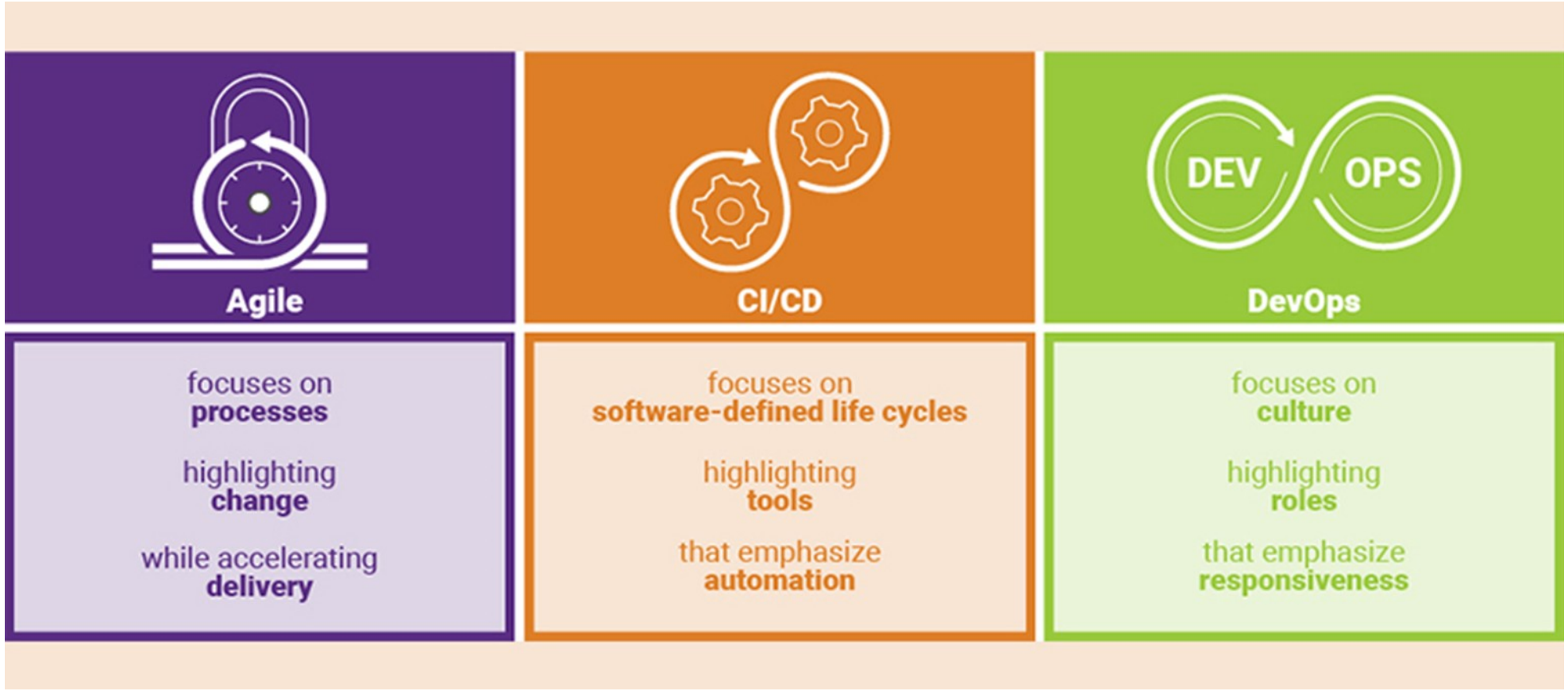
Introduction to CI/CD

Module 2: DevOps and Agile

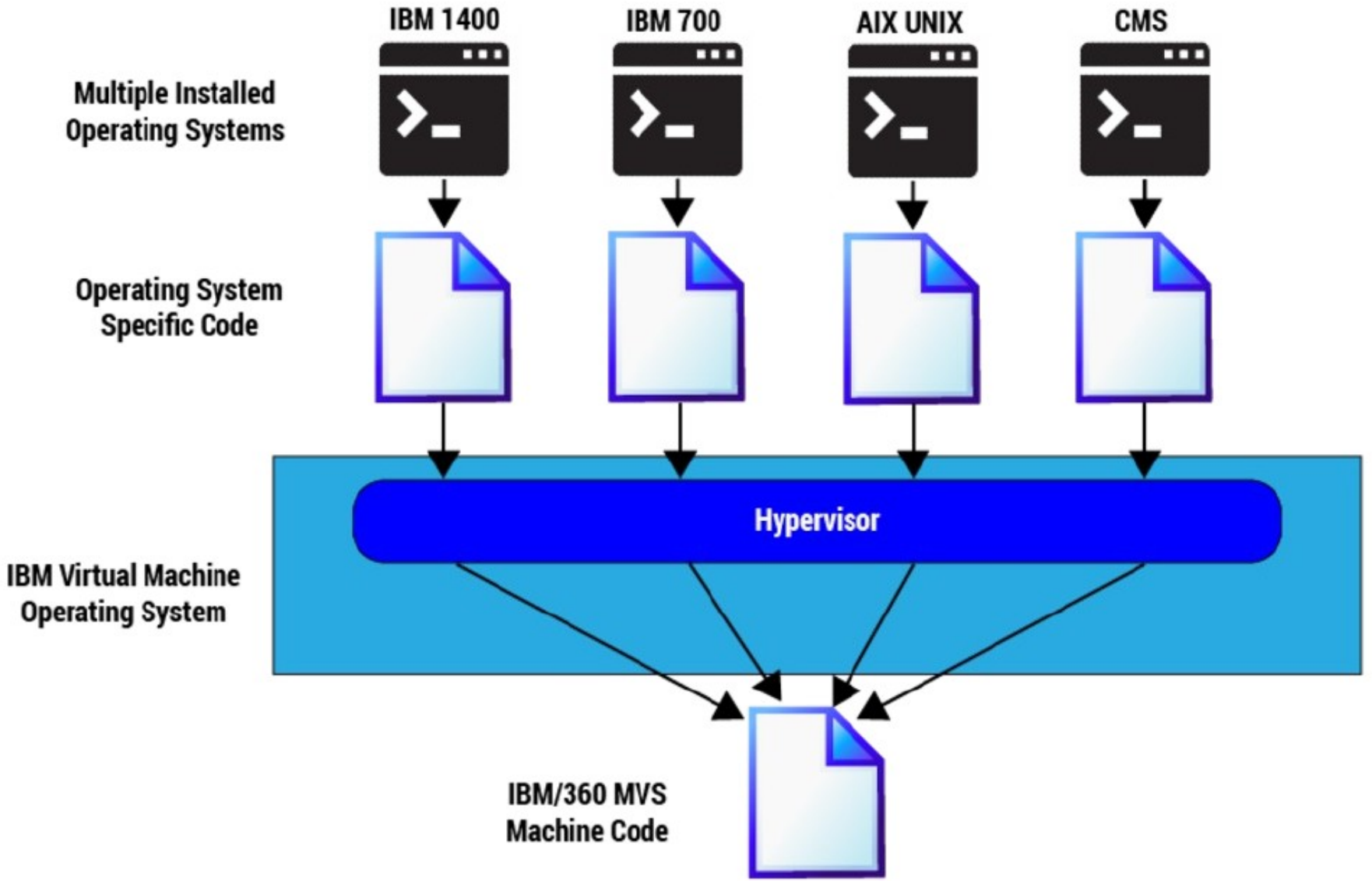
Agile – DevOps - CI/CD



Virtualization

- Created by IBM in 1972 - System VM/OS
  - Coined the term hypervisor
- IBM had a massive installed base of customers
  - Used different IBM mainframes, eg. 700 series, 1400 series
  - IBM wanted to retire these and move clients to the IBM/360
  - IBM only rented hardware to clients so there would be no hardware costs
- The problem
  - Each OS used software tightly coupled to the underlying hardware
  - Would entail massive software rewrites and so customers balked
  - The VM/OS could emulate all the legacy systems
  - Customer software from a IBM 1400 would run in the VM/OS emulating a 1400

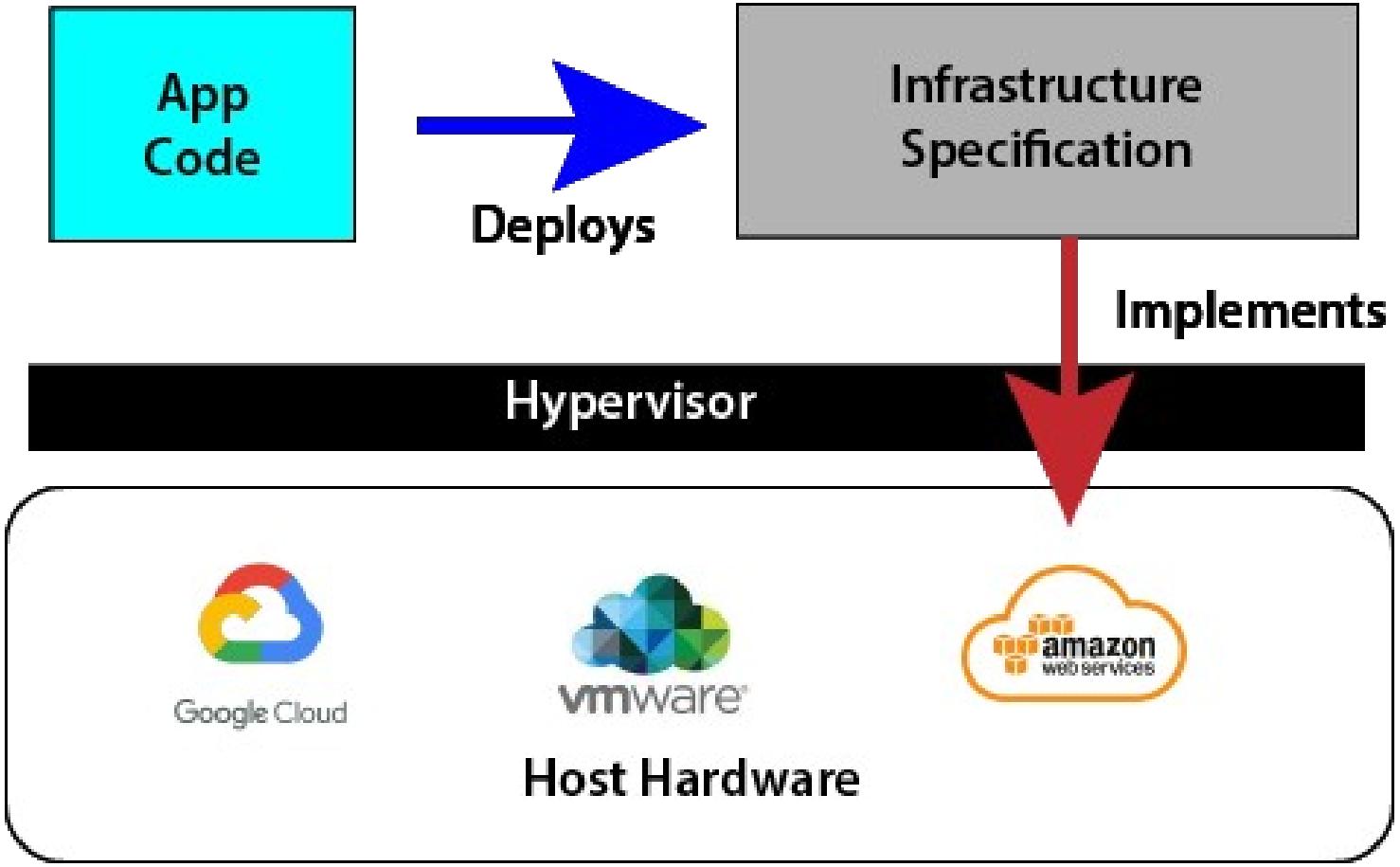
IBM VM/OS


IBM 1400
IBM 700
AIX UNIX
CMS
Multiple Installed Operating Systems
Operating System Specific Code
Hypervisor
IBM Virtual Machine Operating System
IBM/360 MVS Machine Code

Infrastructure as Code

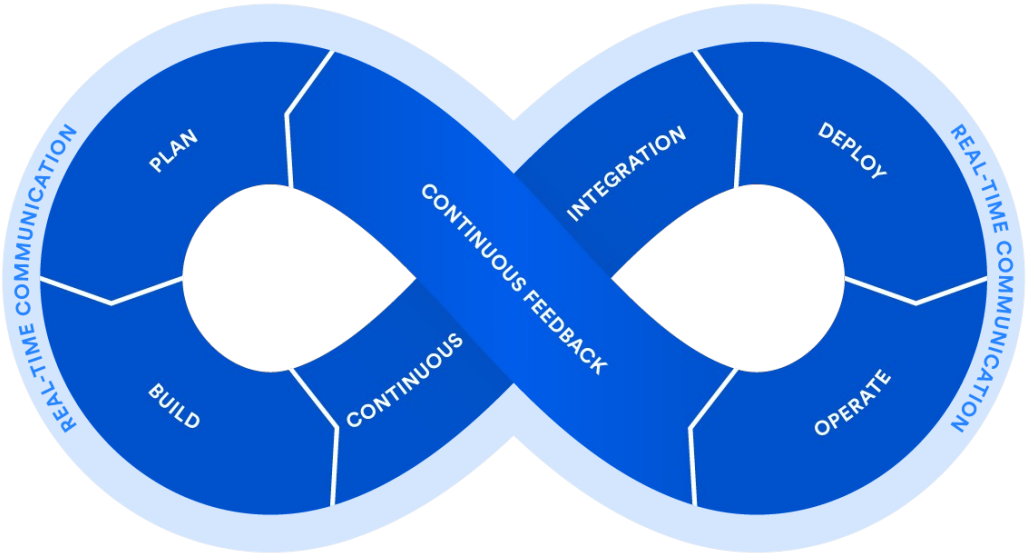
- Virtual machines directed the hypervisor to allocate hardware from the host systems
  - The host hardware was presented as virtual devices in the VM
  - Allowed AWS and others to implement VMs “in the cloud”
- This meant that provisioning an operational environment
  - Did not mean working with hardware directly
  - Instead, a specification or set of instructions to the hypervisor is written
  - The spec tells the hypervisor what virtual hardware to set up
  - The hypervisor does the hardware allocation
- Writing and executing this specification is s“infrastructure as code”

Infrastructure as Code



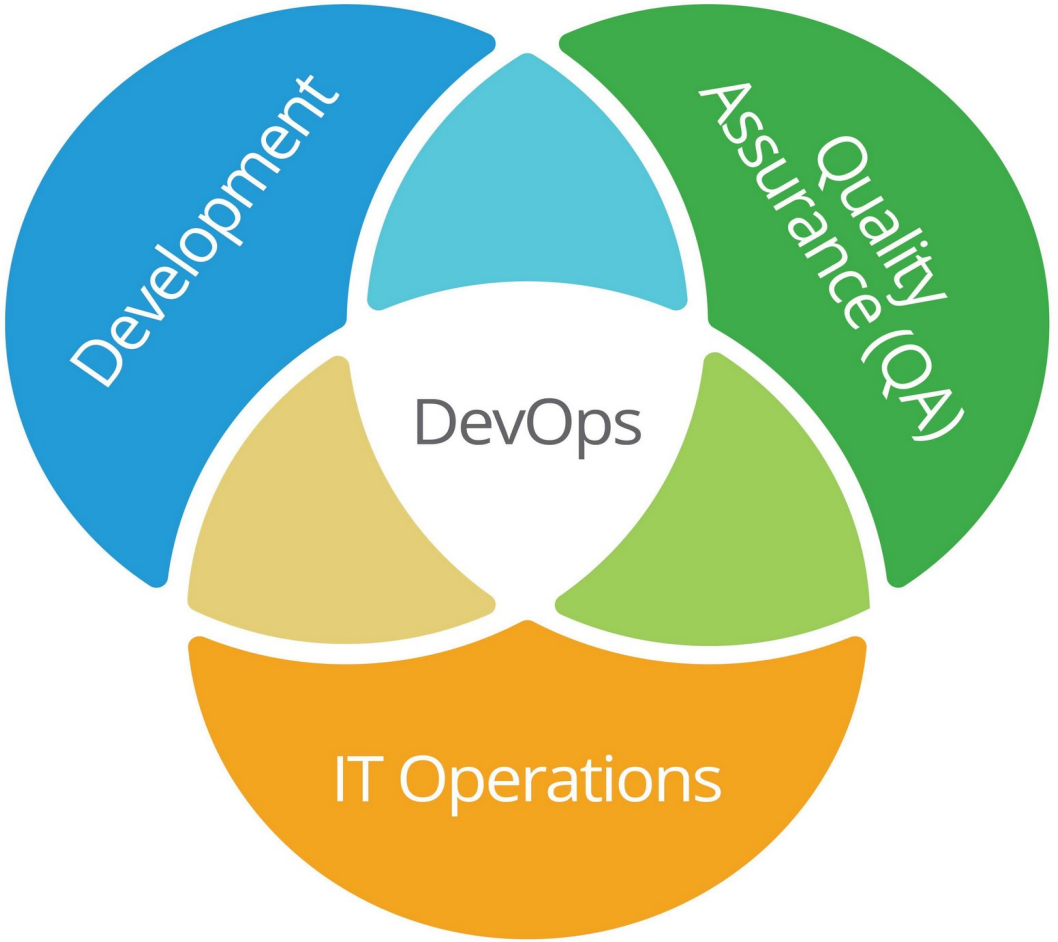
DevOps

- Driven by virtualization and Infrastructure as code
  - Dev and Ops had been two separate worlds
  - Dev was somewhat automated
  - Ops was manual and bare metal
- Virtualization turned it all into code
  - Now the same tools can be used in the entire life cycle of a software product
  - Opportunity for full process automation support
  - It allowed the integration of the product support phase with development



The Goal of DevOps

- De-siloize the three areas in software development
- To get everyone using the same sorts of tools, practices and automation
- Operations infrastructure is now code
  - The same processes are used to manage both application code and infrastructure code
- It also allows for integration of the processes in development with operations
  - Able to respond to real time feedback on how the applications running in the Ops environment



Defining CI/CD

- CI/CD is not a methodology
  - Continuous Integration Continuous Delivery
  - It is not Agile or DevOps, although both rely on and use it extensively
- CI/CD is a process automation applied to SE
  - Not a development or process methodology
  - Similar to other kinds of automation
  - Improves process efficiency and effectiveness
- CI/CD is process agnostic
  - Can be used anywhere a SE process is well defined
  - Using CI/CD with bad processes makes them worse

A fool with a tool is still a fool

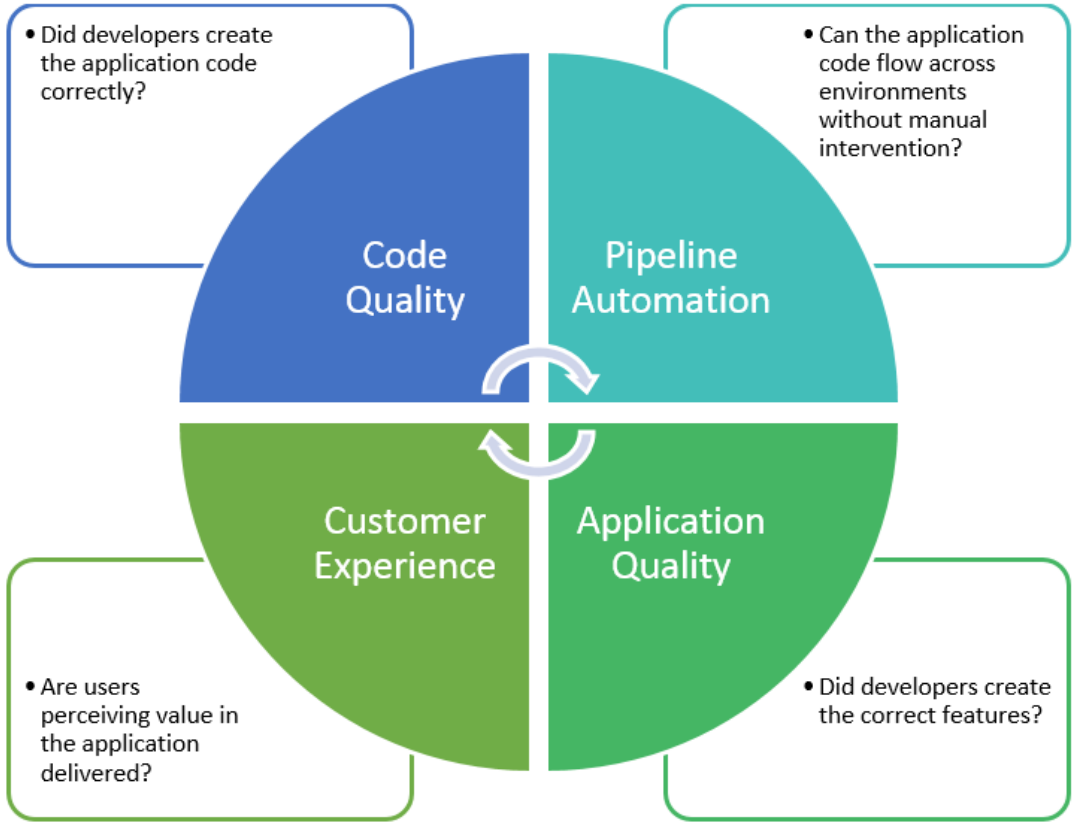
Martin Fowler

A computer lets you make more mistakes faster than any invention in human history – with the possible exceptions of handguns and tequila

Mitch Ratcliffe

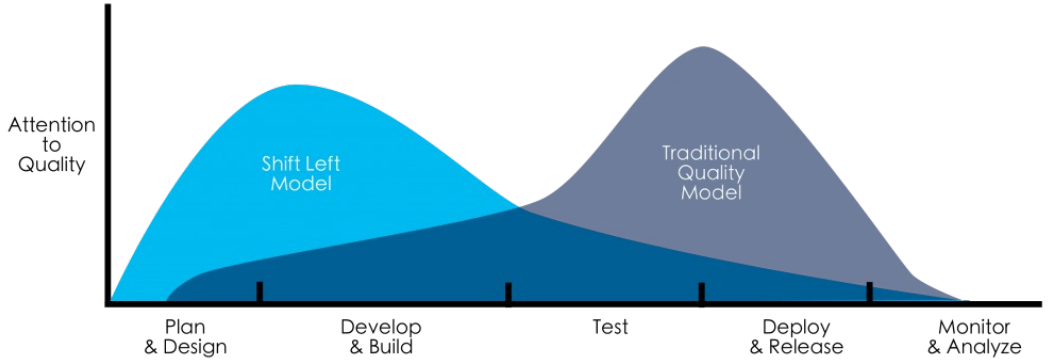
Continuous Testing

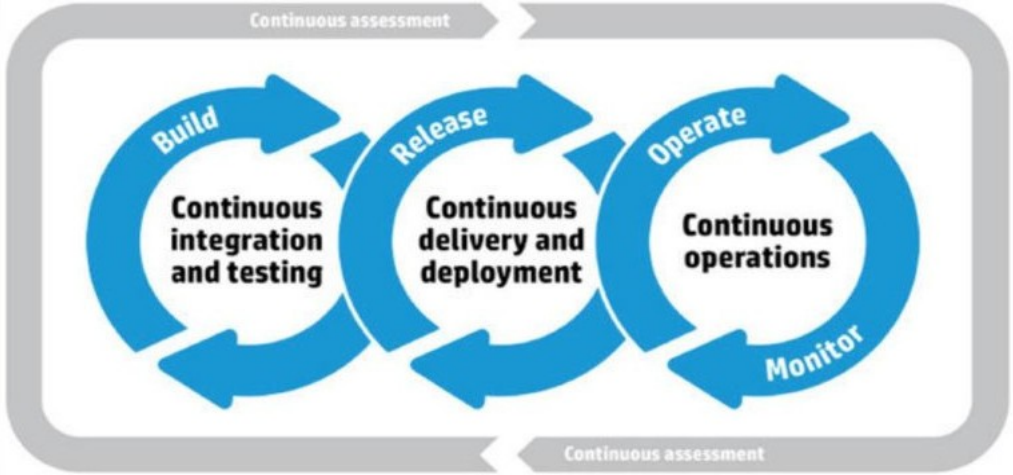
- Does not replace human based testing
  - Like pair programming and code reviews
- Testing is now build into the entire development and operations cycle
- Creates “quality gates”
  - Development pipelines abort when tests fail
  - Identifies problems early in production so they can be remediated early
  - Adding continuous security testing and security planning is called DevSecOps



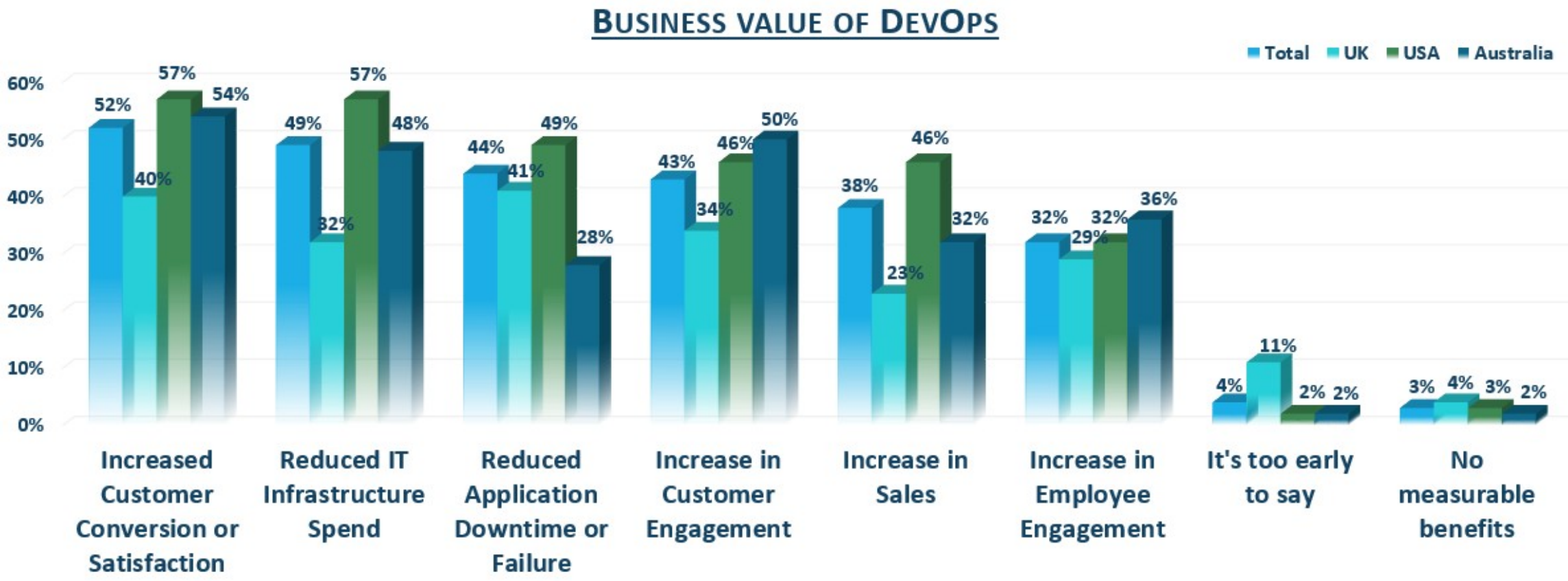
Continuous Testing

- Continuous Testing
  - Every artifact is tested as it is created
  - Shift Left Model
  - Test early, test often
- CI/CD also adds
  - Automated testing at every stage
- CT is triggered by events in the CI/CD process
  - Checking in code => automated unit testing
  - Build => integration testing

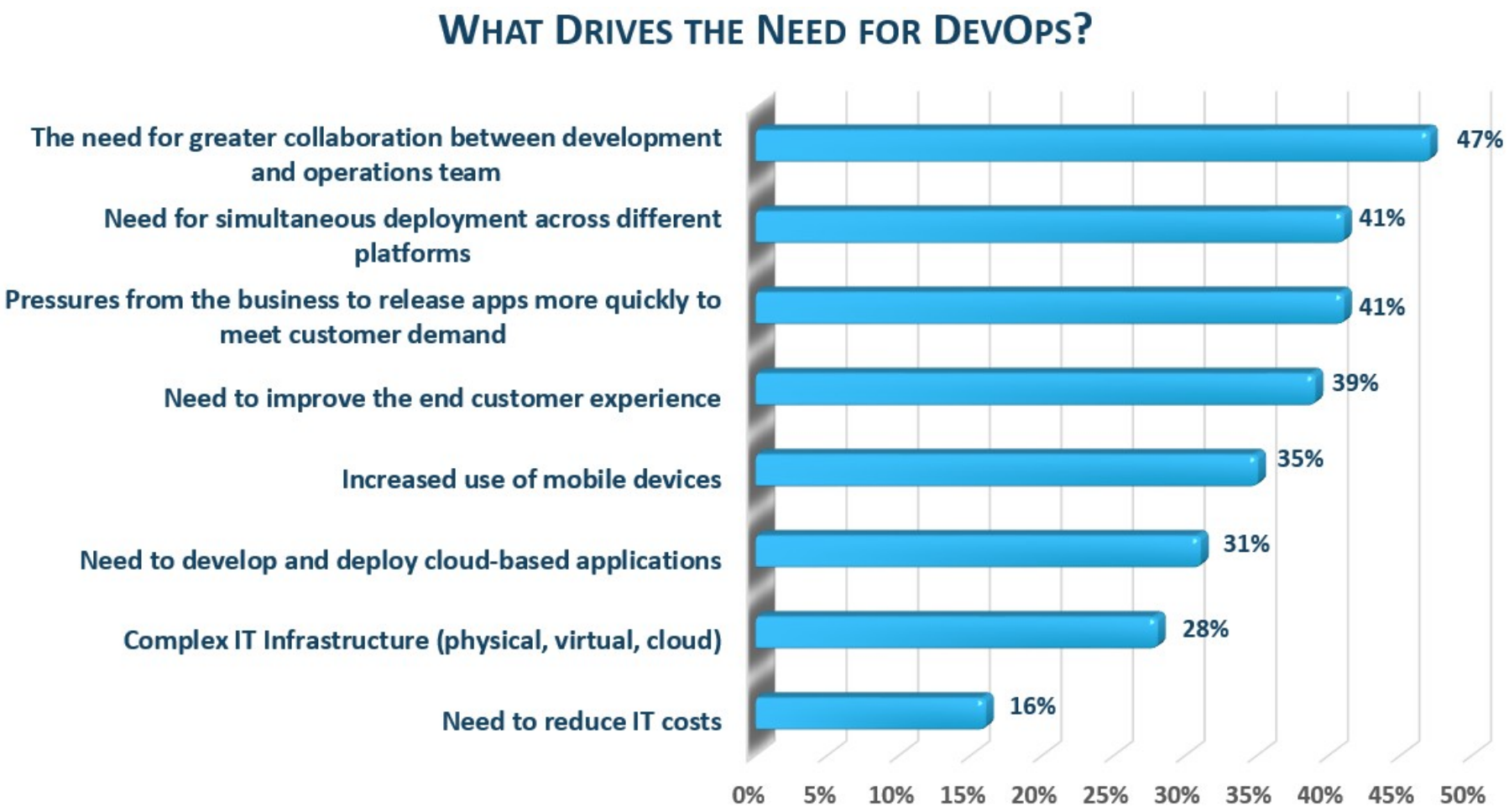




DevOps Cost Benefit



DevOps Cost Benefit



DevOps Cost Benefit

Functions	Previous Time Frame	Present Time Frame	DevOps Benefit
Project initiation	10 days	2 days	80% faster
Overall time to development	55 days	3 days	94% faster
Build verification test availability	18 hours	< 1 hour	94% faster
Overall time to production	3 days	2 days	33% faster
Time between releases	12 months	3 months	75% faster

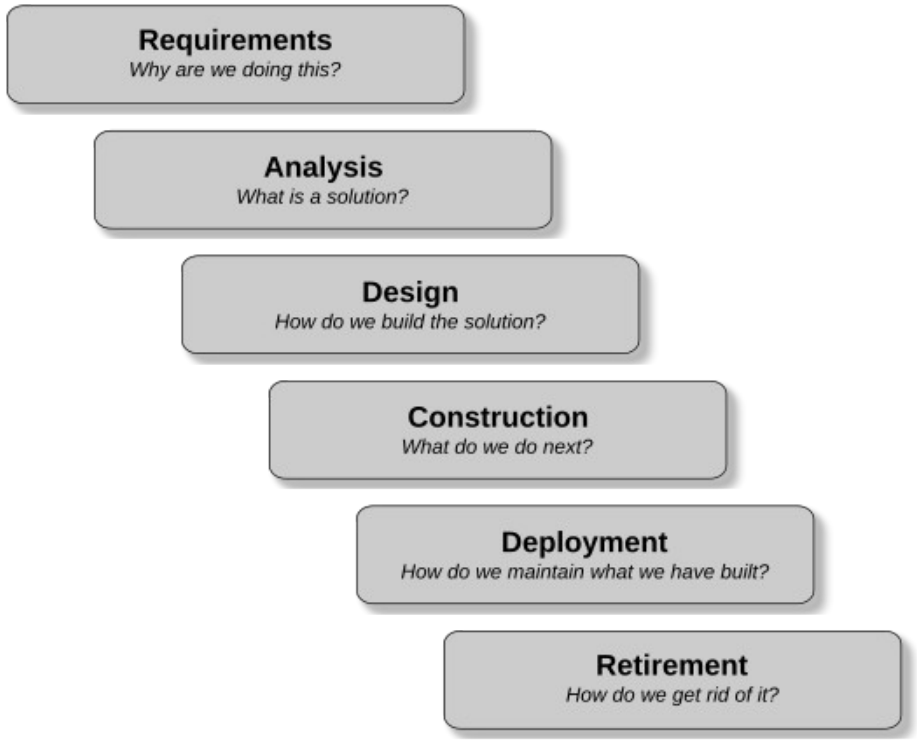
DevOps, clearly an extension of lean and agile principles, was as much, in IBM, born of necessity to respond to a pervasive industry mandate to "do more with less" and has evolved to "quality software faster."

- Kristof Kloeckner,
General Manager,
IBM Software Group – Rational

https://www.compaid.co.in/Articles/DevOps-Value-and-Cost-Savings

The Engineering Cycle

- The Engineering Cycle: a set of logical steps
  - Each step builds on the previous one
  - How we apply this cycle is a “process type”
- Waterfall process types
  - Completes each step in the process fully before moving on to the next step
  - Also called a “predictive” process because given the full set of requirements and technical constraints, we can accurately predict the final product will be
  - Common in engineering systems and high risk systems
  - Like nuclear reactor control software or airline navigation software
  - Or where requirements and technology don’t change over the lifetime of the project



In Real Life

- For a lot of applications, the waterfall doesn’t work
  - There is often too much uncertainty and variation at each stage of the engineering cycle
  - Results in a lot of re-work as we respond to unplanned for change
- These problems are not just software related
  - They were common across a variety of industries
  - Various alternatives to the waterfall approach were being experimented with
- Collectively, these are referred to as adaptive methodologies
  - The continuously adapt to uncertainty and variations
  - Essentially incorporating risk management into a production process
  - Most notable of these is Scrum

Scrum

- Scrum was not originally intended for software development
  - It was originally developed for industrial manufacturing in the 1950s-1980s
  - Very influential: Lean manufacturing & Toyota production system
- In the 1980s, there was a major crisis in software development
  - It was being developed in a big-bang approach
  - Siloed teams (design, development, testing) with one-time hand offs of artifacts
- NATO software engineering conference in 1968 was held to address the high failure rate of these big-bang projects
  - The conclusion of this and other similar conferences was that in incremental and iterative approach, like Kaizen (continuous improvement) and lean approaches to product development were needed

Scrum

- The term Scrum is first used in 1986 with reference to new product development
  - Hirotaka Takeuchi and Ikujiro Nonaka “The New New Product Development Game”
- Jeff Sutherland & Ken Schwaber (1997)
  - Presented the first public paper on Scrum: "SCRUM Development Process."
  - Emphasized:
    - *Empirical process control theory (transparency, inspection, adaptation)*
    - *Lean principles*
    - *Nonaka & Takeuchi’s knowledge-creation theory (SECI) – the basis for cross functional teams*
- The basic ideas expressed in Scrum started to be adopted generally
  - Primarily by teams working with volatile types of projects
  - Often smaller teams of developers working closely with the business side

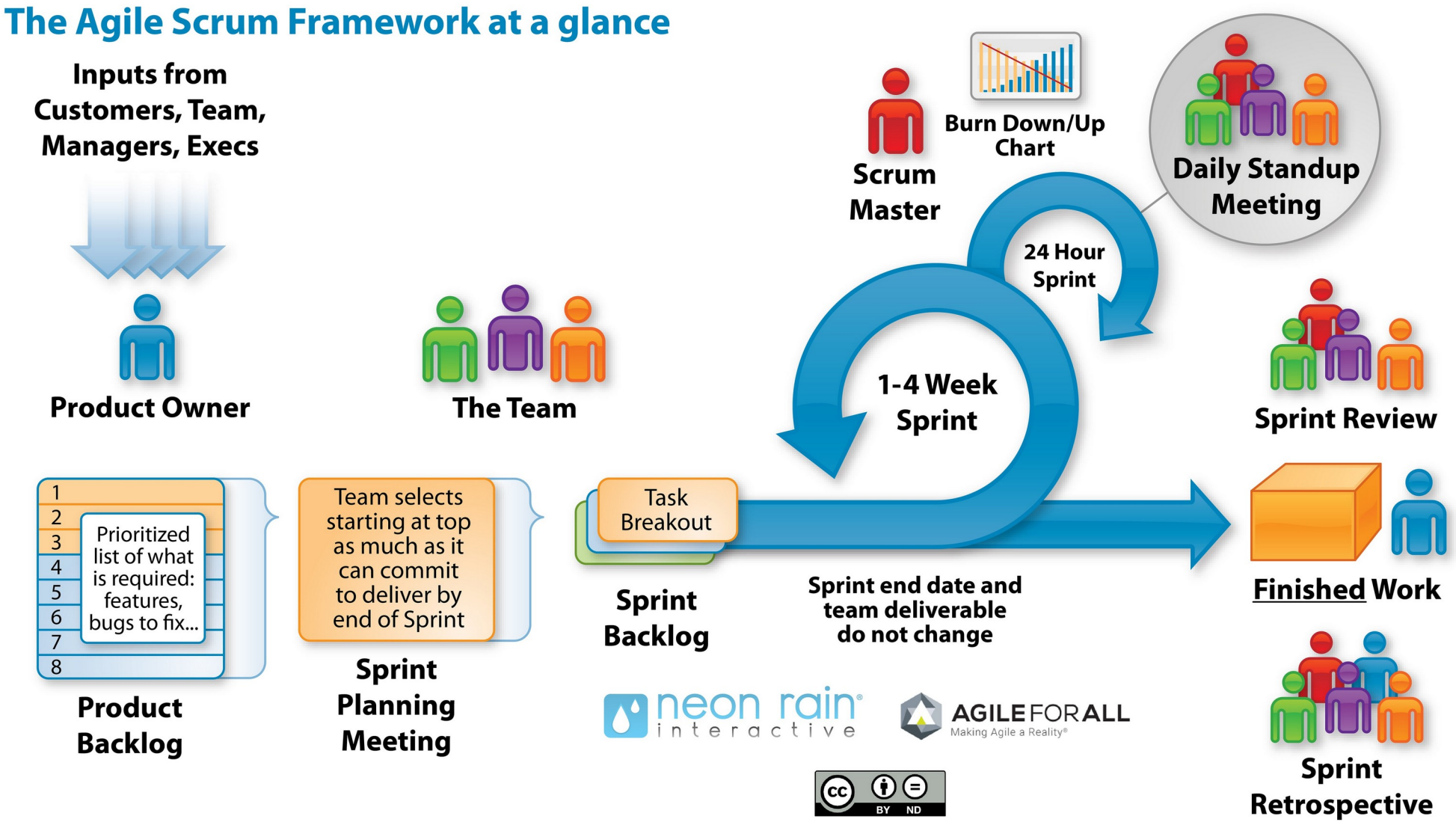
Scrum

- In the 1980s and 1990s
  - Companies experimented with using what they called adaptive development
  - IBM, DuPont, and others experimented with iterative prototyping and empirical process control
- Characterized by
  - Use of successive prototypes to get feedback on requirements, design and performance
  - Short iterations (one month or less)
  - Cross-functional teams
  - Daily meetings for synchronization
  - A prioritized feature list

Frameworks and Methodologies

- A common mistake is to call Scrum an Agile methodology
  - It is not a methodology, it is a process framework
  - Failure to have a methodology defined while using Scrum negates the value of using it
- Scrum is process-focused - managing what and when
- Agile methodologies are practice-focused
  - Managing how the work is done
- The two complement each other
  - Scrum without a software engineering methodology risks low quality
  - A software engineering methodology without Scrum risks lack of direction and prioritization

Scrum at a Glance



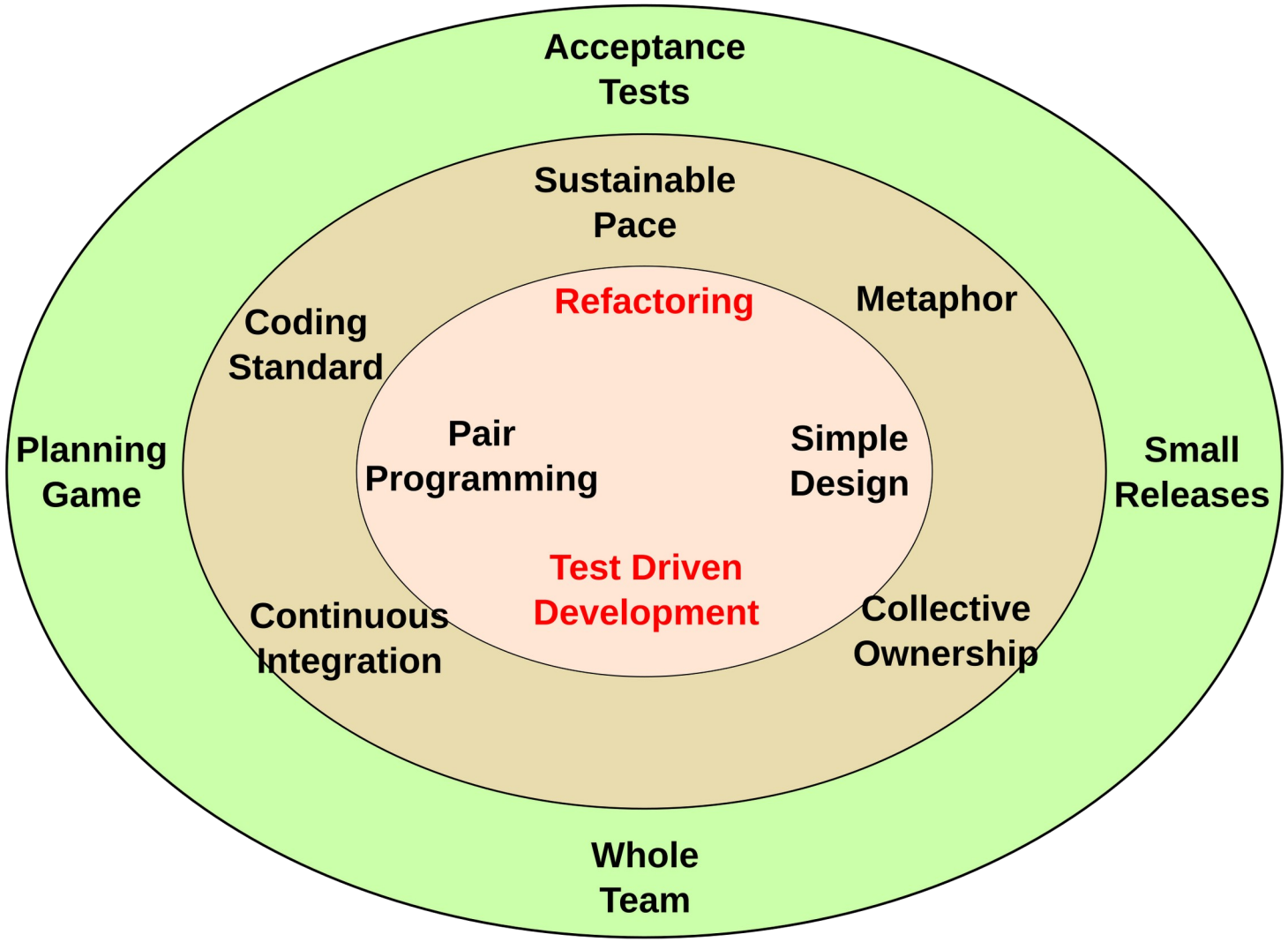
Scrum and Agile

- Agile was the general term adopted in 2001
  - Defined by owners of adaptive methodologies that shared a similar approach to development
  - Derived many of their ideas from their use of Scrum ideas
    - *For example: Extreme Programming, Feature Driven Development*
- Many of the features of Scrum were incorporated into the Agile Manifesto and the Agile Principles
  - Note that Scrum is NOT an Agile methodology, but its concepts were shared among Agile methodologies
  - Specifically:
    - *Individuals and interactions*
    - *Working software prototypes*
    - *Customer collaboration and feedback loops*
    - *Responding to change in a planned systematic way*

Extreme Programming (XP)

- Developed in the late 1990s by Kent Beck
  - One of the original Agile methodologies
  - Designed to improve software quality and responsiveness to changing customer requirements through frequent releases, continuous feedback, and disciplined technical practices
- Core ideas
  - XP focuses on adaptability, collaboration, and technical excellence
  - Pushes core agile principles to the “extreme”
  - Ensuring that teams can deliver value rapidly and sustainably even in high-uncertainty environments
  - Very dependent on automation, typical of most Agile methodologies

Extreme Programming (XP)



Extreme Programming (XP)

- Key characteristics:
  - Highly iterative and incremental: Frequent releases and feedback loops
  - Human-centered: Collaboration and trust are central
  - Quality-Driven: Testing, integration, and refactoring prevent defects
  - Adaptable: Embraces change as part of the process, not a disruption
- But to meet these goals, automation is essential
  - Automated unit testing
  - Automated build management
  - Automated deployments
  - Automated feedback from prototypes

Drivers of CI/CD

- Need to develop and deploy large numbers of microservice components
  - These all need to done via an automated development pipeline
  - Traditional app development doesn’t get the job done
  - Need to build in quality control and testing
- Need to automate a number of phases of Agile development
  - Working prototypes need be regularly produced
  - Continuous testing during development
- Infrastructure as code
  - Code is specified in code
  - The same sort of development requirements as above now apply to operations and IaC

Benefits of CI/CD

- Smaller code changes are simpler (more atomic) and have fewer unintended consequences
- Fault isolation is simpler and quicker
- Mean time to resolution (MTTR) is shorter because of the smaller code changes and quicker fault isolation
- Testability improves due to smaller, specific changes
  - These smaller changes allow more accurate positive and negative tests
- Elapsed time to detect and correct production issues is shorter with a faster rate of release
- The backlog of non-critical defects is lower because defects are often fixed before other feature pressures arise
- The product improves rapidly through fast feature introduction and fast turn-around on feature changes

Benefits of CI/CD

- CI/CD product feature velocity is high
  - The high velocity improves the time spent investigating and patching defects
- Feature toggles and blue-green deploys enable seamless, targeted introduction of new production features
- Upgrades introduce smaller units of change and are less disruptive
- You can introduce critical changes during non-critical (regional) hours
  - This non-critical hour change introduction limits the potential impact of a deployment problem
- Release cycles are shorter with targeted releases and this blocks fewer features that aren’t ready for release
- End-user involvement and feedback during continuous development leads to usability improvements
  - You can add new requirements based on customer’s needs on a daily basis

Challenges for CI/CD

- Organization silos and corporate culture
  - Lack of communication between development, QA and operations
- Failure to automate testing or do continuous testing
  - QA starts lagging behind development requiring rework to fix buggy code
- Legacy systems integration
  - Automated tools may not be available for legacy systems
  - E.g. Unit testing frameworks for COBOL code
- Complexity and size of applications
  - Trying to apply CICD to too big a “chunk” of development
  - Especially when introducing CICD improvements

Questions