool Choice, and Potentiality…

Continuously strive to Unify & Integrate tools

- Single Portfolio Management
- Single Tool for Requirements
- Single Tool for User Stories
- Single Wiki Style Product & Knowledge Linking
  - Single Rich Document Store (Governance, Architecture, Framework, Process)
- Single SCM (Git)
- Single Kanban Tool
- Single Bug Tracking
- Single Test Script Management
- Single Defect Management
- Single Feedback Tracker / Collaboration Tool
- Single Release Management
- Multiple Alert Channels, Multiple media

Tools to make it work (Examples of usual **s**uspects)

This is a map that requires actual population in context with <client>



Docker

Salt

Chef

Puppet

Jenkins

Vagrant

SSH

Ansible

Plan

Dev

Test

Int Test

Stage

Prod

Trello

*Kanban

Jira

*Stories

Confluence

*WIKI

…

Git

GitHub

Svn

Eclipse

…

…

Jenkins

Bamboo

Maven

ANT

TeamCity

Nexus

Selenium

JUnit

….

Cucumber

Quality Centre

JMeter

…

Service Anywhere

…

…

BMC

Urban Deploy

CA

ChatOps

ELK Stack

…

Slack

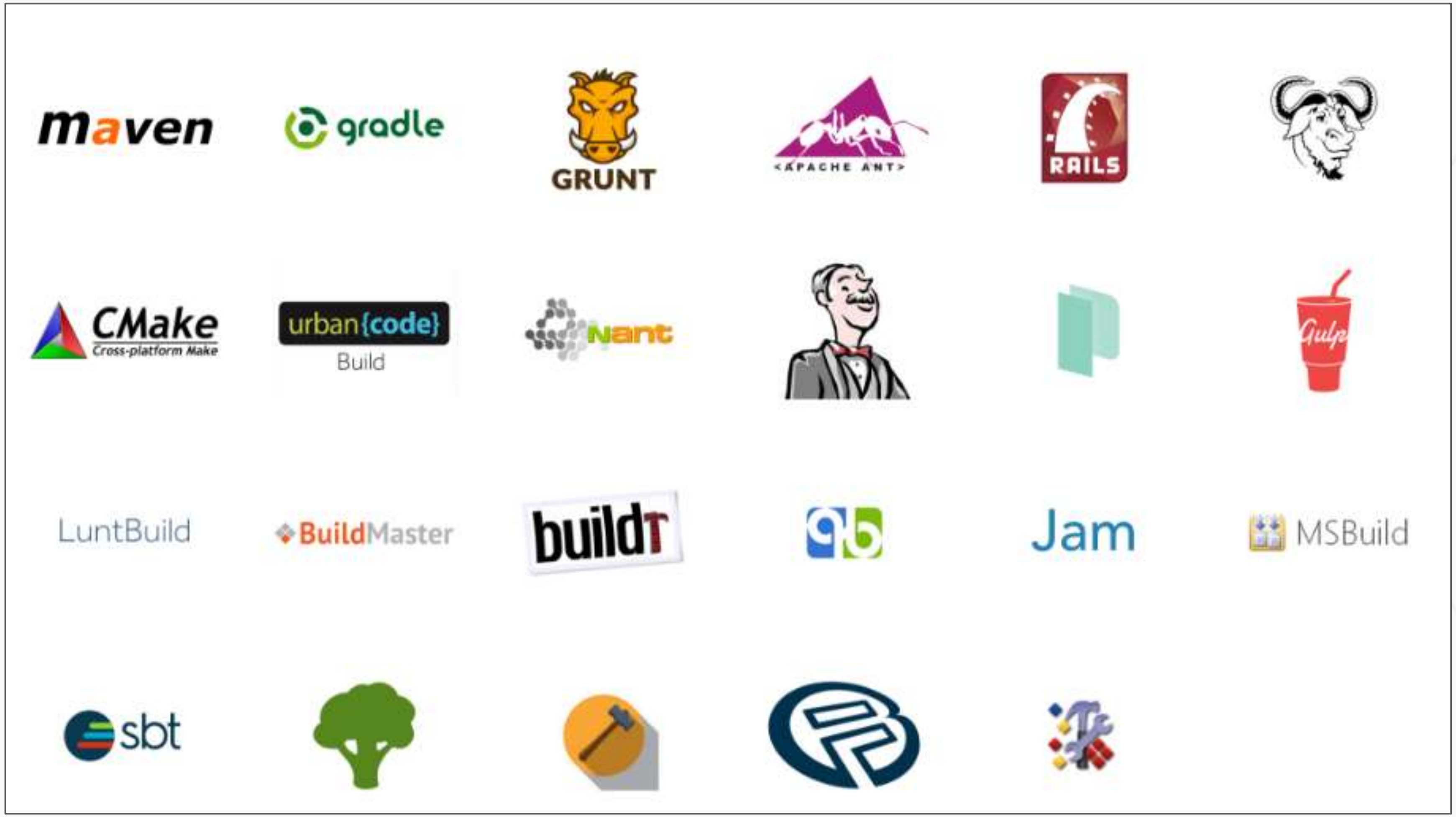
Loggly

Splunk

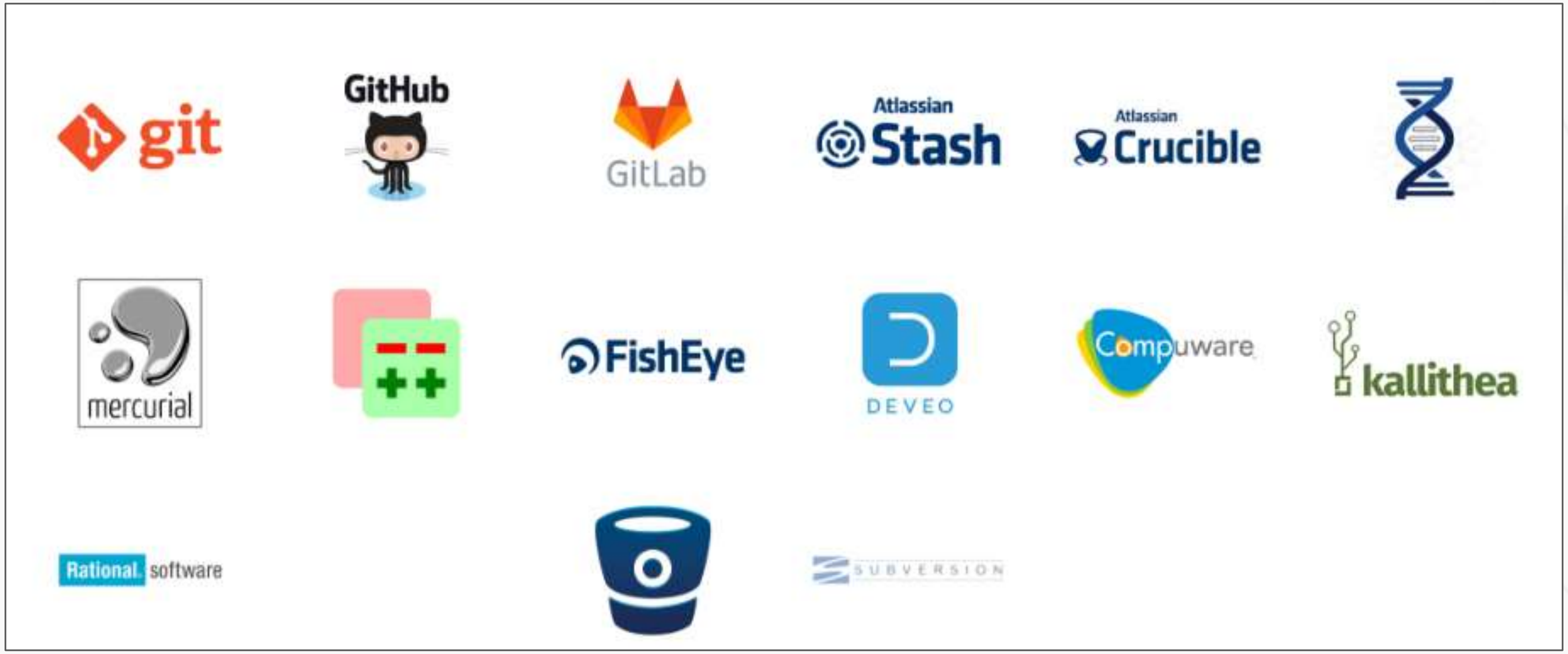
Nagios

© Testhouse 2018 | Private and Confidential

Build Tools


maven
gradle
GRUNT
<APACHE ANT>
RAILS
CMake
Cross-platform Make
urban{code}
Build
Nant
Gulp
LuntBuild
BuildMaster
buildr
Jam
MSBuild
sbt

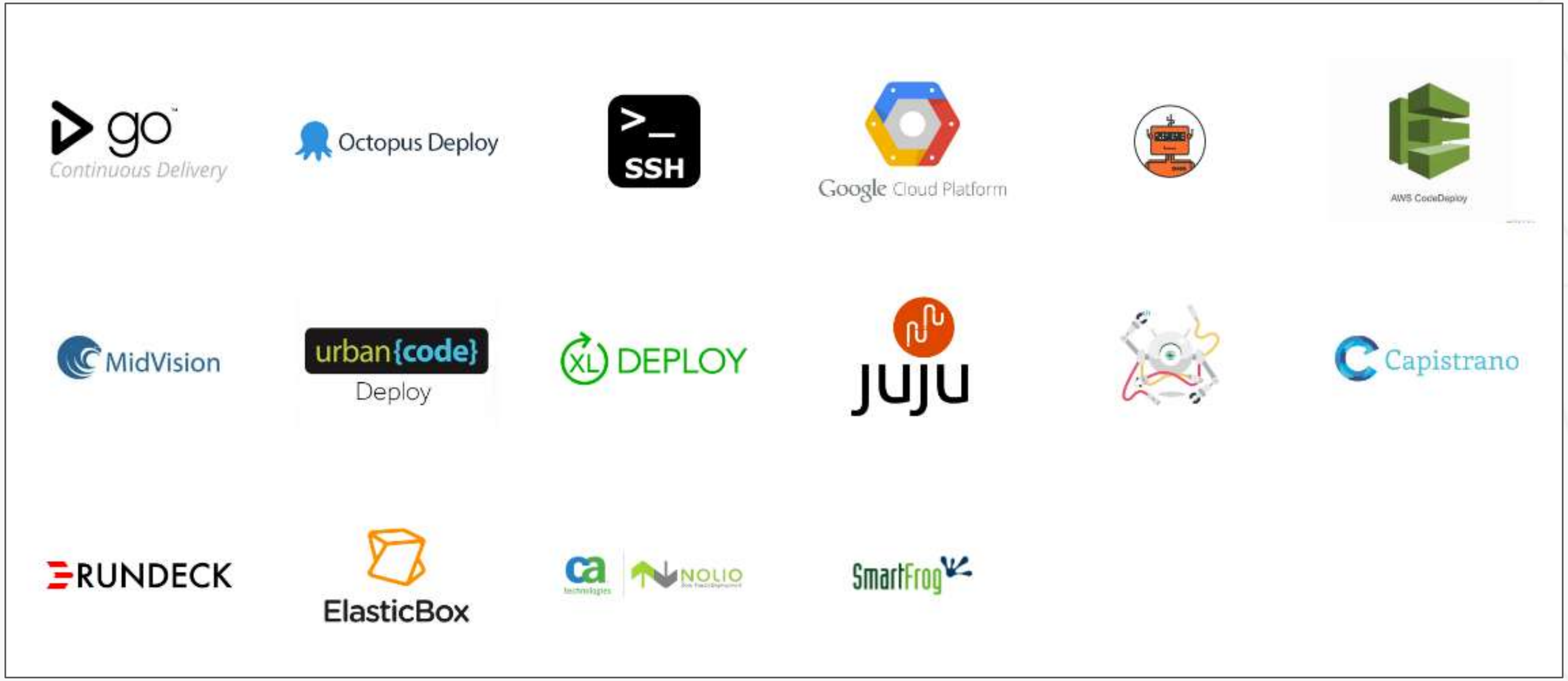
Source Control Management



Continuous Integration

Jenkins
Solano Labs
Visual Studio
TeamCity
shippable
snap
cruisecontrol.
continuum
Apache Gump
circleci
Bamboo

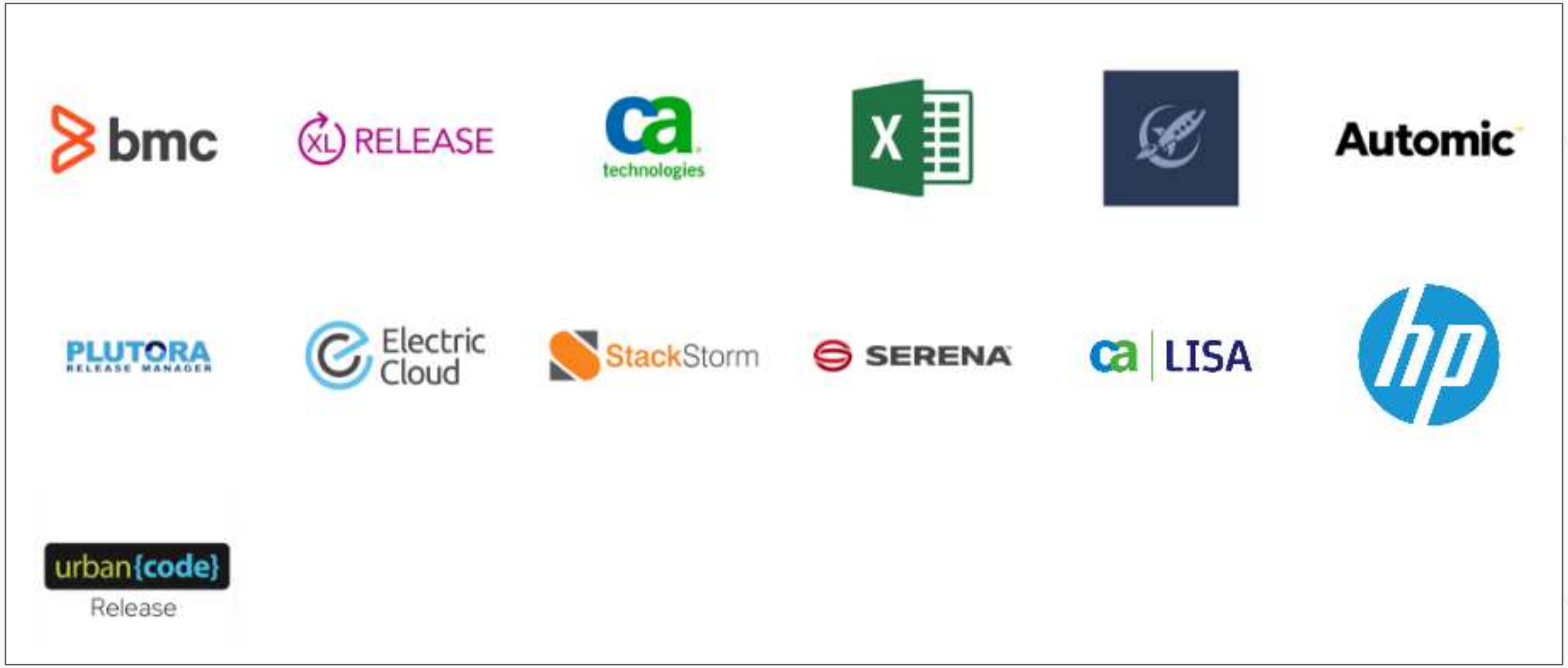
Deployment Tools



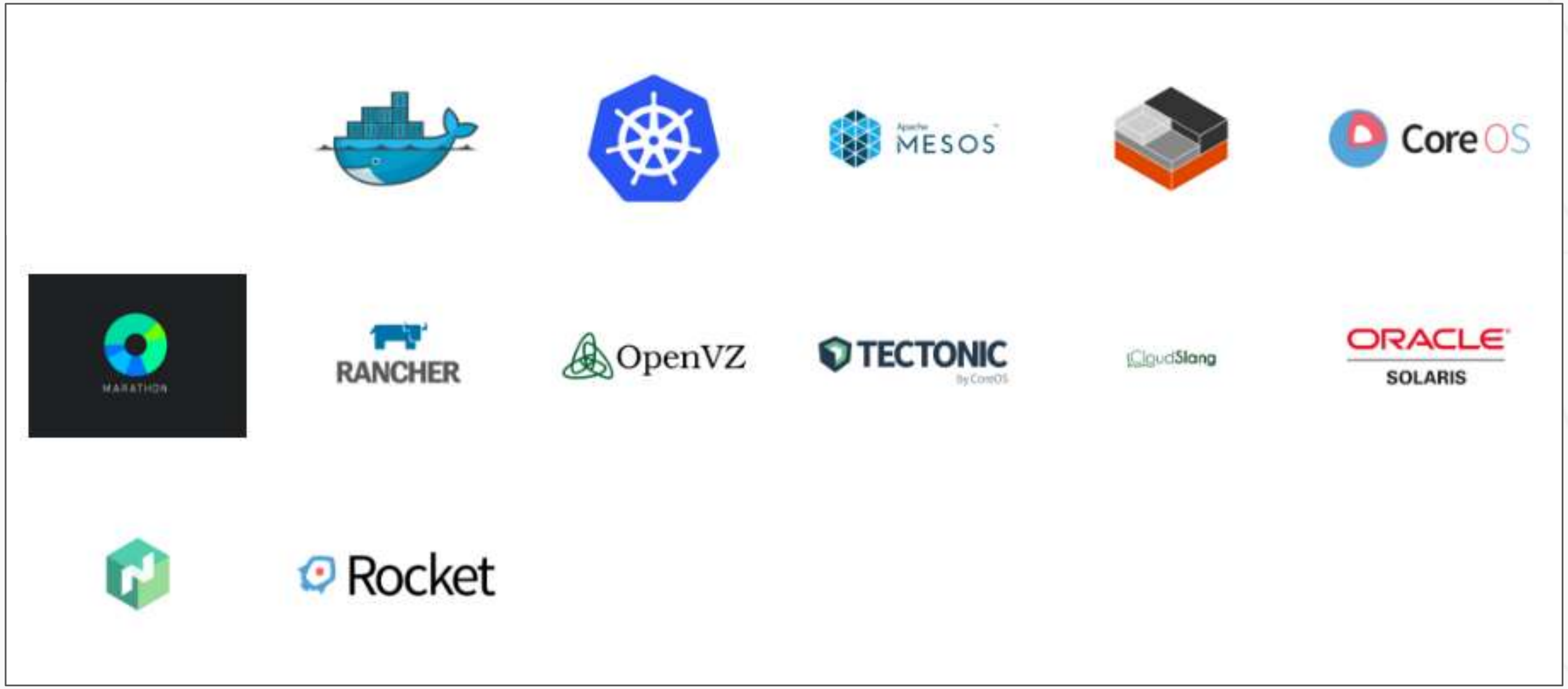
Repository Management

Nexus
JFrog Artifactory
npm
nuget
archiva
pulp

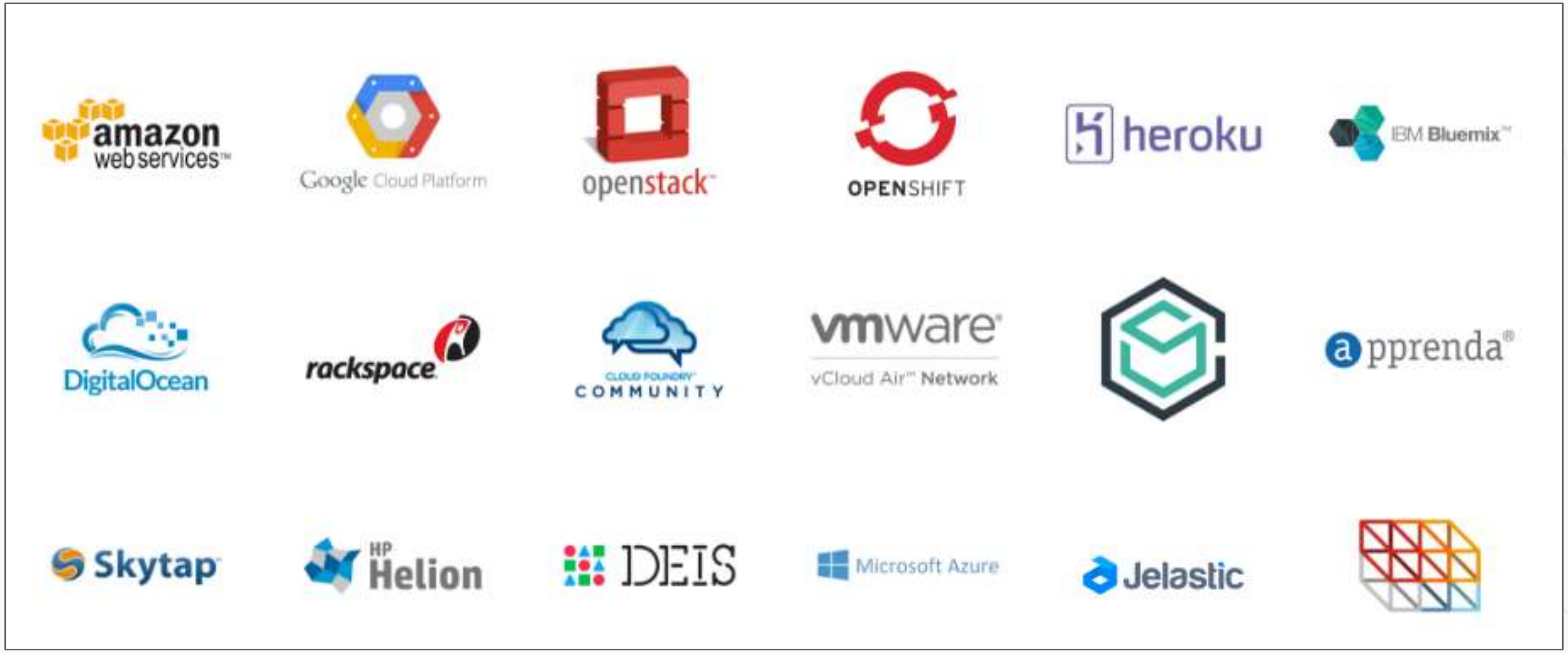
Release Management



Containerisation



Cloud, IAAS, PAAS



Thank You!

www.testhouse.net

United Kingdom

8 Lanark Square, London, E14 9RE
United Kingdom

+44 20 8555 5577

Middle East

1403-27, City Tower 2, Near Durrah Tower
Sheikh Zayed Road, Dubai, UAE

+971 50 354 9541

United States

10100 Santa Monica Boulevard
#300 Los Angeles, CA, 90067, USA

+1 630 917 1053

India

II Floor, Nila Building, Technopark Campus
Thiruvananthapuram, Kerala 695581

+91 471 270 0117