innovate achieve lead

BITS Pilani
Pilani Campus

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

By Prof A R Rahman
CSIS Group WILP

BITS Pilani
Pilani Campus

Course Name: Introduction to DevOps
Course Code : CSI ZG514/SE ZG514
CS -2

By Prof A R Rahman
CSIS Group WILP

Coverage

- Problems of Delivering Software
- Principles of Software Delivery
- Need for DevOps
- Evolution of DevOps- Define the stages of a DevOps evolution
- DevOps practices in organizations
- The Continuous DevOps Life Cycle Process (Continuous Integration, Continuous Inspection, Continuous Deployment, Continuous Delivery, Continuous Monitoring)
- DevOps Culture
- Case Study- (IBM/Facebook/NetFlix)

Problems of Delivering Software

- Late Delivery of Projects: The issue of late delivery due to communication gaps in the team and pending approvals from customers.

- It includes editable features and large content boxes to add information on topics like development, communicate, consumers, and process.

- Problems in Agile Project Methodologies: The problems faced in agile project methodologies while using traditional software development methods like waterfall model, V model, and Rational unified process.

- It includes editable features and large content boxes to add information on topics like budget deficit, unproductivity, agile project methodologies, process, and software development.

Problems of Delivering Software

- Software Delivery Performance Metrics: Aspects of software performance assessment such as deployment frequency, lead time for changes, time to restore services, and change failure rate.

- It provides visual cues and insights and can be used to share invaluable insights on rate, service, deployment, and software

- Challenges in Software Delivery Pipeline: This template addresses common issues in the software delivery pipeline, such as large Pull Requests and the need for structured work.

- It provides guidance on how to use tools like Athenian to spot and address these issues.

Problems of Delivering Software

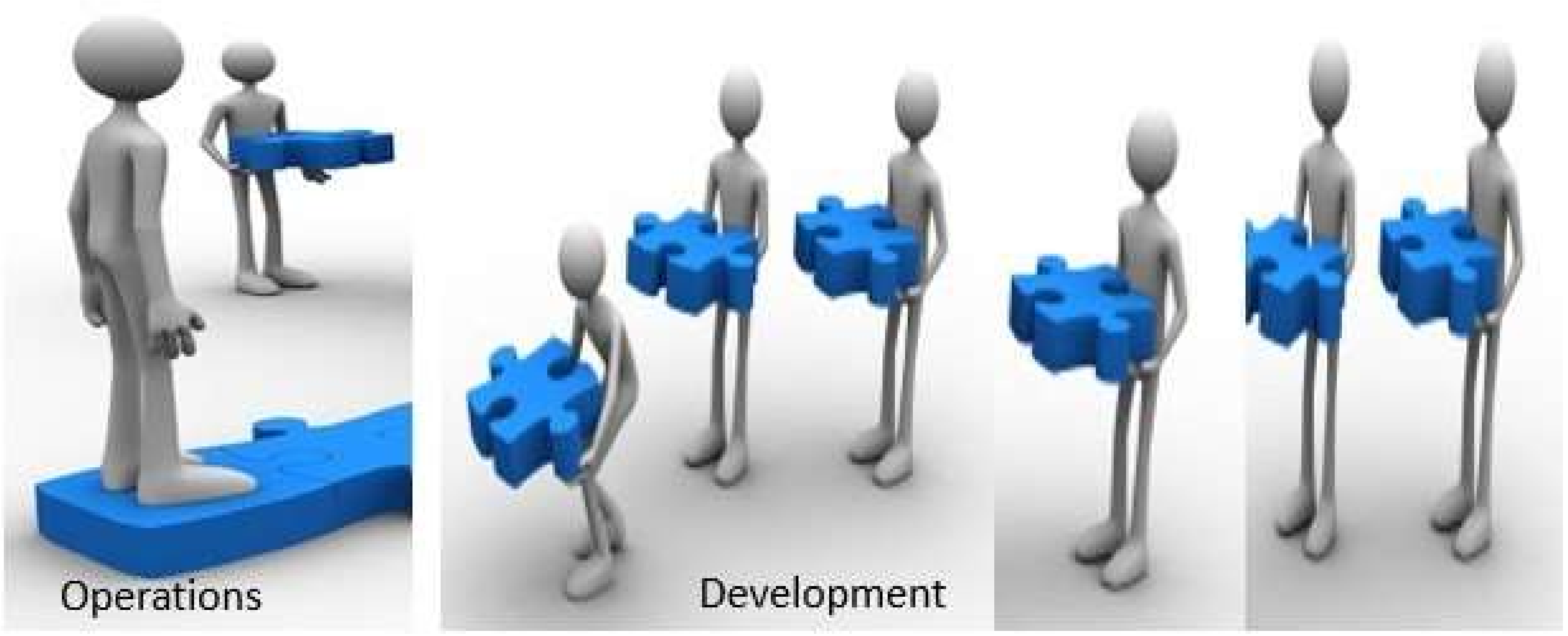
- Common Challenges Faced by Software Delivery Managers: Six common challenges faced by software delivery managers, including incorrect metrics, poor communication, multitasking, and disconnected priorities.

- Agile Development Create Problems Since software organizations have to rapidly deliver the software to keep up with the market, they adapt agile development and this increases the burden on operations.

- Operations team needs to be ready with pre-requirements including infrastructure, tools, software for all deployments coming in.

- This require the operations team to be extra agile to handle the changes coming from multiple agile development teams.

Agile Development Create Problems

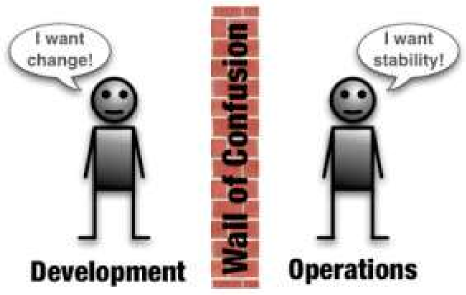


Problems of Delivering Software

- **Mindset Mismatch** Development creates new features and changes that brings value to business.

- Operations wants to maintain reliability and stability (by resisting rapid changes) to reflect the business integrity.

- Both of them have conflicting mindset and believe in what they are doing is right for the business.



Problems of Delivering Software

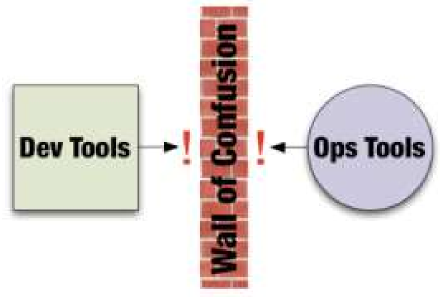
- Lack of Communication - There is no proper communication between development and operations and this leads to silos.

- Because of silos, sometimes operations are unable to deliver the changes in production.

- A simple example would be development handoff a package to operations which requires OS yy, ops doesn’t know this (production has OS xx) and deploy fails.

Problems of Delivering Software

- Lack of Unified Process There is no defined unified process throughout the entire lifecycle so that development and operations can follow the same.
- Development defines their own process while operations defines their own.
- Some scenarios include how to bring up and maintain the environments, maintain configurations, and do the release.



Problems of Delivering Software

- Mind The Gap Divide and conquer doesn’t work anymore.
- In general, development takes care of Dev, Sit, QA environments and operations takes care of Staging and Production environments.
- To execute the same tasks like packaging, deployment method, applying configurations, development uses some tools & methods while operations uses different tools & methods.
- Firefighting When development and operations teams act within silos, they often spend too much time firefighting problems, which reduces the time available to build innovative new applications and services

Technical Challenges

- Building And Maintaining Servers Time consuming and unproductive.
- Absence of Environment Management Differences in dev, qa, staging and prod environments.
- Slow Deployment Costly error prone manual deployments and efforts.
- No Shared Ownership Lack of feedback and proper metric leads, blame game.
- Lack Of Configuration Management Discrepancies in managing configurations.
- Deployments Are A Blocker Due to manual deploys, depends on specific engineer/skill.
- Production Downtime - Due to lack of defined pre-requirements, instructions, checklist.

Technical Challenges

- Hacking – Fixing directly in production and forgets to check-in into source control.
- Lack Of Development To Test Against Production-Like Systems.
- Increase in Time-To-Market Due to slow, manual, time consuming unreliable deploys.
- Decrease in Quality No Automated testing, increase cost/time to test.
- Increase in Defect cycle time Inability to reproduce fast and fix defects.
- Deployment Related Downtime in Production.
- Rollback Often

Technical Challenges

- Knowing the challenges in front of us, now we need to address all these challenges
- break the inter departmental silos, figure out a way to bridge the gap between development and operations, define a unified set of goals, process, tools and methods that everyone can follow, so that we can deliver stable software on time.
- But How?
- this is exactly why Scaled Agile (SAFe) has been introduced and has helped everybody to sync together and unified the process!

Principles of Software Delivery

- The principles of software delivery are essential for ensuring efficient, reliable, and high-quality software development and deployment.
- Automation at scale: Automation at scale means when we know a fix is available, we are primarily focused on delivering that fix to all customers and environments.
- Applying a specific minor update to an individual customer in an adhoc way is also valuable, and there is a place for transformational self service however the ultimate focus for every organization is automation at scale.

Principles of Software Delivery

- **Standardization across applications**: Standardization whether in the form of naming conventions, configuration, workflows, technologies or frameworks is important.

- Requiring standards like only allowing certain naming conventions in production services or even lower environments means you won't have automation break and there won't be special snowflakes to handle manually via toil.

- **Continuous feedback loops**: In the DevOps Handbook, The Second Way describes the principles of creating fast and continuous feedback loops from operations to development.

- By doing this, we shorten and amplify feedback loops so that we can see problems as they occur and radiate this information to everyone in the value stream.

Principles of Software Delivery

- Error budgets: Services should be evaluated based on the number of issues that occur throughout the lifecycle of their release, customer impact (outages/degradation of service) is one of the most valuable measures to determining impact.
- By better understanding the risk and challenges with each service delivered (and by breaking down monolithic apps into microservices) we can provide better gating as to which services will be released and which ones will require more work.
- Test in a lower environment first: This one almost seems too obvious to be in the list, however one might be surprised how many production services get deployed and run for customers without a lower environment first.
- Always include lower environment testing and deployment prior to delivering (or even troubleshooting) in production first.

Principles of Software Delivery

- Errors should be captured and improved: Every error is important.

- Often in software engineering a benign error will go overlooked and continuously be logged or returning a non 200 HTTP response. Even if the error is "Known" or could be classified as non-impactful (red herring in troubleshooting), that error still takes up space in the minds of engineers, in the logs/metrics/traces and ultimately makes an application less observable.

- Break up deployments to enable faster delivery: By breaking up deployments into small batches to deliver updates more quickly and regularly we can reduce risk of big issues showing up across multitudes of customers, correcting issues as they are found in the wild.

- Feature flags are a valuable way to deliver more modular updates of components in which we can deliver deliberate updates of increasing surface area to customers as we gain confidence in the updates before the entirety is delivered.

Principles of Software Delivery

- Data driven decision making: Embracing telemetry to build observability into our updates help us understand the changes we are introducing into production.
- We can test the code that is being cut by development pre and post update for quality and provide that data upstream.
- We can also test our deployment pipeline to determine if any environmental issues or deployment issues happen pre and post update on the side of operations.
- Decommission apps, environments and frameworks: Letting applications, frameworks and environments run when they are not needed is incredibly expensive.

Principles of Software Delivery

- Decommission apps, environments and frameworks: It is much easier to deploy new services than to know when and how we can turn off old services, or to understand if they are still needed.

- By creating a central software delivery and deployment mechanism we can more assuredly understand what applications may be running but could be flagged for decommissioning.

- Continuous delivery to continuous deployment: Making use of all the tools and experiences to mitigate risk, monitor quality and improve software delivery speed we eventually get to the place of continuous deployment.

- Continuous deployment is where each change committed to version control is integrated, tested and deployed into production.

Principles of Software Delivery

- Common Quote “The day of a software release tends to be a tense one”
- Antipattern: Deploying Software Manually
  - The production of extensive and detailed documentation for steps to be performed
  - Reliance on manual testing
  - Frequent calls to the development team to explain
  - Frequent corrections to the release process
  - Sitting bleary-eyed in front of a monitor at 2 A.M

Principles of Software Delivery

- Antipattern: Deploying to a Production-like Environment Only after Development Is Complete
  - Tester tested the system on development machines
  - Releasing into staging is the first time that operations people interact with the new release
  - Who Assembles? The Development Team
  - Collaboration between development and Operations?

Principles of Software Delivery

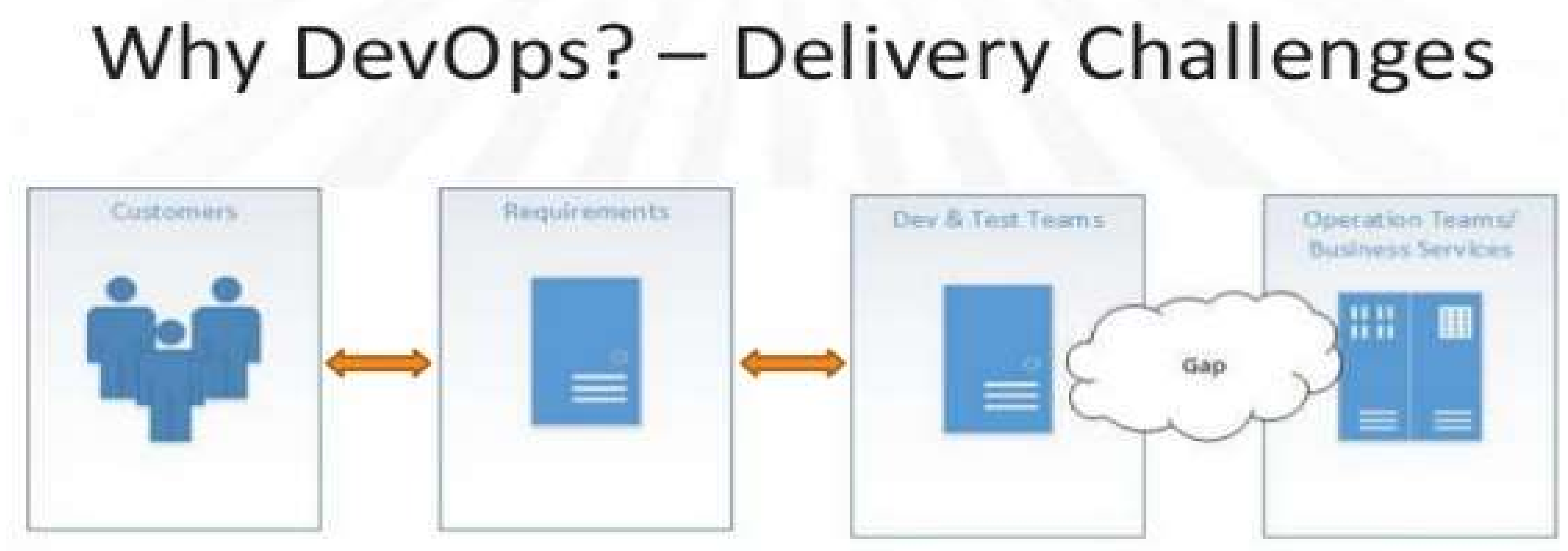
- Antipattern: Manual Configuration Management of Production Environments
  - Difference in Deployment to Stage and Production
  - Different host behave differently
  - Long time to prepare an environment
  - Cannot step back to an earlier configuration of a system
  - Modification to Configuration Directly

Problems of Delivering Software

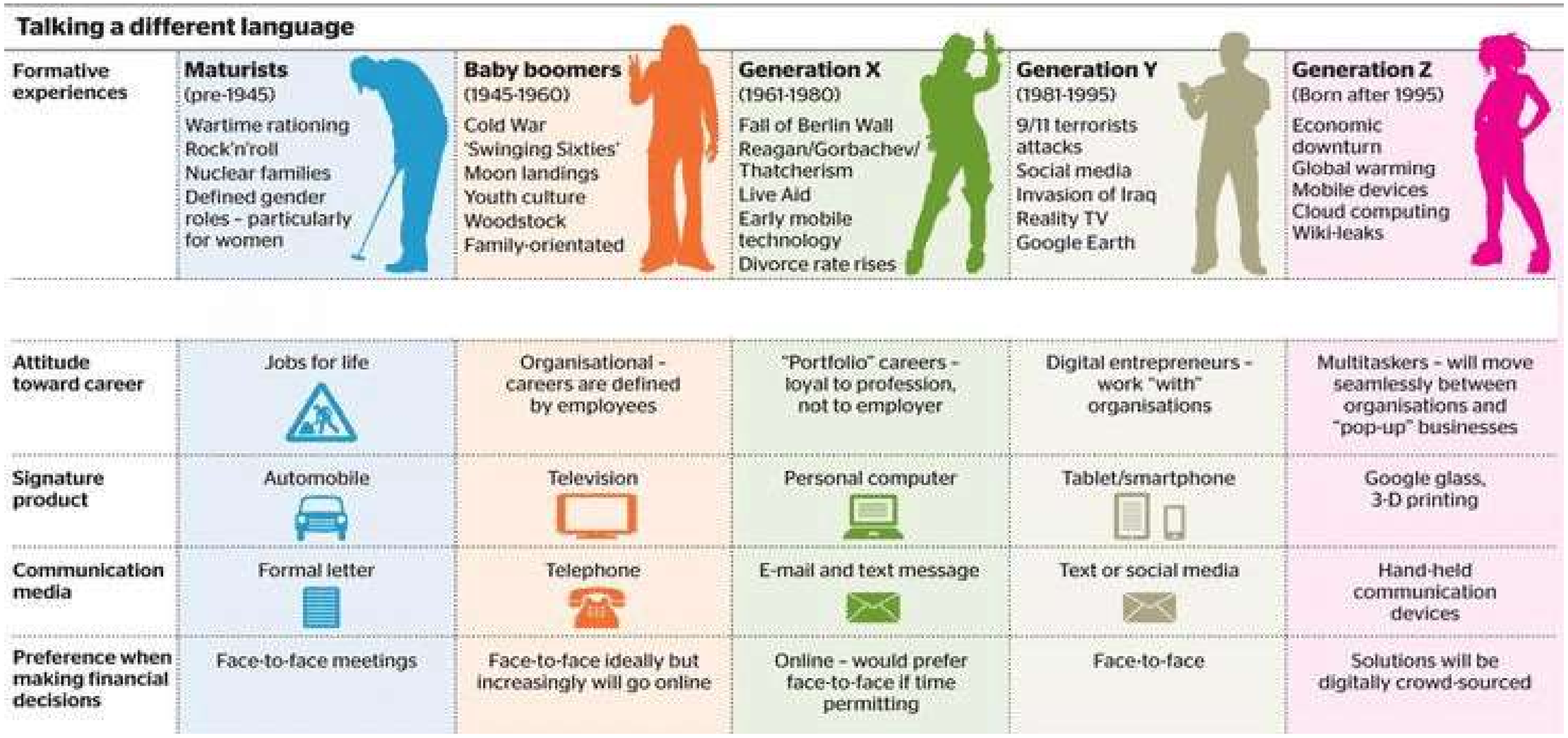
- Poor communication between Dev and Ops
- Opposing goals
  - Devs want to push new features
  - Ops want to keep the system available
  - It leads to slow-release schedule

- Limited capacity of operations staff
- Limited insight into operations
- Organizations want to reduce time to market for new features, without sacrificing quality
- Throw your final version over the fence and let operations worry about running it

Problems of Delivering Software



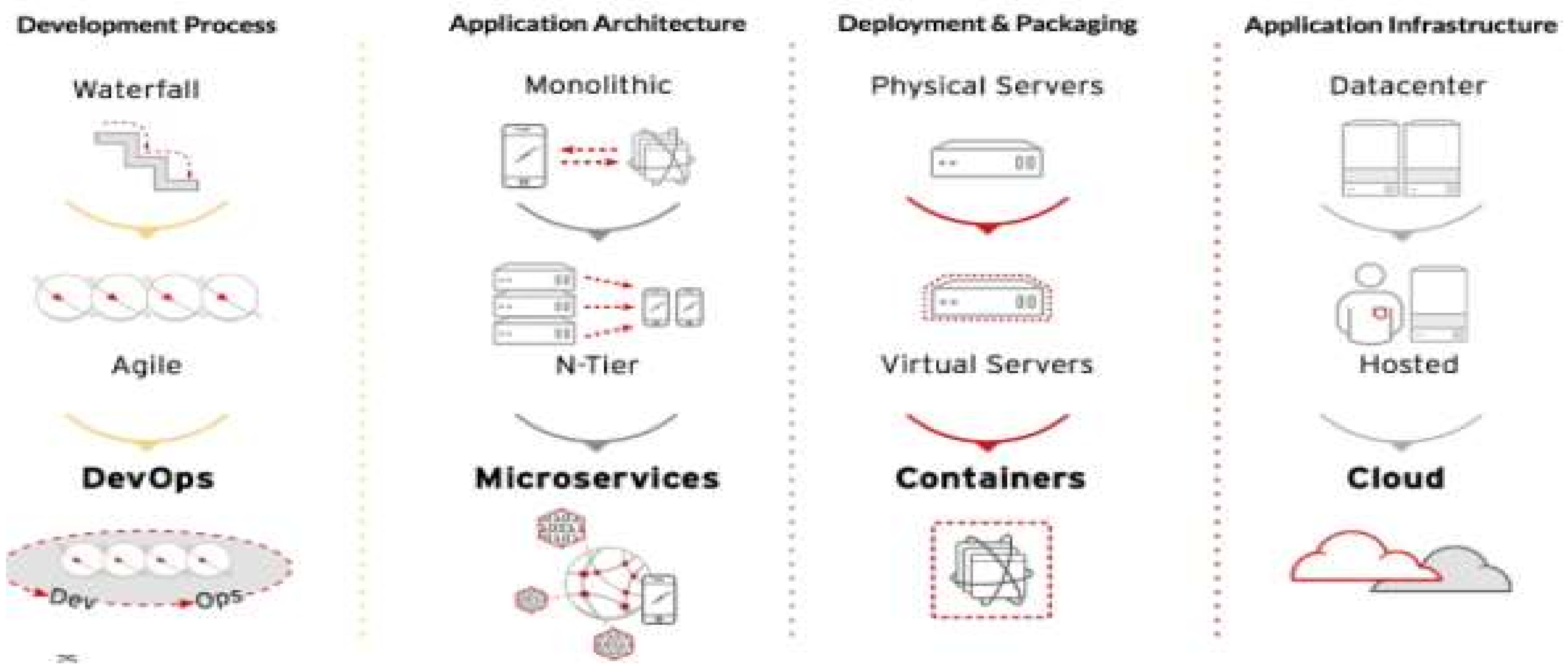
Problems of Delivering Software



Talking a different language

	Maturists (pre-1945)	Baby boomers (1945-1960)	Generation X (1961-1980)	Generation Y (1981-1995)	Generation Z (Born after 1995)
Formative experiences	Wartime rationing Rock'n'roll Nuclear families Defined gender roles - particularly for women	Cold War 'Swinging Sixties' Moon landings Youth culture Woodstock Family-orientated	Fall of Berlin Wall Reagan/Gorbachev/ Thatcherism Live Aid Early mobile technology Divorce rate rises	9/11 terrorists attacks Social media Invasion of Iraq Reality TV Google Earth	Economic downturn Global warming Mobile devices Cloud computing Wiki-leaks
Attitude toward career	Jobs for life	Organisational - careers are defined by employees	"Portfolio" careers - loyal to profession, not to employer	Digital entrepreneurs - work "with" organisations	Multitaskers - will move seamlessly between organisations and "pop-up" businesses
Signature product	Automobile	Television	Personal computer	Tablet/smartphone	Google glass, 3-D printing
Communication media	Formal letter	Telephone	E-mail and text message	Text or social media	Hand-held communication devices
Preference when making financial decisions	Face-to-face meetings	Face-to-face ideally but increasingly will go online	Online - would prefer face-to-face if time permitting	Face-to-face	Solutions will be digitally crowd-sourced

Problems of Delivering Software



Delivery : Dev View

- Why Gaps?
- Dev View:
- Mostly delivers features after testing in the development system
- Dev Systems may not be same as production
- Developers will have fast turn around time with respect to features.
- Not much concerned about the infrastructure as well as deployment impact because of code change

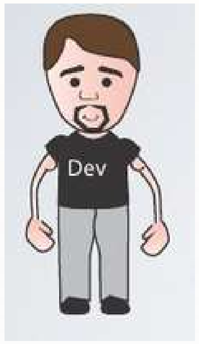
Delivery : Dev View

- Why Gaps?
- Ops View:
- Worries more about PSR
- Reward mainly for uptime
- Developers will have fast turn around time with respect to features.
- Lesser turn around time with respect to features deployment and testing due to large number of dev builds coming their way.
- Very much concerned about the infrastructure as well as deployment impact because of code change

Dev and Ops dialogue

Dev:

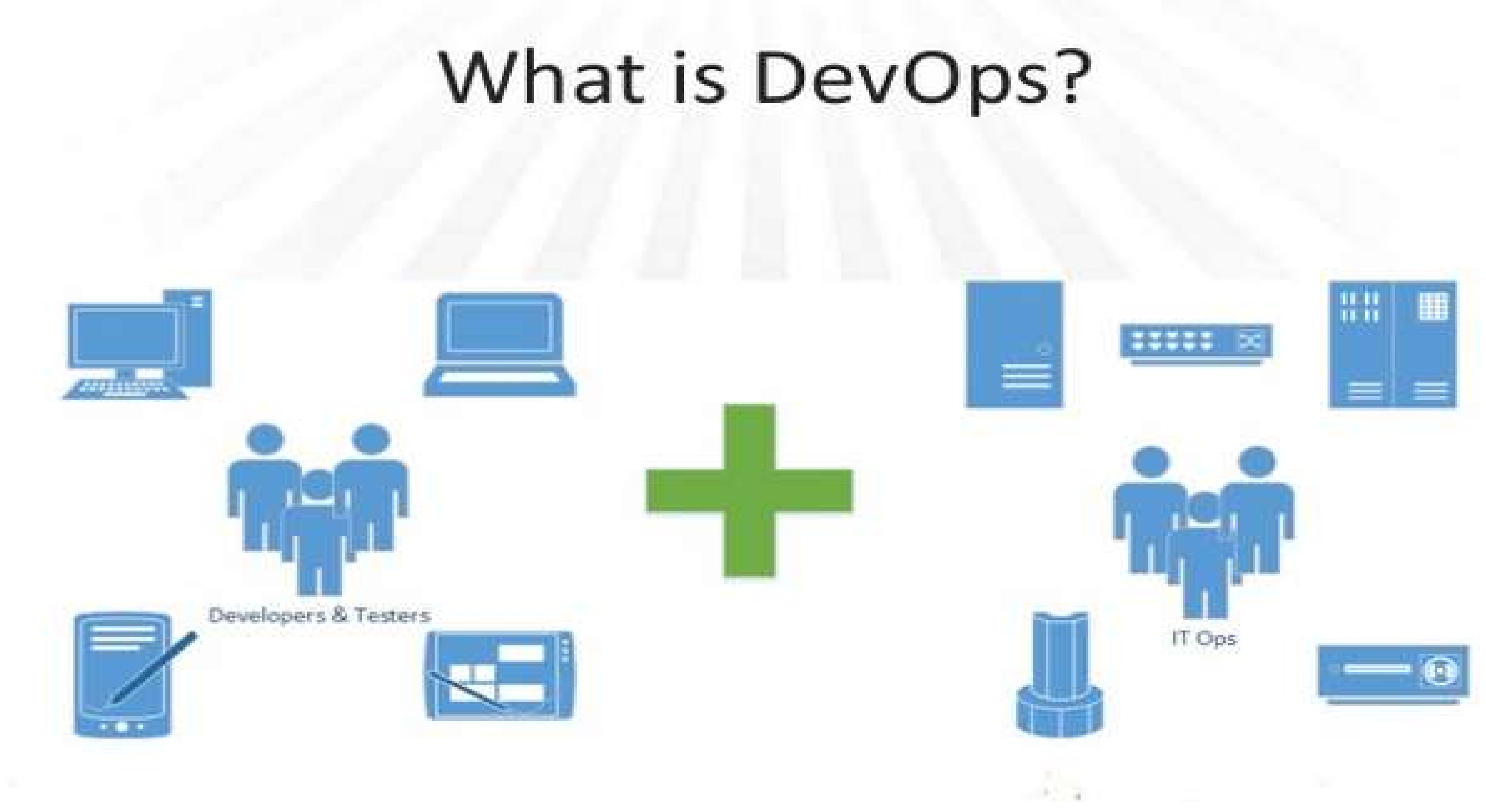
- Put the current release live, NOW!
- It works on my machine
- We need this Yesterday
- You are using the wrong version



Ops:

- What are the dependencies?
- No machines available…
- Which DB?
- High Availability?
- Scalability?

Need for DevOps

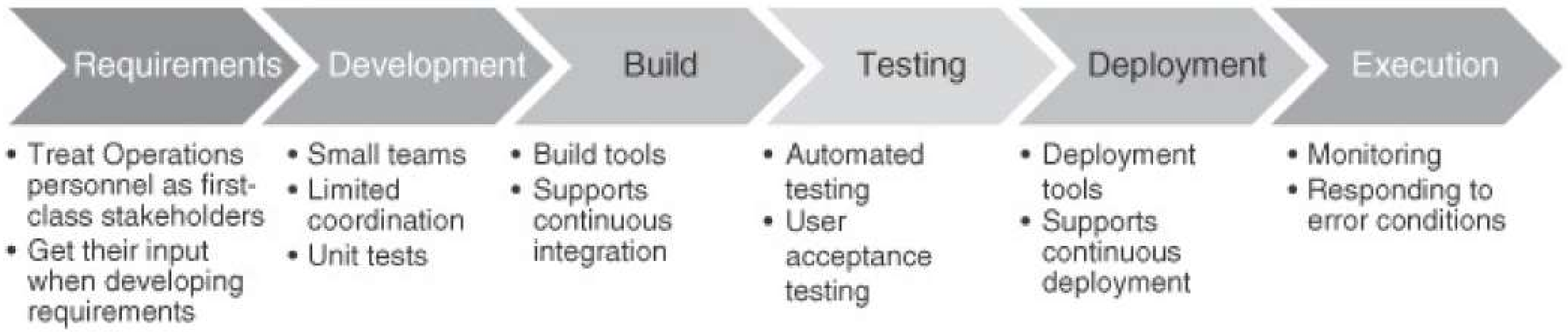


DevOps – Development + Operations

- Set of principles, practices, and techniques helps to seamless communication, collaboration and integration between software developers and information technology (IT) operation professionals and automate the process.

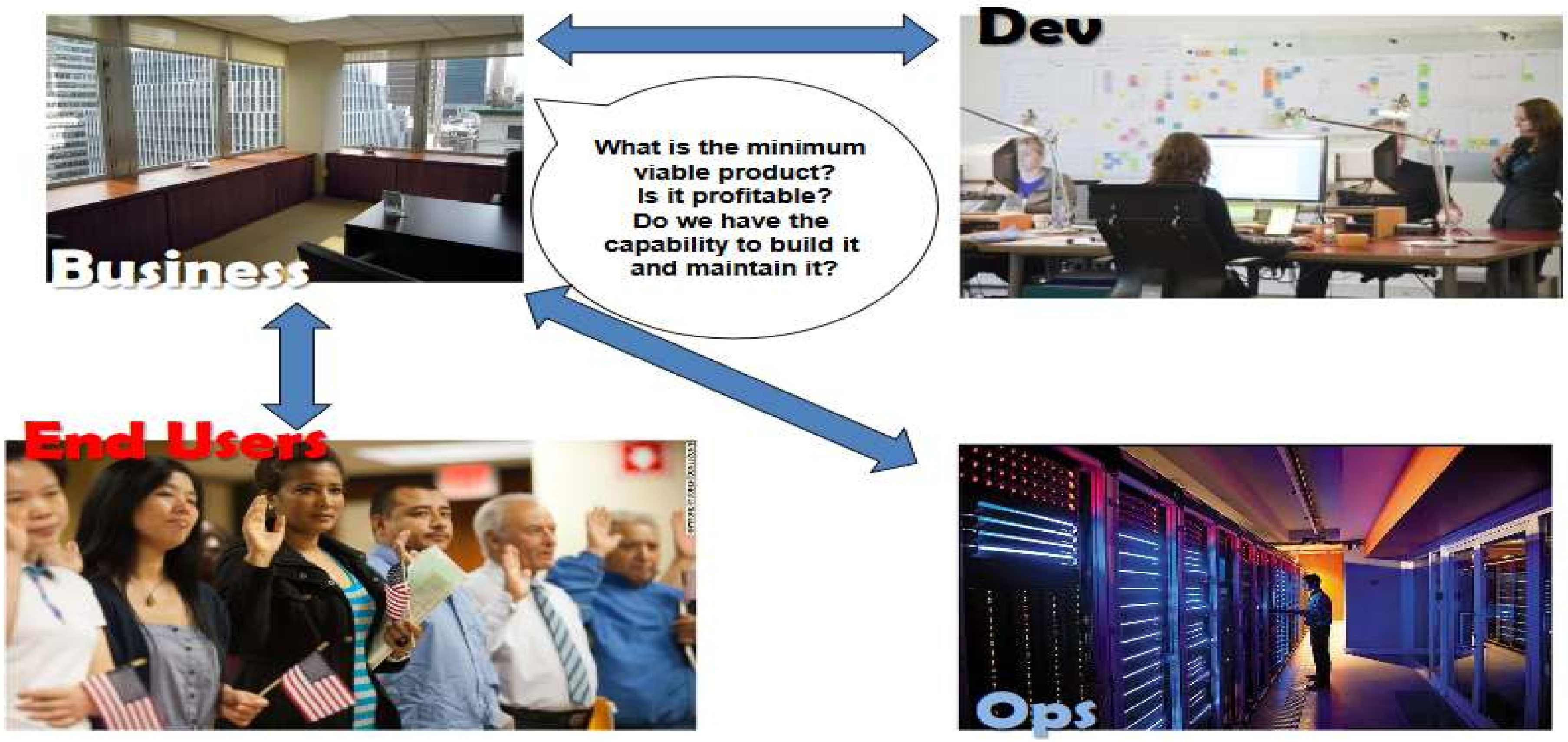
- collaborative way of developing and deploying software
- Bridge the gap between build/test and Operations
- To create low-risk, frequent, cheap, rapid, and predictable process in software delivery
- Create a Repeatable, Reliable Process for Releasing Software
- Speed is essential because there is an opportunity cost associated with not delivering software
- Bringing “agile” methods to operations

DevOps –Life Cycle

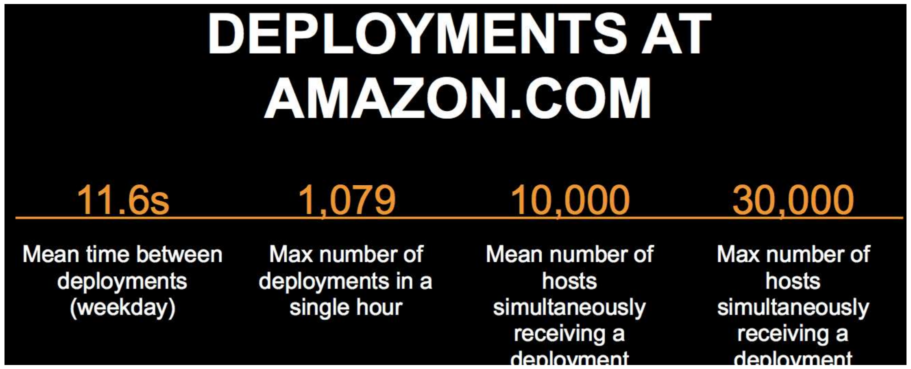

Requirements
Development
Build
Testing
Deployment
Execution
• Treat Operations personnel as first-class stakeholders
• Get their input when developing requirements
• Small teams
• Limited coordination
• Unit tests
• Build tools
• Supports continuous integration
• Automated testing
• User acceptance testing
• Deployment tools
• Supports continuous deployment
• Monitoring
• Responding to error conditions

- Treat Ops as first-class citizens throughout the lifecycle (E.g. During requirement)
  - Many decisions can make operating a system harder or easier
  - Logging and monitoring to suit Ops
- Make Dev more responsible for relevant incident handling
  - Shorten the time between finding and repairing errors

Evolution of DevOps- Define the stages of a DevOps evolution

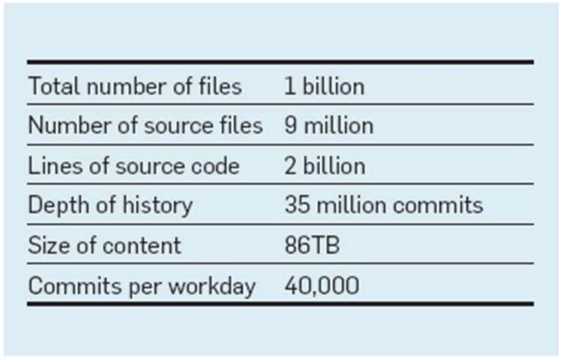

Business
Dev
What is the minimum viable product? Is it profitable? Do we have the capability to build it and maintain it?
End Users
Ops

Evolution of DevOps- Define the stages of a DevOps evolution



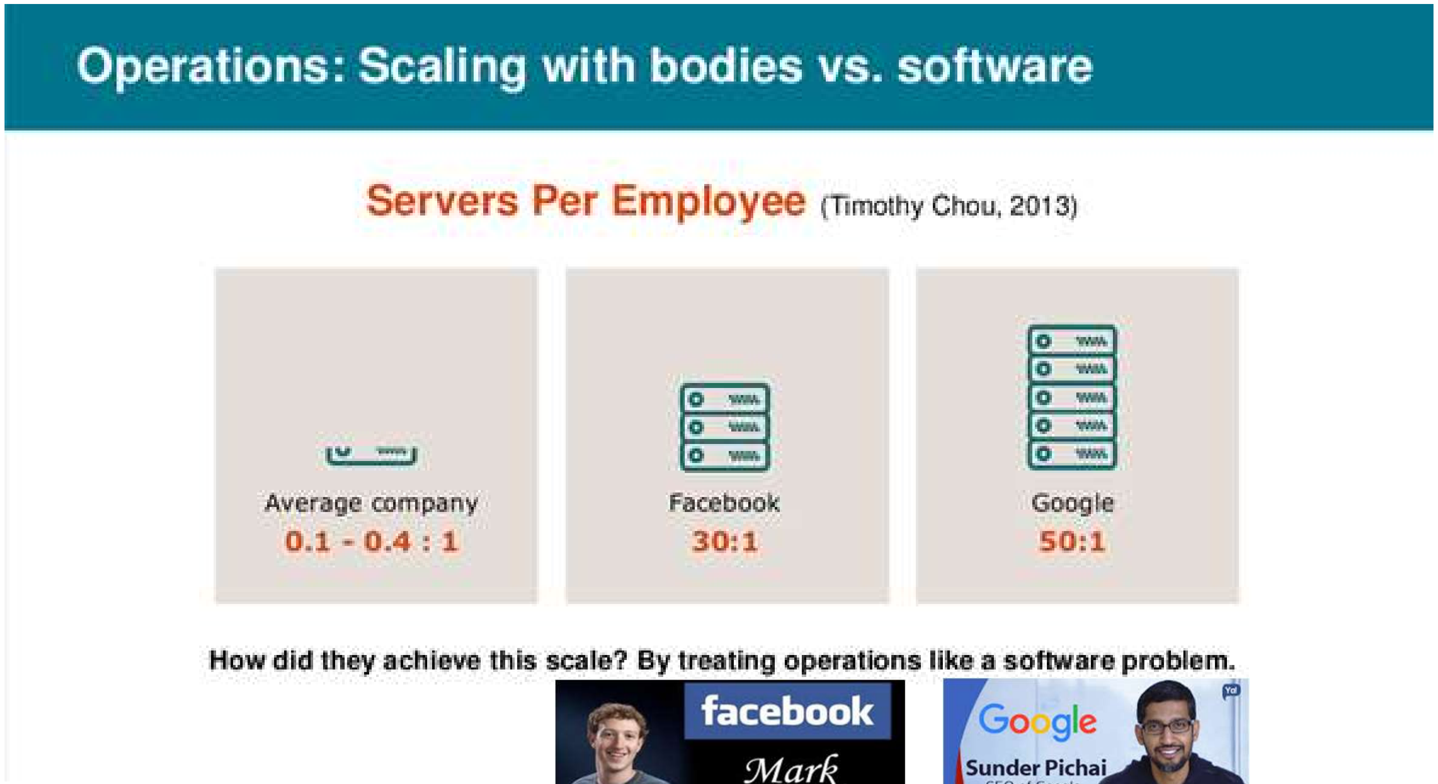
DEPLOYMENTS AT AMAZON.COM

11.6s	1,079	10,000	30,000
Mean time between deployments (weekday)	Max number of deployments in a single hour	Mean number of hosts simultaneously receiving a deployment	Max number of hosts simultaneously receiving a deployment



Total number of files	1 billion
Number of source files	9 million
Lines of source code	2 billion
Depth of history	35 million commits
Size of content	86TB
Commits per workday	40,000

Evolution of DevOps- Define the stages of a DevOps evolution



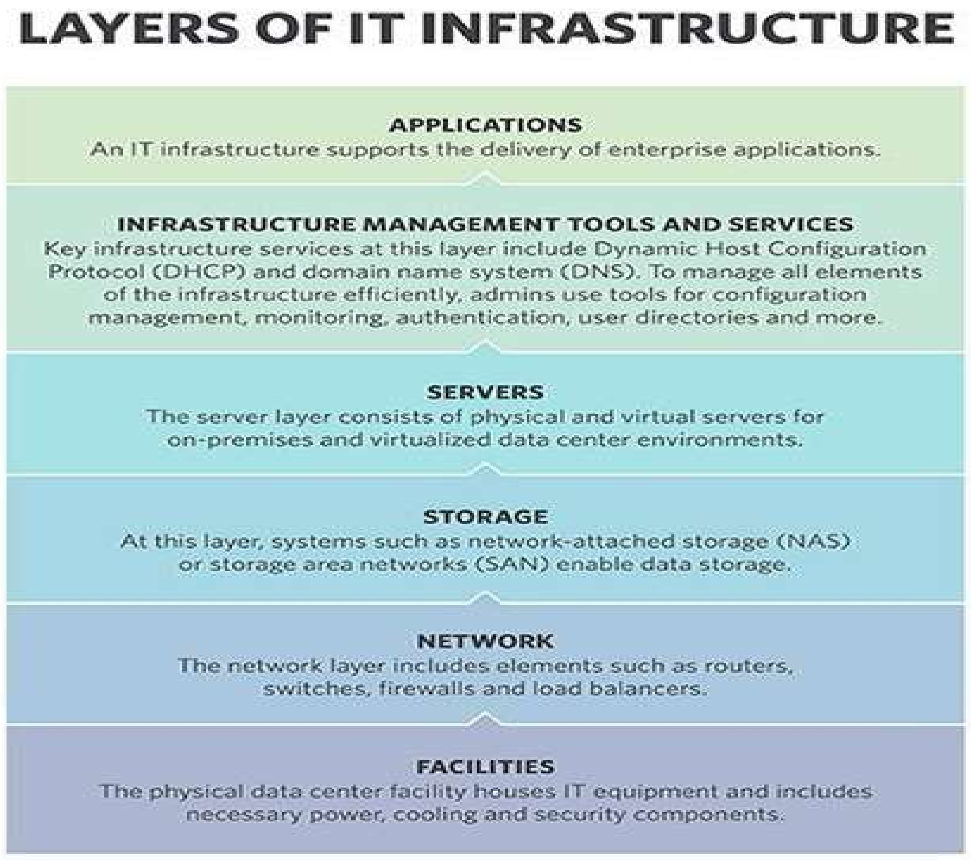
Operations: Scaling with bodies vs. software

Servers Per Employee (Timothy Chou, 2013)

Average company	Facebook	Google
0.1 - 0.4 : 1	30:1	50:1

How did they achieve this scale? By treating operations like a software problem.

Evolution of DevOps- Define the stages of a DevOps evolution



LAYERS OF IT INFRASTRUCTURE

APPLICATIONS
An IT infrastructure supports the delivery of enterprise applications.

INFRASTRUCTURE MANAGEMENT TOOLS AND SERVICES
Key infrastructure services at this layer include Dynamic Host Configuration Protocol (DHCP) and domain name system (DNS). To manage all elements of the infrastructure efficiently, admins use tools for configuration management, monitoring, authentication, user directories and more.

SERVERS
The server layer consists of physical and virtual servers for on-premises and virtualized data center environments.

STORAGE
At this layer, systems such as network-attached storage (NAS) or storage area networks (SAN) enable data storage.

NETWORK
The network layer includes elements such as routers, switches, firewalls and load balancers.

FACILITIES
The physical data center facility houses IT equipment and includes necessary power, cooling and security components.

Evolution of DevOps- The stages of a DevOps evolution

- Normalize the technological stack
  - Teams have begun adopting true Agile methods and are implementing proper version control as they seek to provide continuous integration and delivery
  - practitioners must build on a standard set of technologies, and put application configurations in version control

Evolution of DevOps- The stages of a DevOps evolution

- Standardize and reduce variability
  - Both development and operation teams are working toward one goal – Variability
  - Teams must work to make sure tech is further consolidated to a single OS or OS family, process complexity is reduced, and early collaboration opportunities are explored
  - At this stage, system configurations should also be placed in version control and applications should be re-architected to fit business needs

Evolution of DevOps- The stages of a DevOps evolution

- Develop DevOps practices

  - Administrative burden must be detected and eliminated so that the teams can work and move faster

  - Bureaucratic red tape must be eliminated so that individual contributors can do work without approval from outside the team and changes can be made without as much wait time

  - Collaboration between development and operating teams intensifies

Evolution of DevOps- The stages of a DevOps evolution

- Automate infrastructure delivery
  - lessen the discrepancy between development's output and operations' delivery times
  - security configurations, system configurations, and provisioning are automated, the DevOps team can deliver faster and is better set up for future self-service
  - Agile infrastructure also aims to provide developers with quick access to an environment which is conducive to the deployment of their systems

Evolution of DevOps- The stages of a DevOps evolution

- Provide self-service capabilities
  - Many of the self-service capabilities are realized and more resources are made available.
  - Here, application developers can deploy testing environments on their own and success metrics are clearly visible to the team.

Key DevOps Practices

- Release planning
- Continuous integration
- Continuous delivery
- Continuous testing
- Continuous monitoring and feedback

DevOps– Practices

1. Release planning

- Release planning is a critical business function, driven by business needs to offer capabilities to customers and the timelines of these needs.

- Businesses require well defined release planning and management processes that drive release roadmaps, project plans, and delivery schedules, as well as end-to-end traceability across these processes.

- Traditionally, release planning accomplished by using spreadsheets and holding meetings (often, long ones) with all stakeholders across the business to track all business needs applications under development, their development status, and release plans.

DevOps– Practices

Release planning

- Well-defined processes and automation, however, eliminate the need for those spreadsheets and meetings, and enable streamlined and enable predictable releases.
- Leveraging lean and agile practices also results in smaller, more frequent releases, permitting enhanced focus on quality.

DevOps– Practices

2. Continuous integration

- Allowing large teams of developers, working on cross-technology components in multiple locations, to deliver software in an agile manner.
- Each team's work is continuously integrated with that of other development teams and then validated.
- Continuous integration reduces risk and identifies issues earlier in the software development life cycle.

DevOps– Practices

3. Continuous Delivery/Deployment

- Continuous integration naturally leads to the practice of continuous delivery/deployment
- Process of automating the deployment of the software to the testing, system testing, staging, and production environments.
- Automated process in all environments to improve efficiency and reduce the risk introduced by inconsistent processes

DevOps– Practices

4. Continuous Delivery/Deployment

- processes that they adopt may vary from project to project, based on individual testing needs and on the requirements of service level agreements.
- Adopt processes in three areas
- (1) Test environment provisioning and configuration
- (2) Test data management
- (3) Integration, functional, performance, and security testing

DevOps– Practices

5. Continuous monitoring and feedback

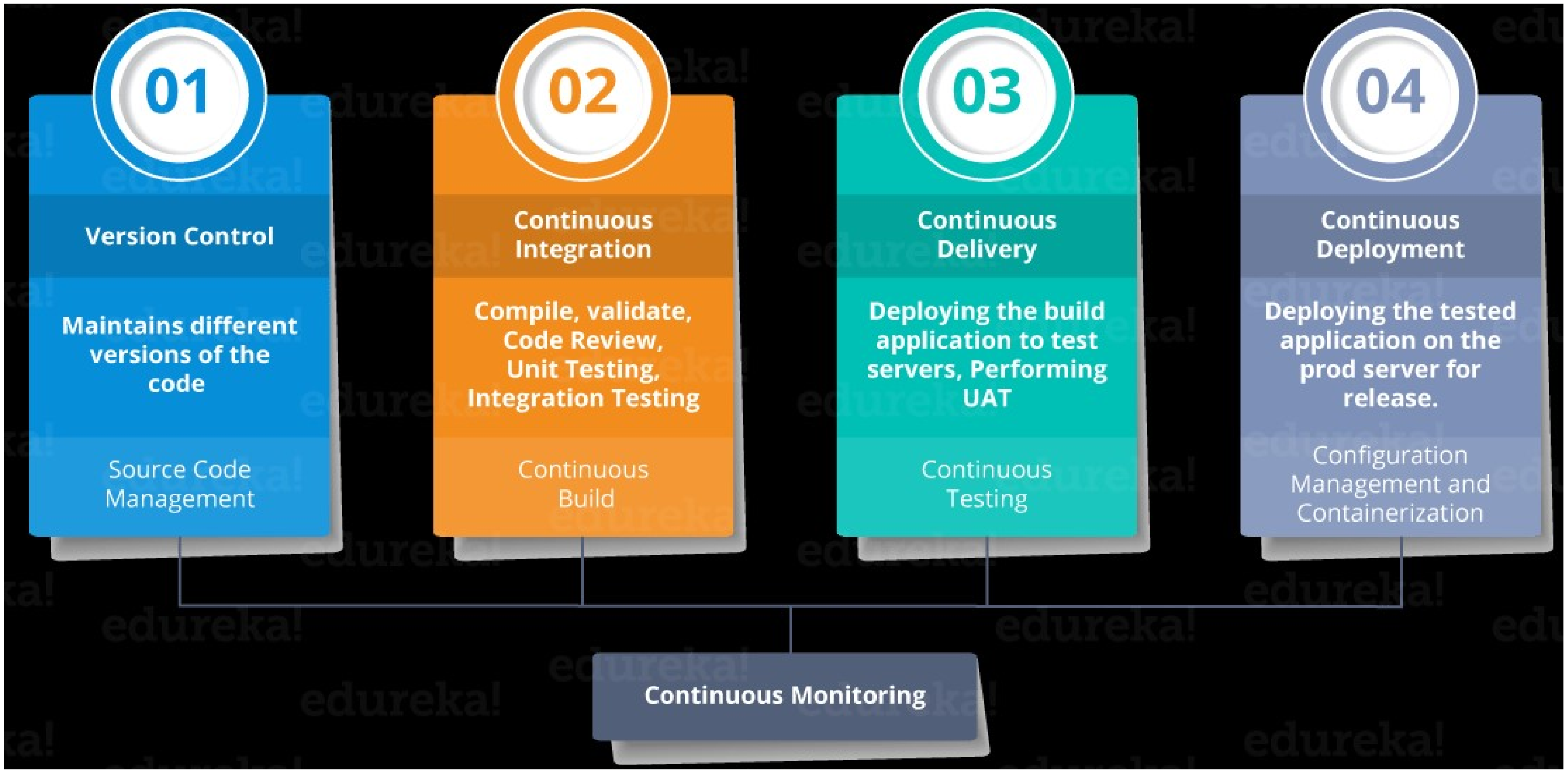
- Customer feedback comes in different forms, such as tickets opened by customers, formal change requests, informal complaints, and ratings in app stores.
- Feedback also comes from monitoring data.
- This data comes from the servers running the application; from Development, QA, and Production; or from metrics tools embedded in the application that capture user actions.
- Businesses need well-defined processes to absorb the feedback from myriad sources and incorporate them into software delivery plans.

DevOps– Practices

6. Continuous Improvement

- Process adoption isn’t a one-time action, but an ongoing process.
- An organization should have built-in processes that identify areas for improvement as the organization matures and learns from the processes it has adopted.
- Many businesses have process improvement teams that work on improving processes based on observations and lessons learned
- Teams that adopt the processes to self-assess and determine their own process improvement paths

DevOps Lifecycle Processes


01
Version Control
Maintains different versions of the code
Source Code Management
02
Continuous Integration
Compile, validate, Code Review, Unit Testing, Integration Testing
Continuous Build
03
Continuous Delivery
Deploying the build application to test servers, Performing UAT
Continuous Testing
04
Continuous Deployment
Deploying the tested application on the prod server for release.
Configuration Management and Containerization
Continuous Monitoring
edureka!

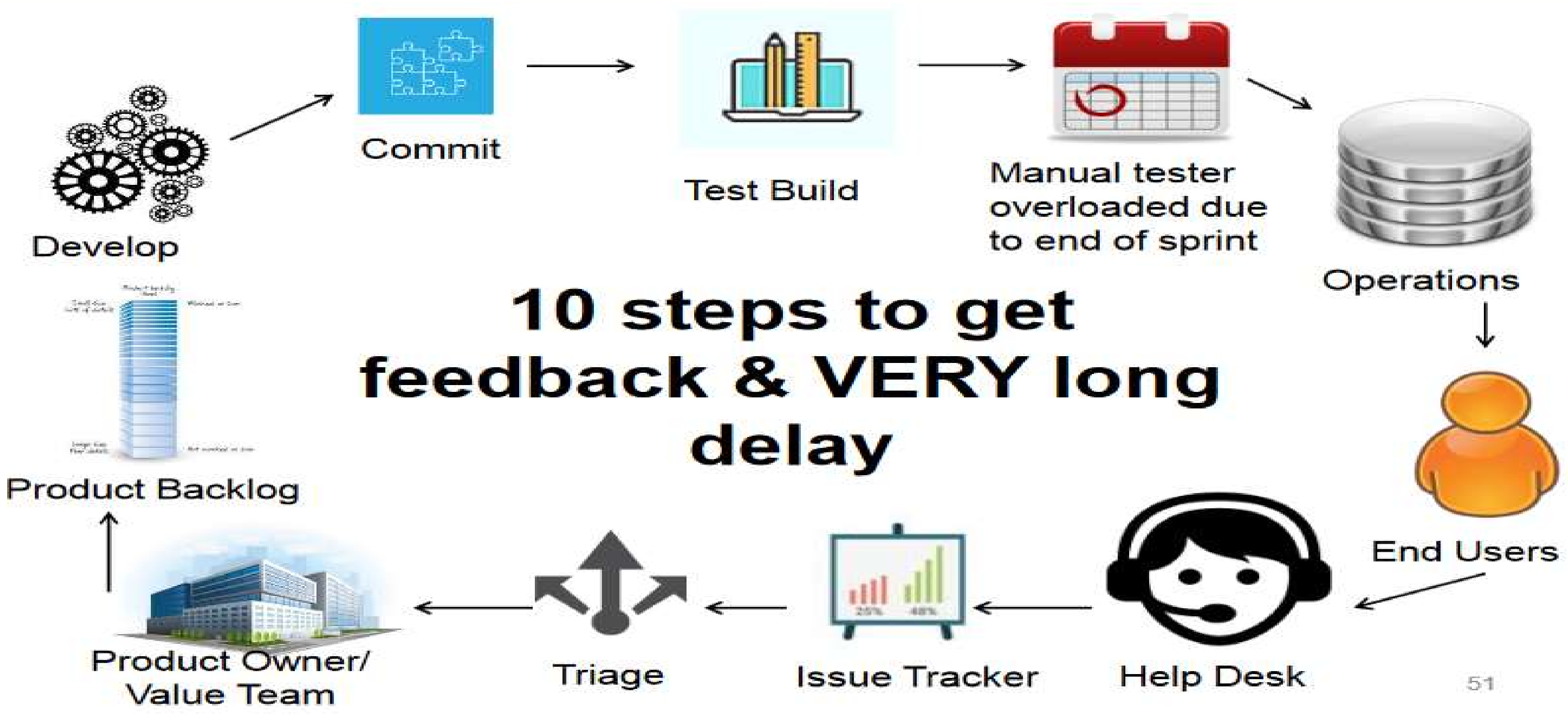
Continuous Integration

- Development practice that requires developers to integrate code into a shared repository several times a day.
- If an organization is set up to merge all branching source code together on one day, the resulting work can be tedious, manual, and time-intensive
- Developers make code commits multiple times per day
- Continuous integration (CI) helps developers merge their code changes back to a shared branch

Continuous Integration

- Each code piece passes through a set of automated tests to detect every error and bug as early as possible
- Changes are validated by automatically building the application and running different levels of automated testing, typically unit and integration tests, to ensure the changes haven’t broken the app
- The major priority for the development team after the issue is detected is to remove it.

Continuous Integration


Develop
Commit
Test Build
Manual tester overloaded due to end of sprint
Operations
End Users
Help Desk
Issue Tracker
Triage
Product Owner/ Value Team
Product Backlog
10 steps to get feedback & VERY long delay
51

Continuous Inspection

- Continuous Inspection is the process of an automated code review of inspection conducted for your code before the actual tests are run

- Inspection analyzes the code based on a set of predefined rules.

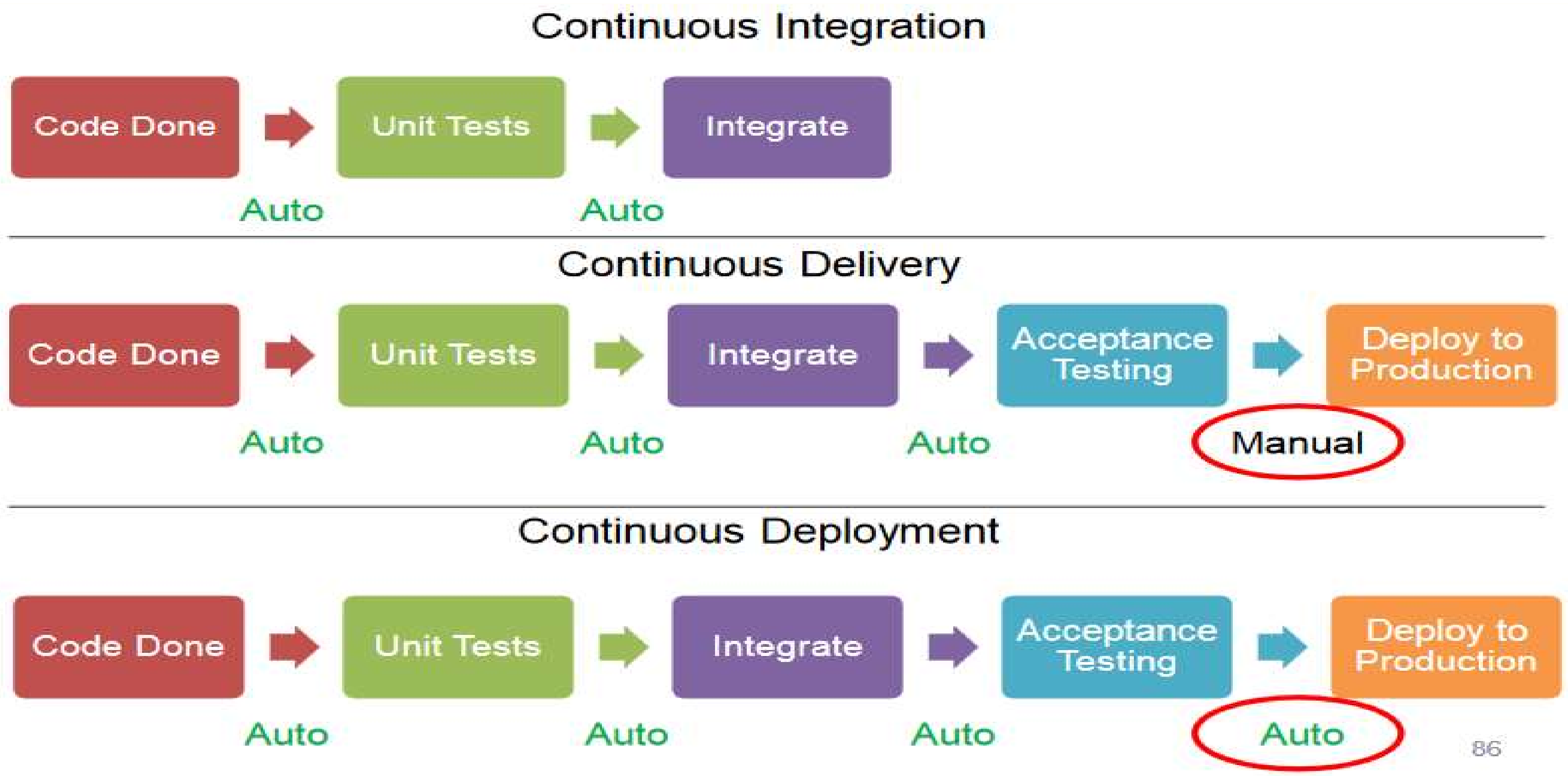
- Inspectors (or static and dynamic analysis tools) are directed by identified standards that teams should adhere to (usually coding or design metrics).

- Examples of inspection targets include coding “grammar” standards, architectural layering adherence, code duplication, and many others.

Continuous Inspection

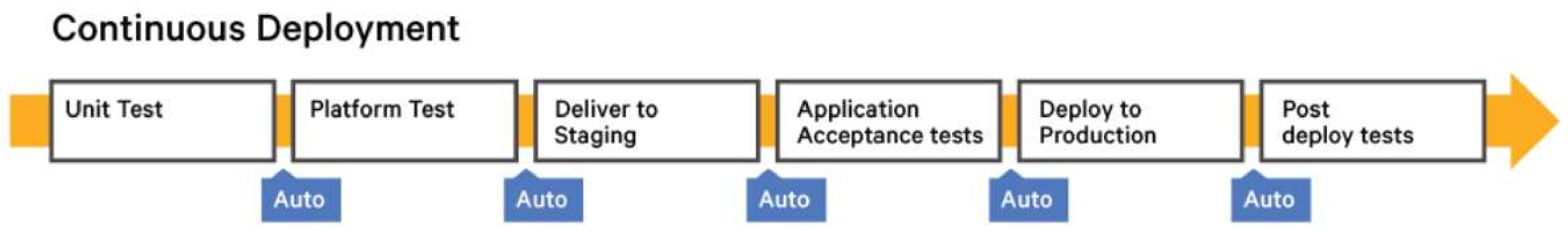
- Continuous Inspection reduces the time between a discovery and a fix.
  - Run-Time Defects
  - Preventative Practices (Code structure, scoping, preprocessor limitations etc.)
  - Metric Thresholds
  - Language usage
  - Style
  - Portability

Continuous Inspection

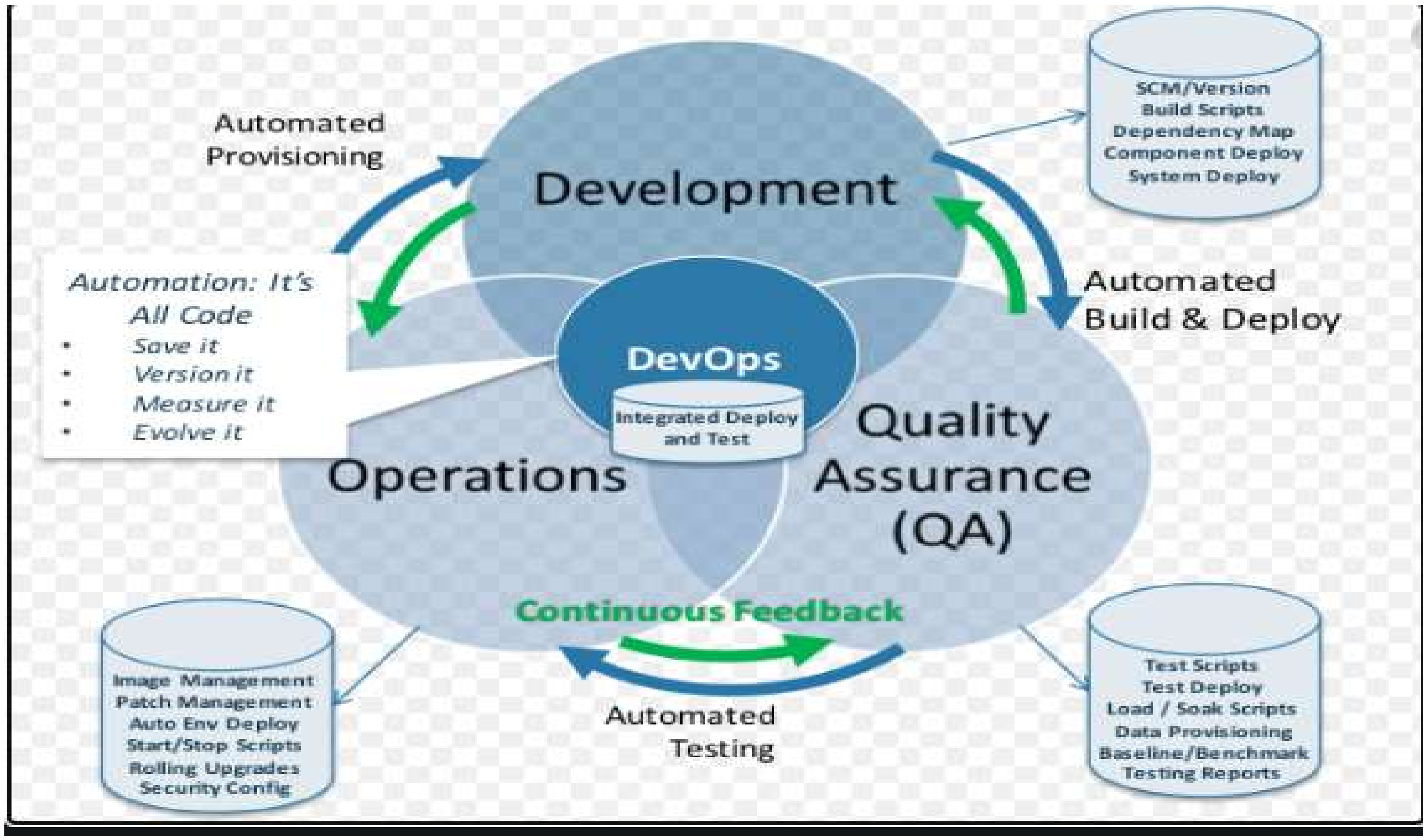


Continuous Deployment

- Stage where the code is deployed to the production servers.
- It is also important to ensure that the code is correctly deployed on all the servers
- Require an Agile process that provides a framework where you work on small, frequent changes and obtain feedback.



Continuous Deployment

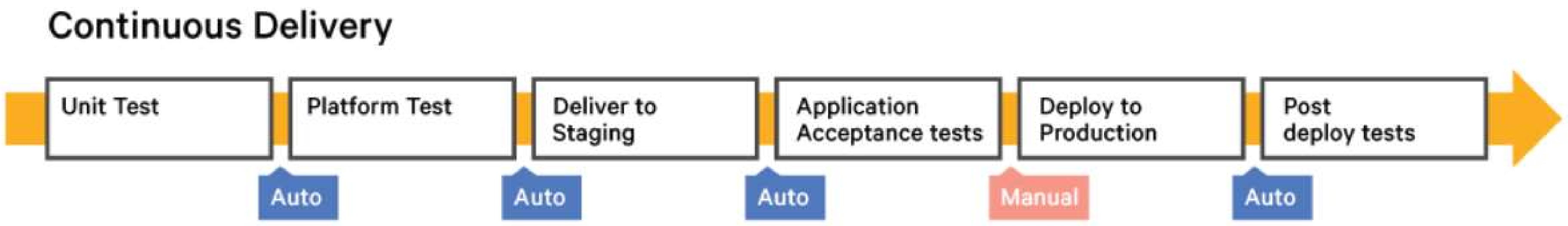


Continuous Delivery

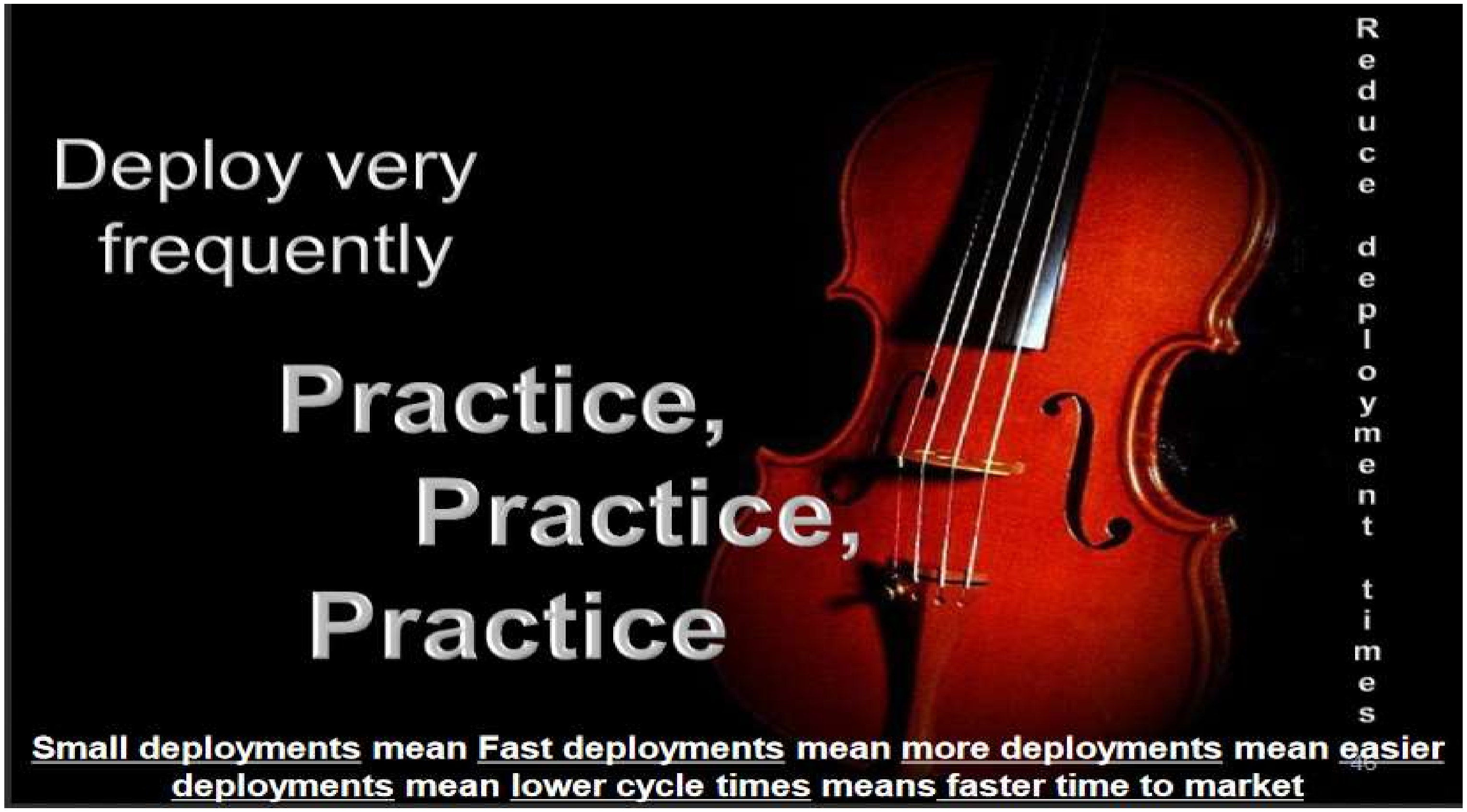
- state of being ready and able to release any version at any time on any platform.
- Ready to be deployed.
- With Continuous Delivery, "Deploy to Production" is a manual process, meaning that it is initiated manually.
- Require an Agile process that provides a framework where you work on small, frequent changes and obtain feedback.

Continuous Delivery

- Workflow based process which accepts tested software to build payload from CI server.
- Automate deployment into Working Quality assurance, pre-production or production environment



Continuous Delivery


Deploy very frequently

Practice, Practice, Practice

Reduce deployment times

Small deployments mean Fast deployments mean more deployments mean easier deployments mean lower cycle times means faster time to market

Continuous Delivery

- Create a Repeatable, Reliable Process for Releasing Software
- Automate Almost Everything
- Keep Everything in Version Control
- If It Hurts, Do It More Frequently, and Bring the Pain Forward Build Quality In
- “The Earlier you catch the defects, the cheaper they are to fix” Done, Means Released
- Everybody Is Responsible for the Delivery Process
- Continuous Improvement

Continuous Monitoring

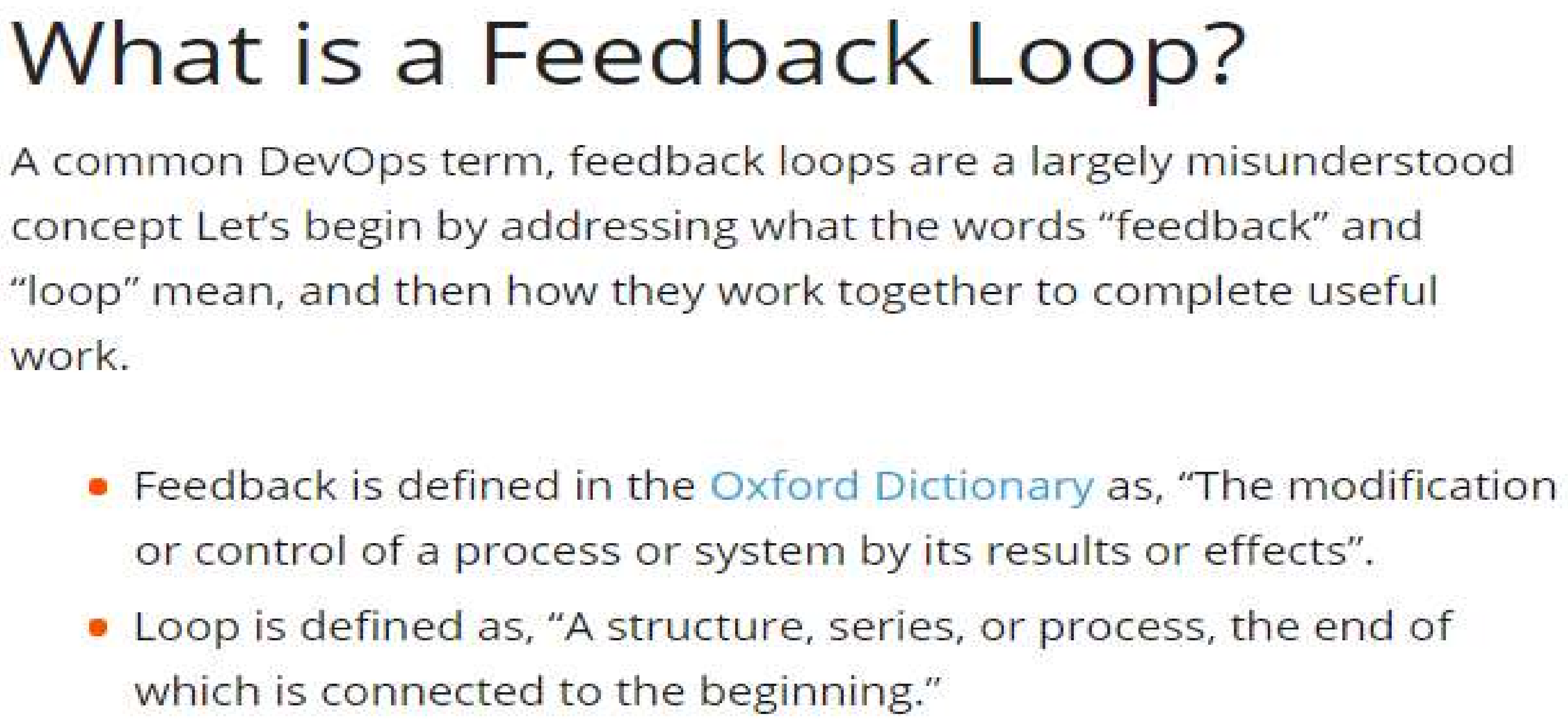
- The process and technology required to incorporate monitoring across each phase of your DevOps and IT operations lifecycles.

- It helps to continuously ensure the health, performance, and reliability of your application and infrastructure as it moves from development to production.

- Continuous monitoring builds on the concepts of Continuous Integration and Continuous Deployment (CI/CD) which help you develop and deliver software faster and more reliably to provide continuous value to your users.

- It is a crucial element in gaining true efficiency and scalability

Continuous Monitoring

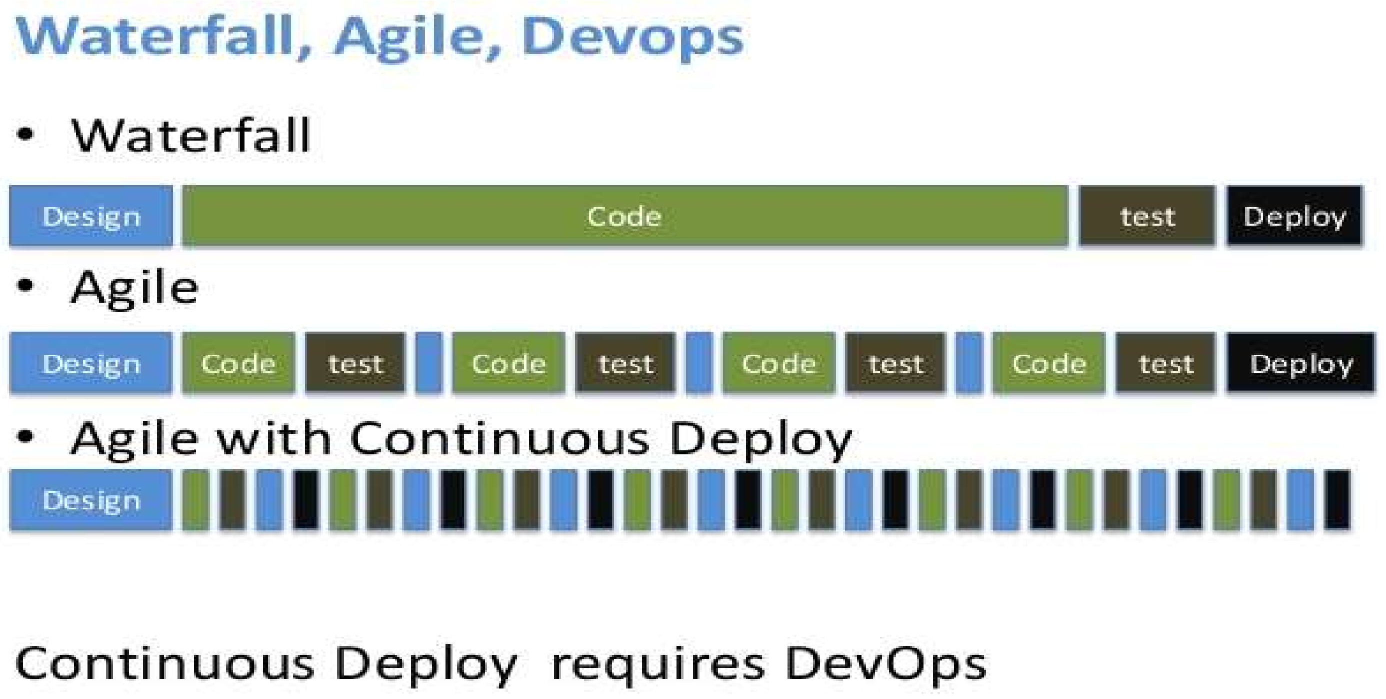


What is a Feedback Loop?

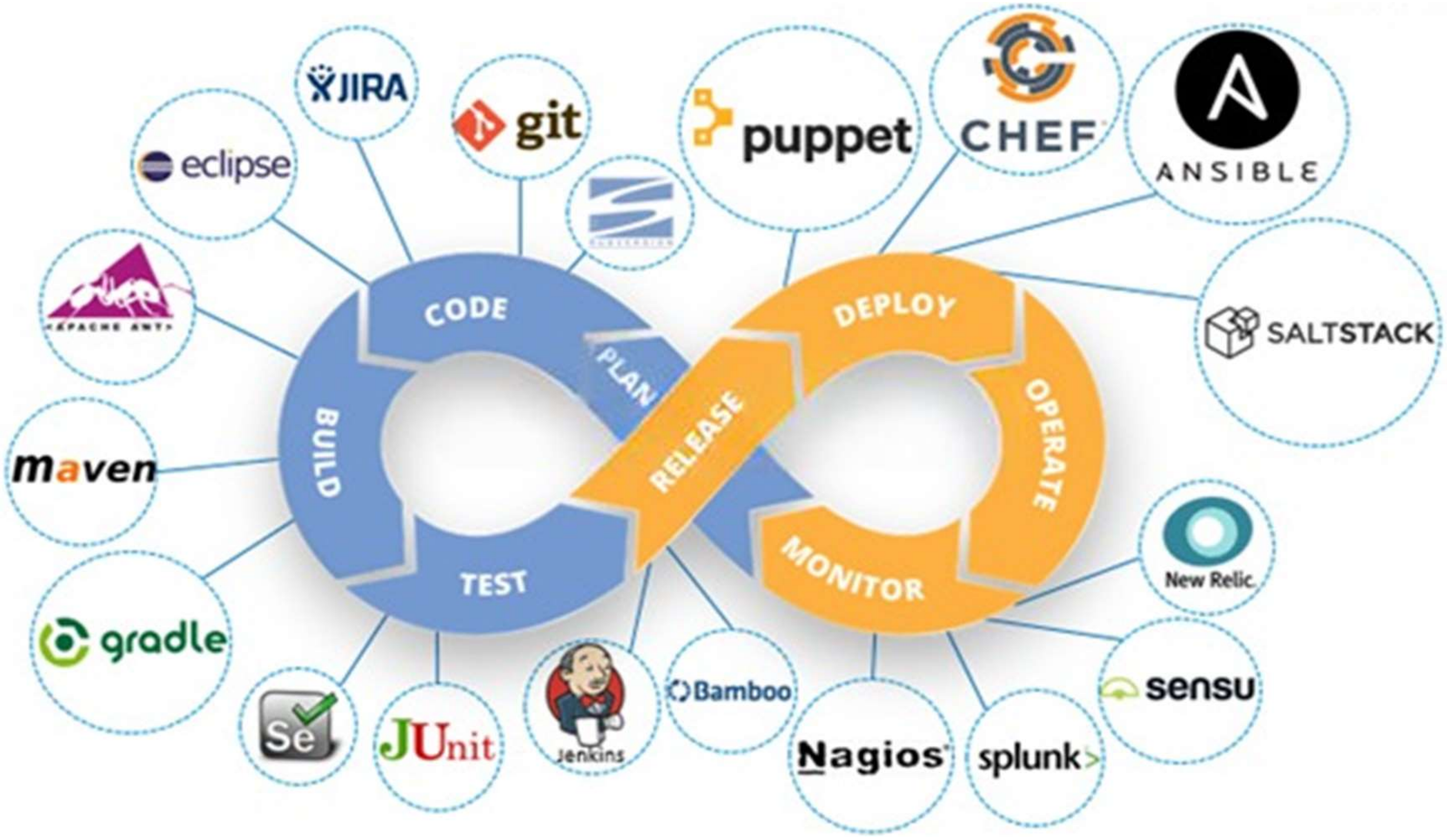
A common DevOps term, feedback loops are a largely misunderstood concept Let's begin by addressing what the words "feedback" and "loop" mean, and then how they work together to complete useful work.

- Feedback is defined in the Oxford Dictionary as, "The modification or control of a process or system by its results or effects".
- Loop is defined as, "A structure, series, or process, the end of which is connected to the beginning."

DevOps Culture



Continuous Integration, Continuous Inspection, Continuous Deployment, Continuous Delivery, Continuous Monitoring



DevOps Case Study - IBM

- By leveraging IBM DevOps solutions featuring continuous integration, continuous delivery, testing, deployment, compliance, monitoring and value stream management, organizations.

- Develop with Speed and Quality
- Respond to the market faster and build secure, engaging experiences

- Secure by Default
- Bullet proof development efforts with this IBM principle that ensures products and services are configured securely out of the box to protect against common threats and vulnerabilities.

- Deploy continuously at scale
- Continuously deploy software into multi-architecture development, test and production environments at scale.

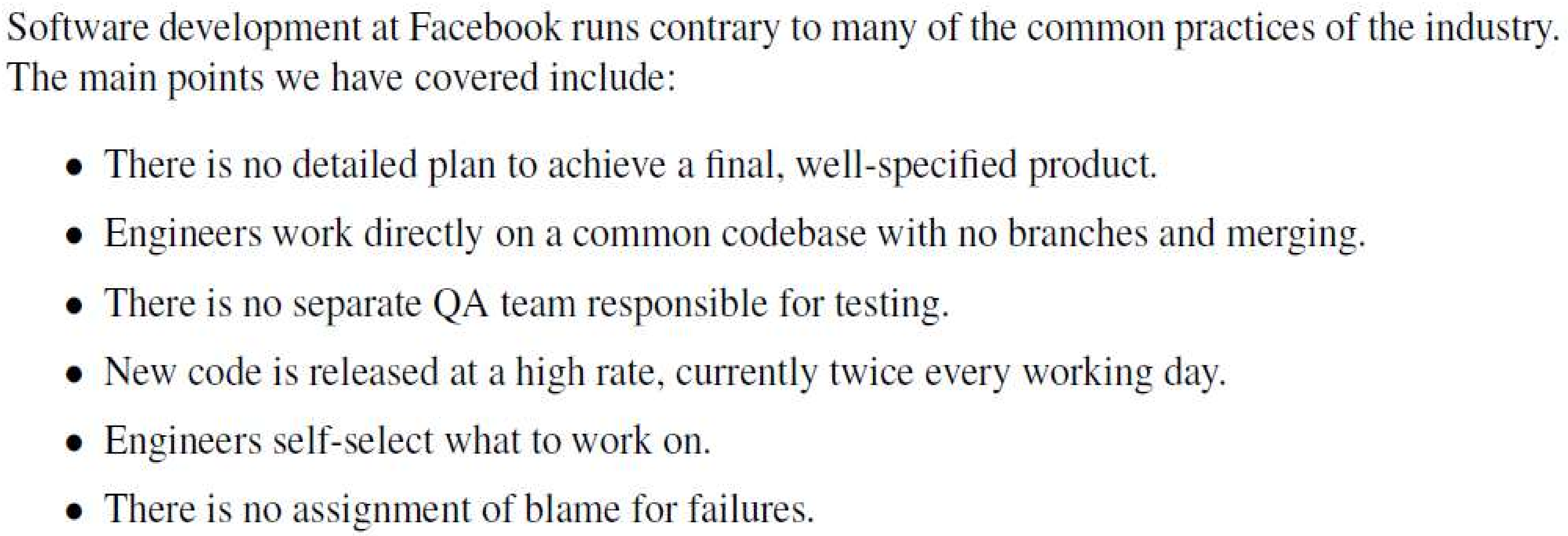
DevOps Case Study - IBM

- Consistency and Accuracy
- Automate manual tasks to improve consistency and accuracy while building your applications.

- Leverage data-driven insights
- Optimize your development and operations processes through collection of data at every step of the process.

- Unite your culture
- Transform your development culture by standardizing on solutions across your teams, bringing teams together and sharing data and insights to drive efficiencies and improvements.

DevOps Case Study - Facebook

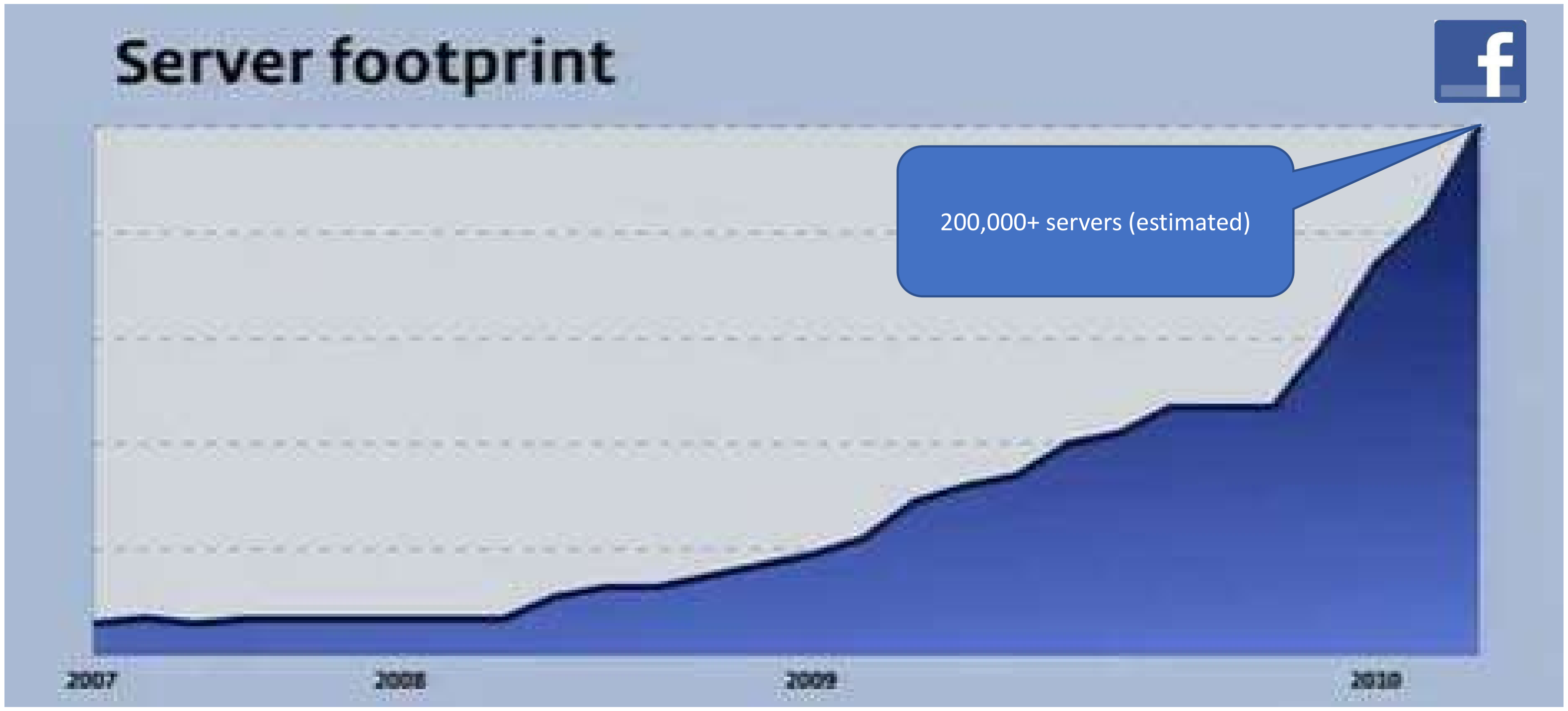


Software development at Facebook runs contrary to many of the common practices of the industry. The main points we have covered include:

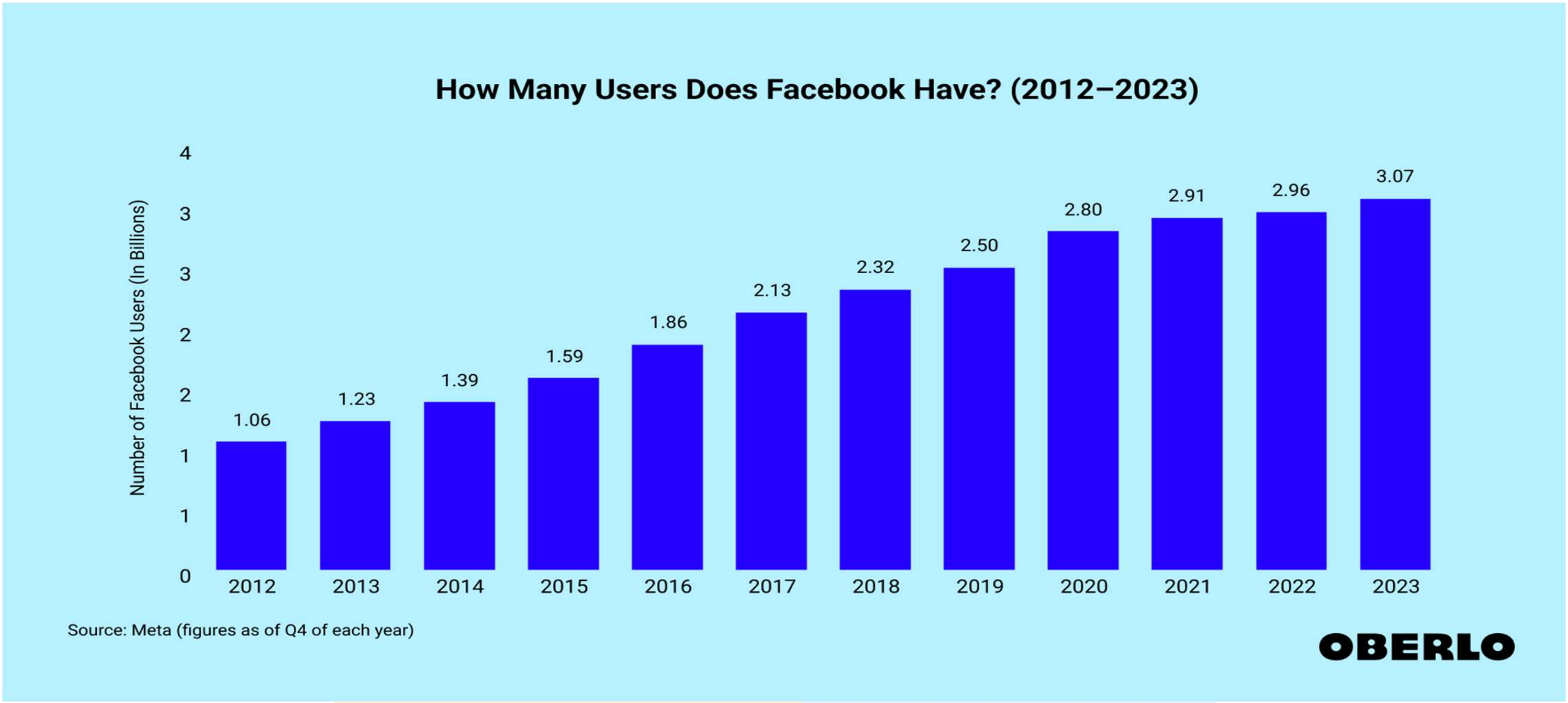
- There is no detailed plan to achieve a final, well-specified product.
- Engineers work directly on a common codebase with no branches and merging.
- There is no separate QA team responsible for testing.
- New code is released at a high rate, currently twice every working day.
- Engineers self-select what to work on.
- There is no assignment of blame for failures.

DevOps Case Study - Facebook

- In 2016, Gartner estimated that Google has 2.5 million servers
- In 2017, Microsoft Azure was reported to have more than 3 million servers.



DevOps Case Study - Facebook



DevOps Case Study - Facebook

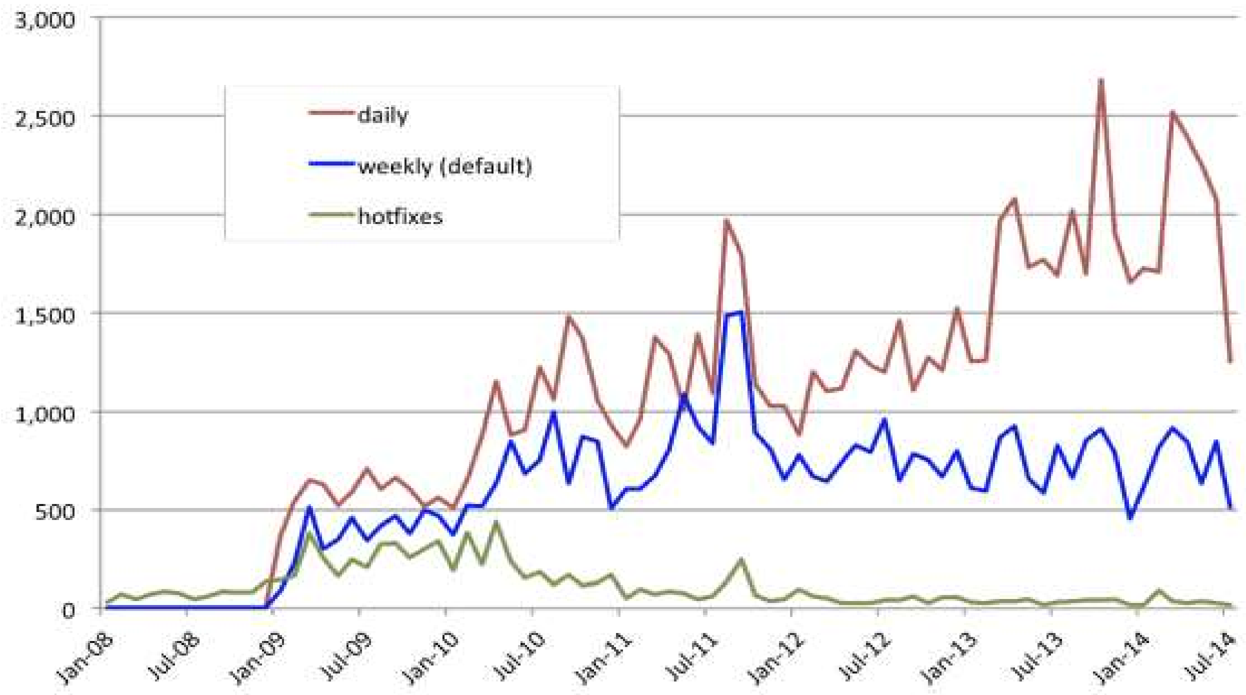


Figure 4: The number of deployment requests per week by type of request.

DevOps Case Study - Facebook

- Continuous Deployment is the process of taking software and deploying it to production quickly, safely, and repeatedly without impacting the development lifecycle.
- Here are a few features to have in a continuous deployment process:
- Fully automated deployments,
- Self-service deployments for developers
- An automated testing suite for generated software artifacts
- Frequent software artifact deployments with incremental development improvements

DevOps Case Study - Facebook

- Facebook now has more than 400 million active users, including more than 200 million who use the service every day.
- Here are some other data points from Cook's presentation and a talk last Thursday at Structure 2010 by Facebook's Jonathan Heiliger.
- Facebook currently has approximately 3.07 billion monthly active users worldwide.
- This makes it the most used social media platform globally.
- In Q2 2023, Meta reported over 3.8 billion users across its family of products

https://acodez.in/facebook-users-worldwide/

DevOps Case Study - Facebook

- Users spend more than 16 billion minutes on Facebook each day
- Every week users share more than 6 billion pieces of content, including status updates, photos and notes.
- Each month more than 3 billion photos are uploaded to Facebook.
- Users view more than 1 million photos every second
- Facebook's servers perform more than 50 million operations per second, primarily between the caching tier and web servers
- More than 1 million web sites have implemented features of Facebook Connect

DevOps Case Study - Facebook

Here's a more detailed look at Facebook's DevOps practices:

1. Code Ownership and Accountability:

•Facebook employs a "you build it, you own it" philosophy, where developers are responsible for the entire lifecycle of their code, from writing and testing to supporting it in production.

•This fosters a sense of ownership and accountability, encouraging developers to write high-quality, maintainable code.

•Code reviews are a crucial part of this process, ensuring that all code changes are thoroughly vetted before being deployed.

•The arc land --hold command allows developers to hold off on merging changes until they've received all necessary approvals and testing.
•This structured review process helps identify and fix potential issues before they make it into production.

DevOps Case Study - Facebook

2. Phabricator for Code Review and Collaboration:

•Facebook uses Phabricator, a code review and collaboration tool, to manage code changes.

•Developers submit their code changes for review using the arc diff --create command.

•The arc land --hold command allows developers to hold off on merging changes until they've received all necessary approvals and testing.

•This structured review process helps identify and fix potential issues before they make it into production.

•The `arc land --hold` command allows developers to hold off on merging changes until they've received all necessary approvals and testing.
•This structured review process helps identify and fix potential issues before they make it into production.

DevOps Case Study - Facebook

3. Gatekeeper for Deployment Control:

- Facebook uses Gatekeeper to manage deployments, separating deployment and release processes.

- This allows for more granular control over deployments, enabling them to release changes to specific groups of users or gradually roll them out.

- Gatekeeper also helps manage the complexities of deploying to both web and mobile platforms, which follow different release schedules.

•The `arc land --hold` command allows developers to hold off on merging changes until they've received all necessary approvals and testing.
•This structured review process helps identify and fix potential issues before they make it into production.

DevOps Case Study - Facebook

4. Tools and Technologies for Efficiency:

- Facebook has built its own tools and libraries to support rapid development and deployment, including:
  - Buck: A build system designed for large, multi-platform projects.
  - Phabricator: As mentioned, a code review and collaboration tool.
  - Infer: A static analysis tool for identifying potential issues in code.
  - React: A JavaScript library for building user interfaces.
  - Nuclide: An IDE for web and mobile development.

•The `arc land --hold` command allows developers to hold off on merging changes until they've received all necessary approvals and testing.
•This structured review process helps identify and fix potential issues before they make it into production.

DevOps Case Study - Facebook

5. Focus on Mobile Development:

- Facebook has developed specific tools and practices to address the unique challenges of mobile development, including:
  - Multi-arch builds: Building for different chip architectures (e.g., ARM, x86).
  - Static analysis using Infer: Identifying resource leaks, unused variables, and other issues.
  - Automated testing: Running unit, integration, and end-to-end tests.

•The `arc land --hold` command allows developers to hold off on merging changes until they've received all necessary approvals and testing.
•This structured review process helps identify and fix potential issues before they make it into production.

DevOps Case Study - Facebook

6. DevOps Culture and Philosophy:

- Facebook's approach to DevOps emphasizes collaboration, communication, and continuous improvement.
- The "you build it, you run it" philosophy fosters a sense of ownership and accountability, leading to faster issue resolution and higher quality code.
- By continuously iterating and improving their processes, Facebook is able to maintain its position as a leader in technology and innovation.

DevOps Case Study - Facebook

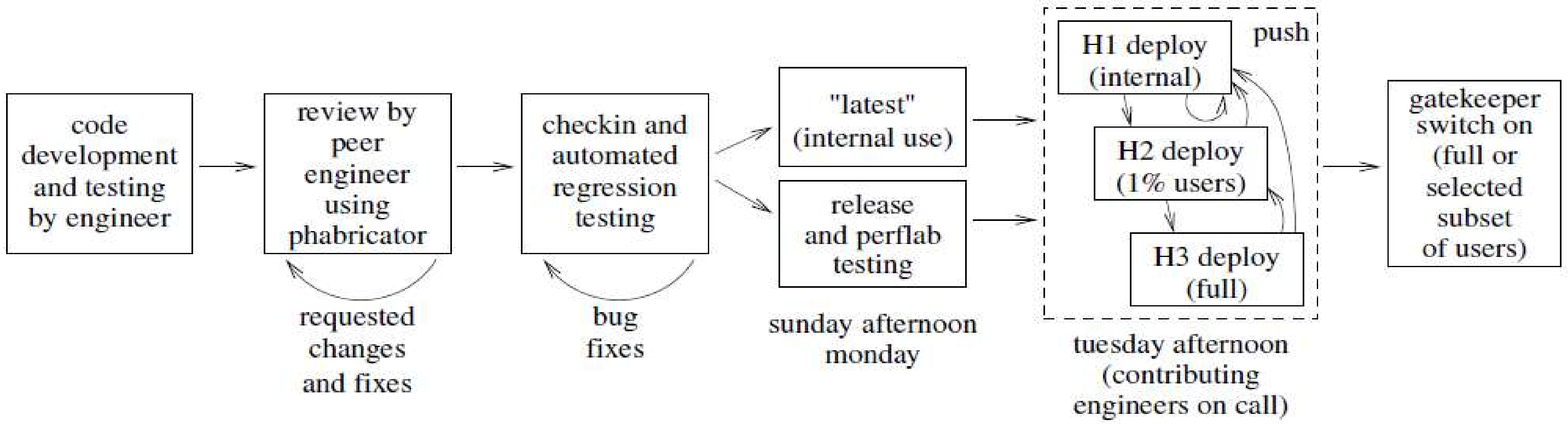

code development and testing by engineer
review by peer engineer using phabricator
requested changes and fixes
checkin and automated regression testing
bug fixes
"latest" (internal use)
release and perflab testing
sunday afternoon monday
push
H1 deploy (internal)
H2 deploy (1% users)
H3 deploy (full)
tuesday afternoon (contributing engineers on call)
gatekeeper switch on (full or selected subset of users)

Figure 7: *Facebook's version of the deployment pipeline, showing the multiple controls over new code.*

DevOps Case Study -Netflix

- DVD rental business, Netflix introduced its online streaming offering in 2007
- Handle this scale the company started moving to cloud providers in 2008, a process they finished in January 2016
- Netflix also point towards a certain level of uncertainty around predicted trends in traffic and uptake in new features
- Short peaks in traffic, where traffic goes up for a brief period of time but then returns to its normal rate, should also be handled

DevOps Case Study -Netflix

- Transformation from a DVD rental service to a global streaming giant.
- Key factors include its innovative subscription model, focus on personalized recommendations, and aggressive investment in original content.
- Netflix also adapted its strategy to changing market conditions and leveraged technology for global expansion.

DevOps Case Study -Netflix

- 1. From DVD Rentals to Streaming:

- Initial Model:
- Netflix started by offering DVD rentals by mail with no late fees, disrupting the traditional video rental market dominated by Blockbuster.

- Transition to Streaming:
- Netflix recognized the potential of streaming technology and shifted its focus to online video distribution, becoming a pioneer in the streaming revolution.

- Global Expansion:
- Netflix expanded its reach to over 190 countries, offering a mix of global hits and local content, catering to diverse audiences.

DevOps Case Study -Netflix

- 2. Key Strategies:
- Personalized Recommendations:
- Netflix developed a sophisticated recommendation system (Cinematch) that uses user-based ratings to predict viewing preferences, enhancing user engagement and satisfaction.
- Content Creation and Original Programming:
- Netflix invested heavily in creating original content, including popular series like "Stranger Things" and "The Crown," attracting a large subscriber base and setting itself apart from competitors.
- Data-Driven Approach:
- Netflix uses data analytics to understand viewer preferences, optimize content recommendations, and make informed decisions about content acquisition and production.
- Flexible Subscription Tiers:
- Netflix offers various subscription plans to cater to different budgets and viewing needs, providing flexibility and accessibility to a broad audience.

DevOps Case Study -Netflix

- 3. Success Factors:
- Innovation:
- Netflix consistently innovated its business model, technology, and content strategy, staying ahead of the curve in the competitive streaming market.
- Customer Focus:
- Netflix prioritized customer experience by offering a user-friendly platform, personalized recommendations, and a vast library of content.
- Adaptability:
- Netflix demonstrated the ability to adapt to changing market conditions, such as the rise of other streaming services and the need to produce more local content.
- Brand Building:
- Netflix established itself as a trusted brand known for its quality content, personalized service, and seamless viewing experience.

DevOps Case Study -Netflix

- 4. Challenges and Future Outlook:
- Intense Competition:
- Netflix faces increasing competition from other streaming services, including Disney+, Amazon Prime Video, and others.
- Content Costs:
- The cost of acquiring and producing high-quality content is rising, putting pressure on Netflix's profitability.
- Global Market Diversity:
- Netflix needs to continue adapting its content and marketing strategies to cater to diverse cultural and regional preferences.
- In conclusion, Netflix's case study showcases a remarkable journey from a DVD rental service to a global streaming giant, driven by innovation, customer focus, and a data-driven approach. While facing challenges in a competitive market, Netflix's ability to adapt and evolve will be crucial to its continued success.

DevOps Case Study -Netflix

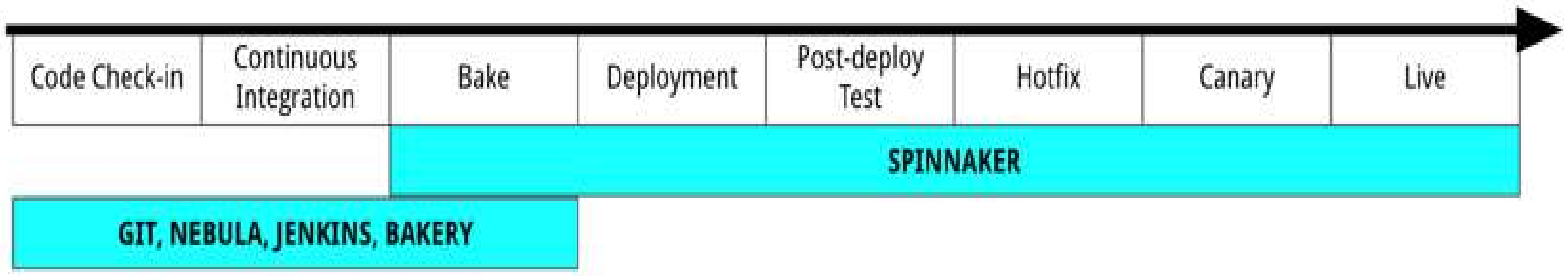
- Building with containers
  - Containerization is a method of abstracting away an applications runtime environment so you can run it consistently on different platforms.
  - In April 2017, Netflix surpassed one million containers launched a week.8 Scaling with cloud services and containerisation often go hand in hand and there are applications such as Kubernetes which help to automate this process.
  - Netflix have developed their own container management tool called Titus

DevOps Case Study -Netflix

- Building for failure
  - On Christmas Eve 2012 Netflix experienced a partial outage to their service that lasted a number of hours.
  - The cause of this was a fault with AWS.
  - In 2014 it was estimated that an hour of downtime would cost Netflix $200,000 .
  - More recent AWS outages have seen major websites taken offline.
  - They have a tool they call ‘Chaos Monkey’ which helps them to test the stability of their production applications.

DevOps Case Study -Netflix

- Creating a DevOps culture
  - positive DevOps culture should promote frequent releases, high automation and software reliability



https://netflixtechblog.com/how-we-build-code-at-netflix-c5d9bd727f15