CICD with Jenkins

Module 1: CICD

Defining CICD

- CICD is not a methodology
  - It is not Agile or DevOps
  - Although both rely on CICD and use it extensively
- CICD is process automation applied to SE
  - It is not Agile or DevOps
  - Similar to other kinds of automation
  - Improves process efficiency and effectiveness
  - CICD is process agnostic
  - Can be used anywhere a SE process is well defined
- Using CICD with bad processes makes them worse

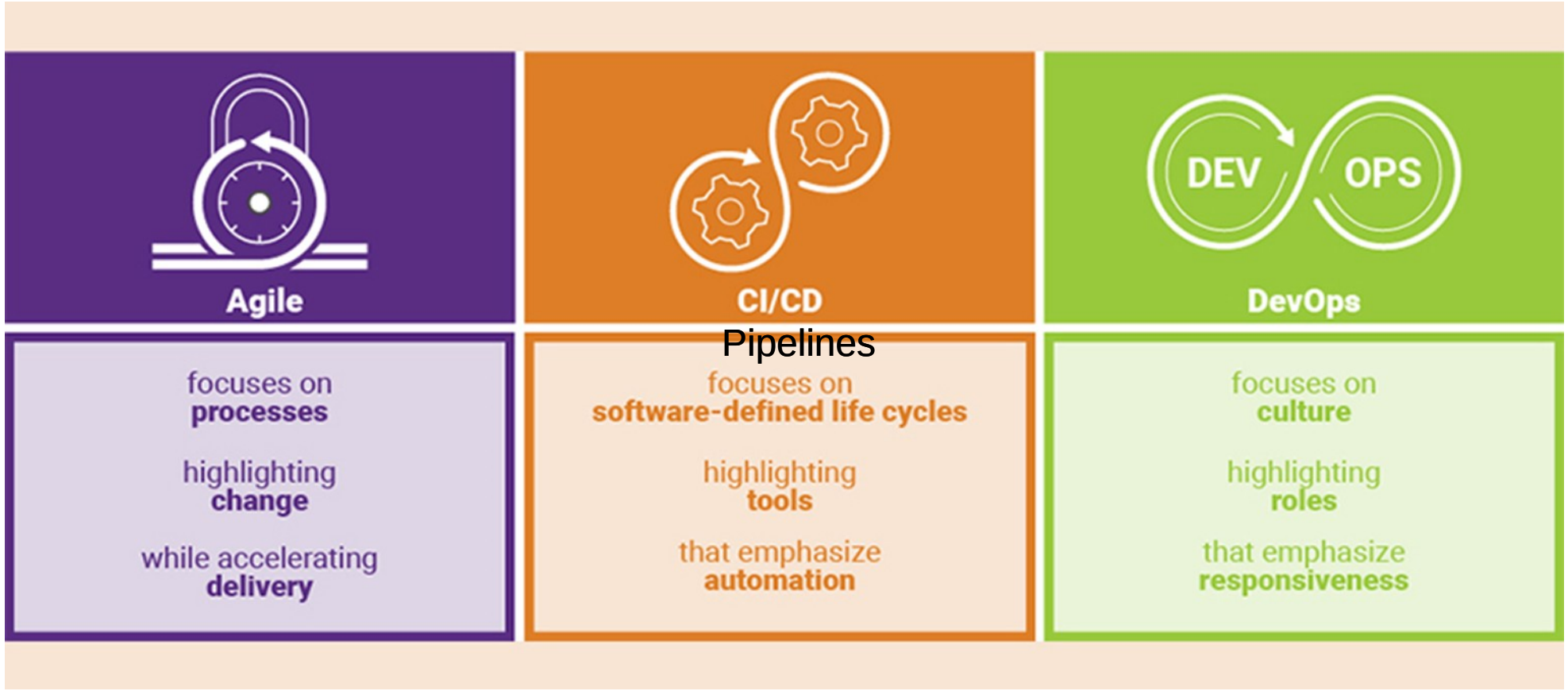
A fool with a tool is still a fool

Martin Fowler

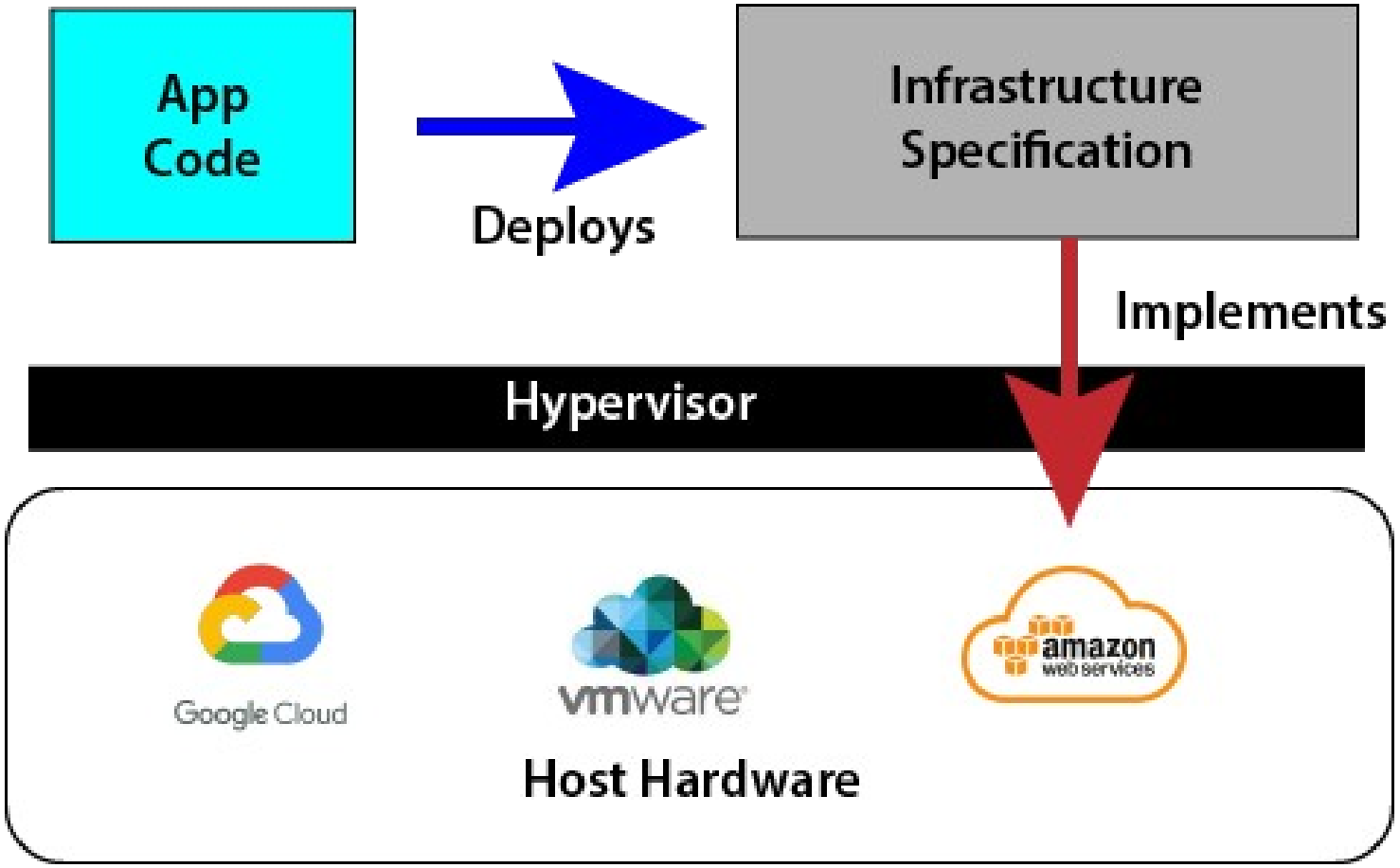
A computer lets you make more mistakes faster than any invention in human history – with the possible exceptions of handguns and tequila

Mitch Ratcliffe

Agile, DevOps and CICD

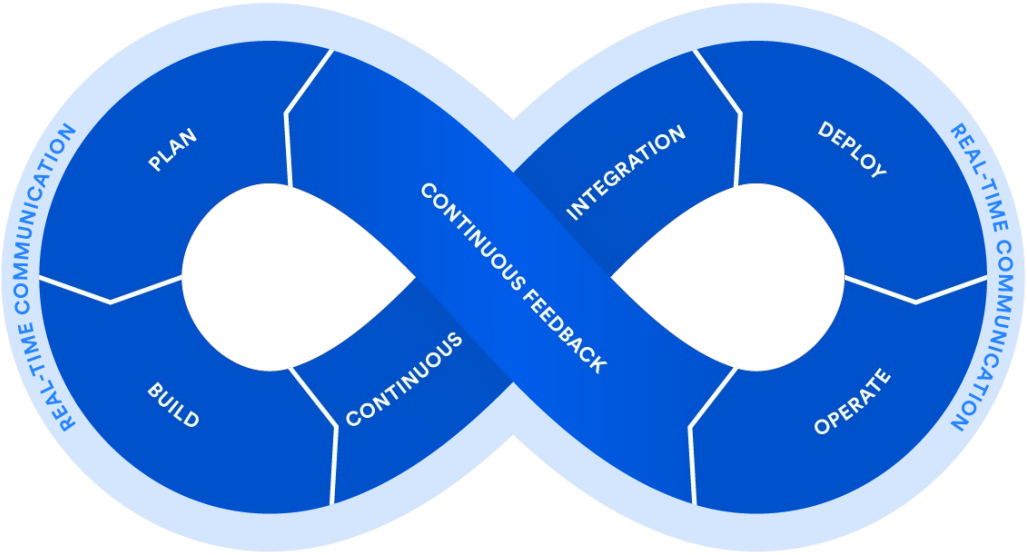


Infrastructure as Code



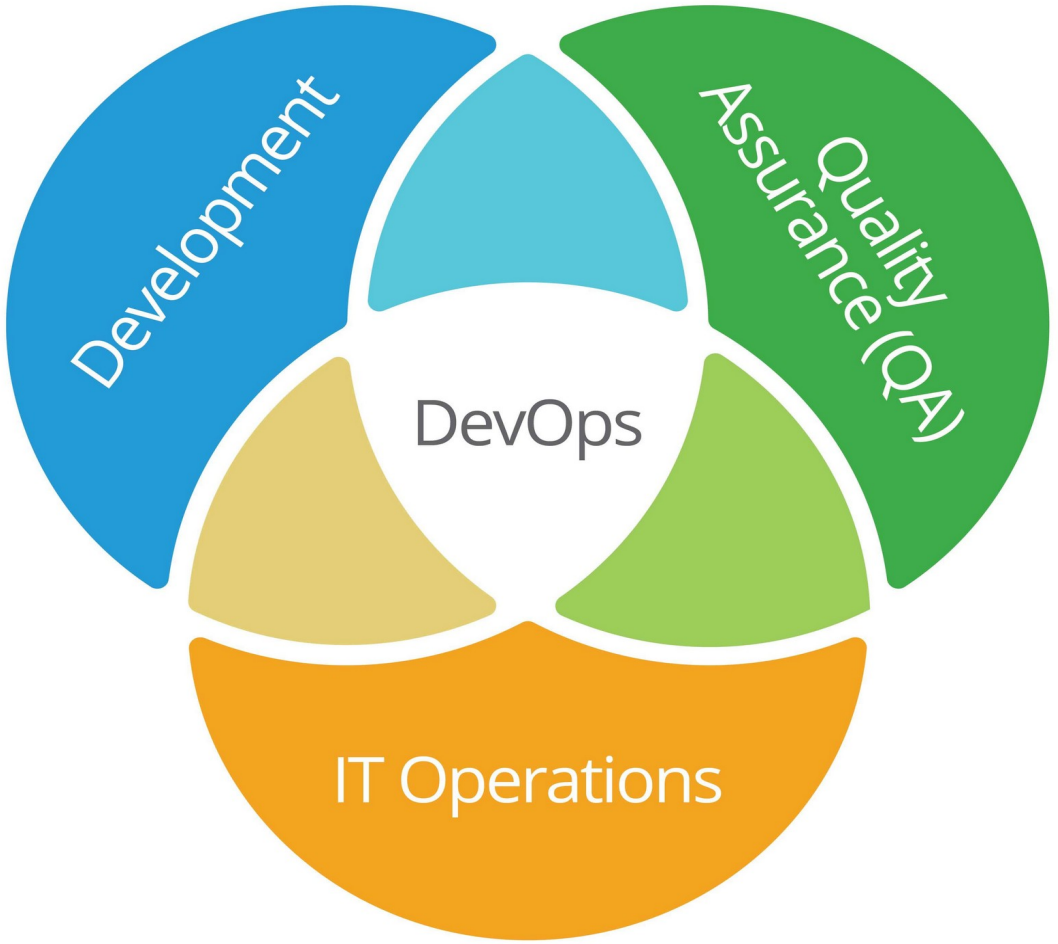
DevOps

- Driven by virtualization and Infrastructure as code
- Dev and Ops had been two separate worlds
  - Dev was sort of 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



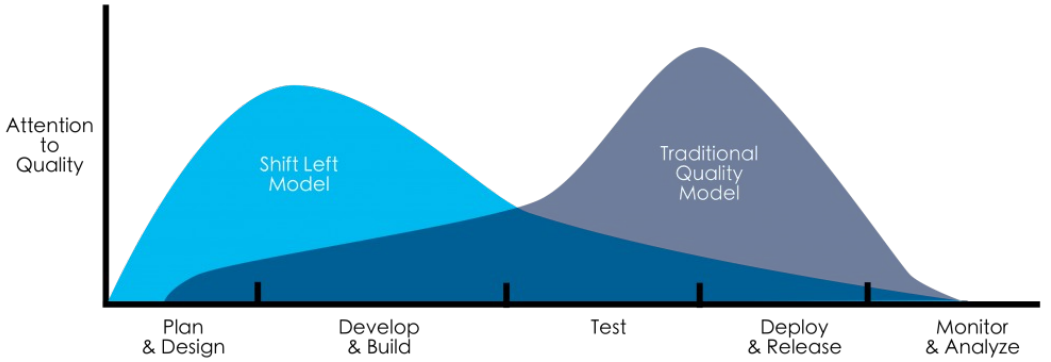
The Goal of DevOps

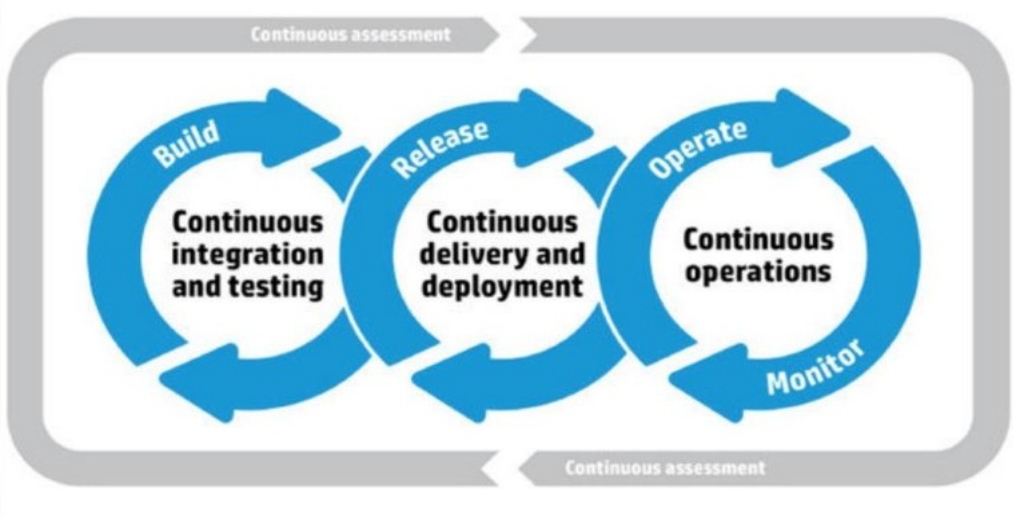
- Desilo-ize the three areas in software development
- Get everyone using the same sorts of tools, practices and automation
- CICD
  - **Continuous Integration**: continuous integration of multiple development activities
  - **Continuous Delivery**: build artifact made available for delivery
  - **Continuous Deployment**: Delivered artifact is also pushed into operational environment



The Goal of DevOps

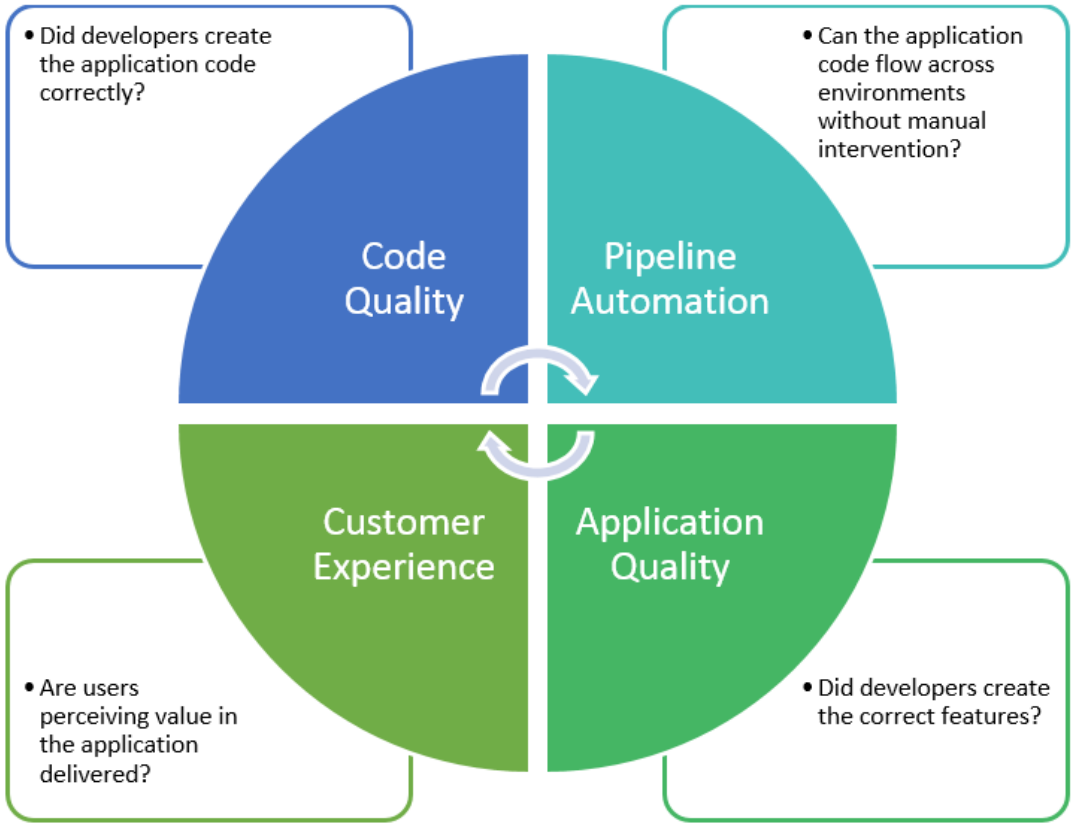
- Continuous Testing
  - Every artifact is tested as it is created
- Shift Left Model
  - Test early, test often
- CICD also adds
  - Automated testing at every stage
- CT is triggered by events in the CICD process
  - Checking in code => automated unit testing
  - Build => integration testing





Continuous Testing

- Does not replace human based testing
  - Like pair programming and code reviews
- Creates “quality gates”
  - Development pipelines abort when tests fail
- Adding continuous security testing and security planning is called DevSecOps

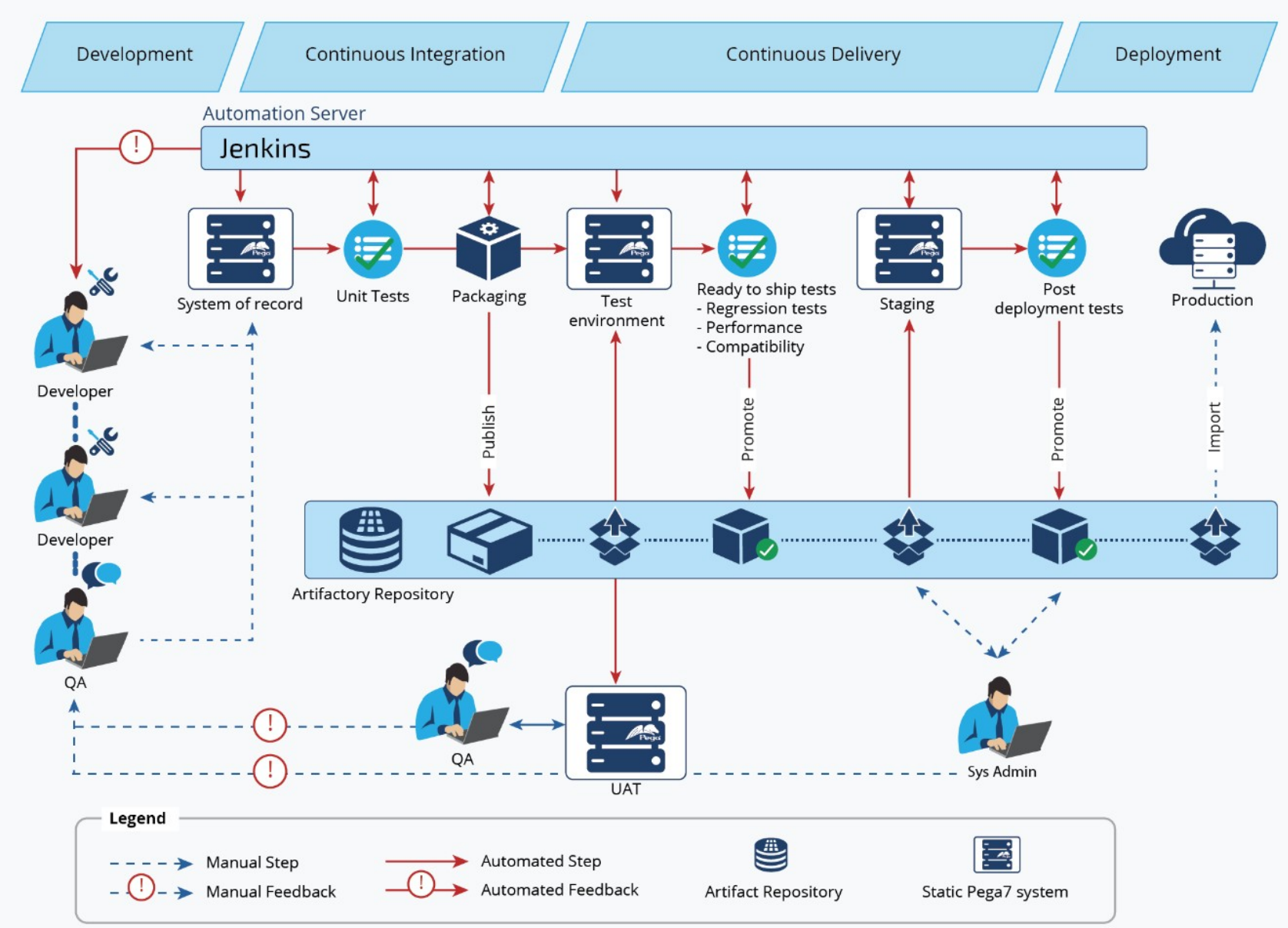


More than the act of testing, the act of designing tests is one of the best bug preventers known. The thinking that must be done to create a useful test can discover and eliminate bugs at every stage in the creation of software, from conception to specification, to design, coding and the rest.

If you can't test it, don't build it.
If you don't test it, rip it out.

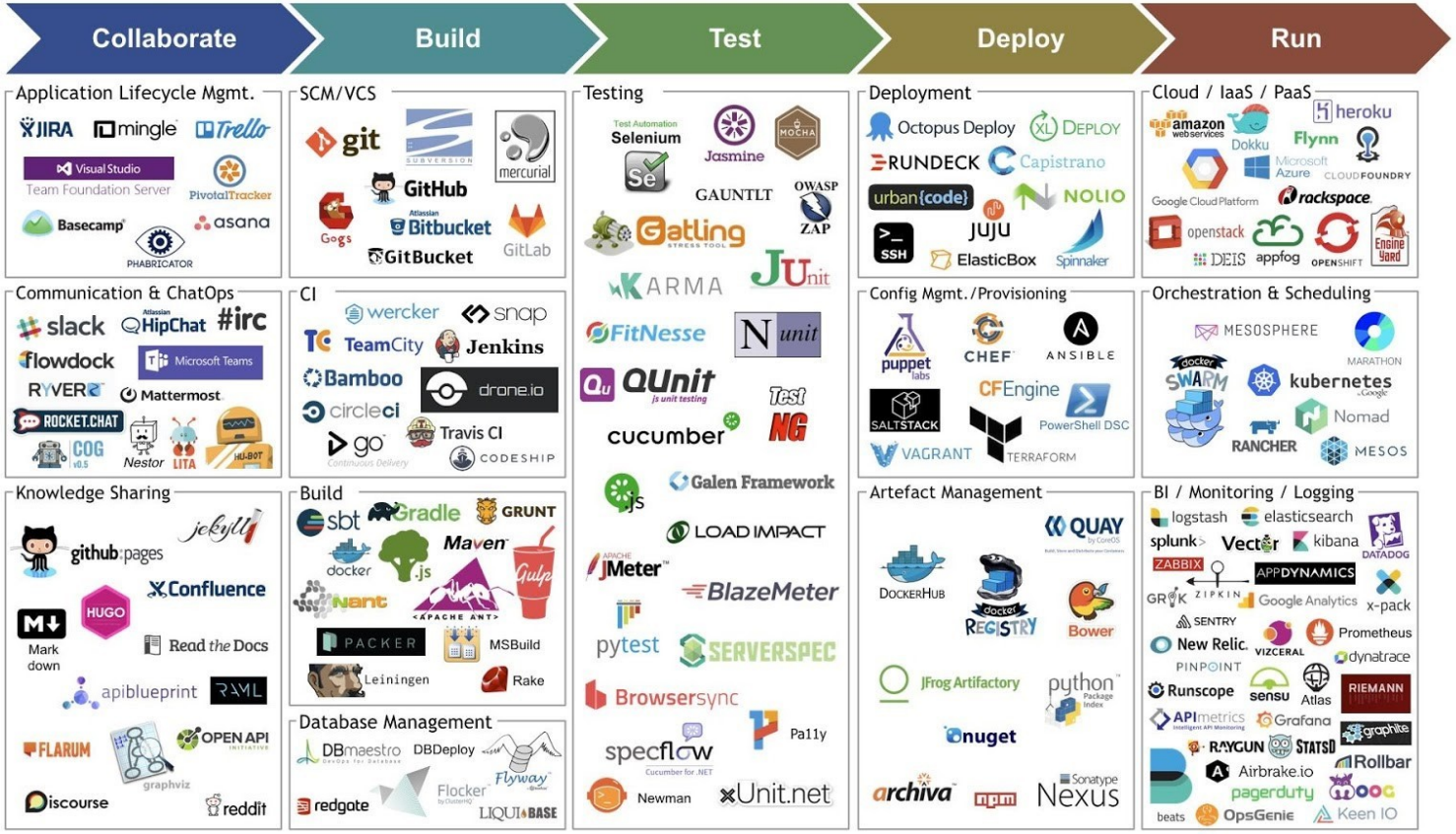
Boris Beizer

Pipelines


Development
Continuous Integration
Continuous Delivery
Deployment
Automation Server
Jenkins
System of record
Unit Tests
Packaging
Test environment
Ready to ship tests
- Regression tests
- Performance
- Compatibility
Staging
Post deployment tests
Production
Developer
Developer
QA
Publish
Promote
Promote
Import
Artifactory Repository
QA
UAT
Sys Admin
Legend
Manual Step
Manual Feedback
Automated Step
Automated Feedback
Artifact Repository
Static Pega7 system

Automation Tools

- A jungle of CICD tools
  - Each automates some part of the pipeline
  - Lots of overlap
  - Not all are compatible
  - Wide range in quality
- Developing a toolset
  - Can be very problematic
  - Especially if some tools become obsolete



CICD Benefits

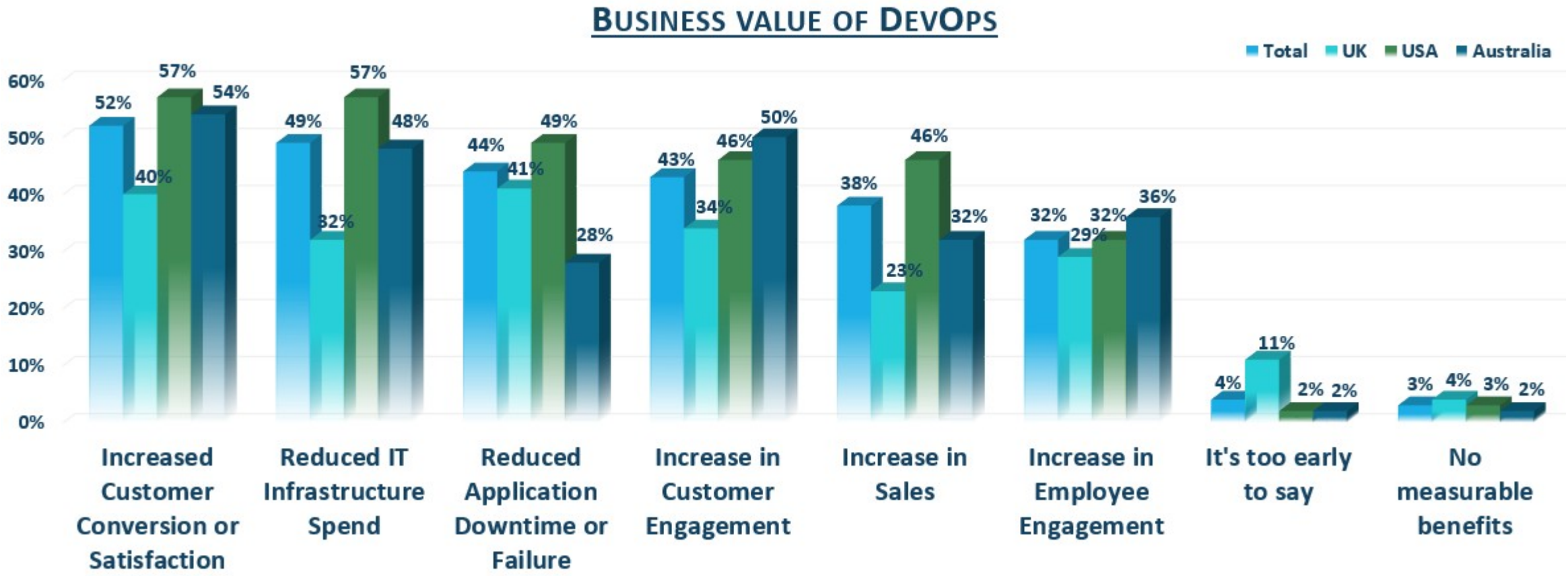
1. Smaller code changes are simpler (more atomic) and have fewer unintended consequences.
2. Fault isolation is simpler and quicker.
3. Mean time to resolution (MTTR) is shorter because of the smaller code changes and quicker fault isolation.
4. Testability improves due to smaller, specific changes.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

CICD Benefits

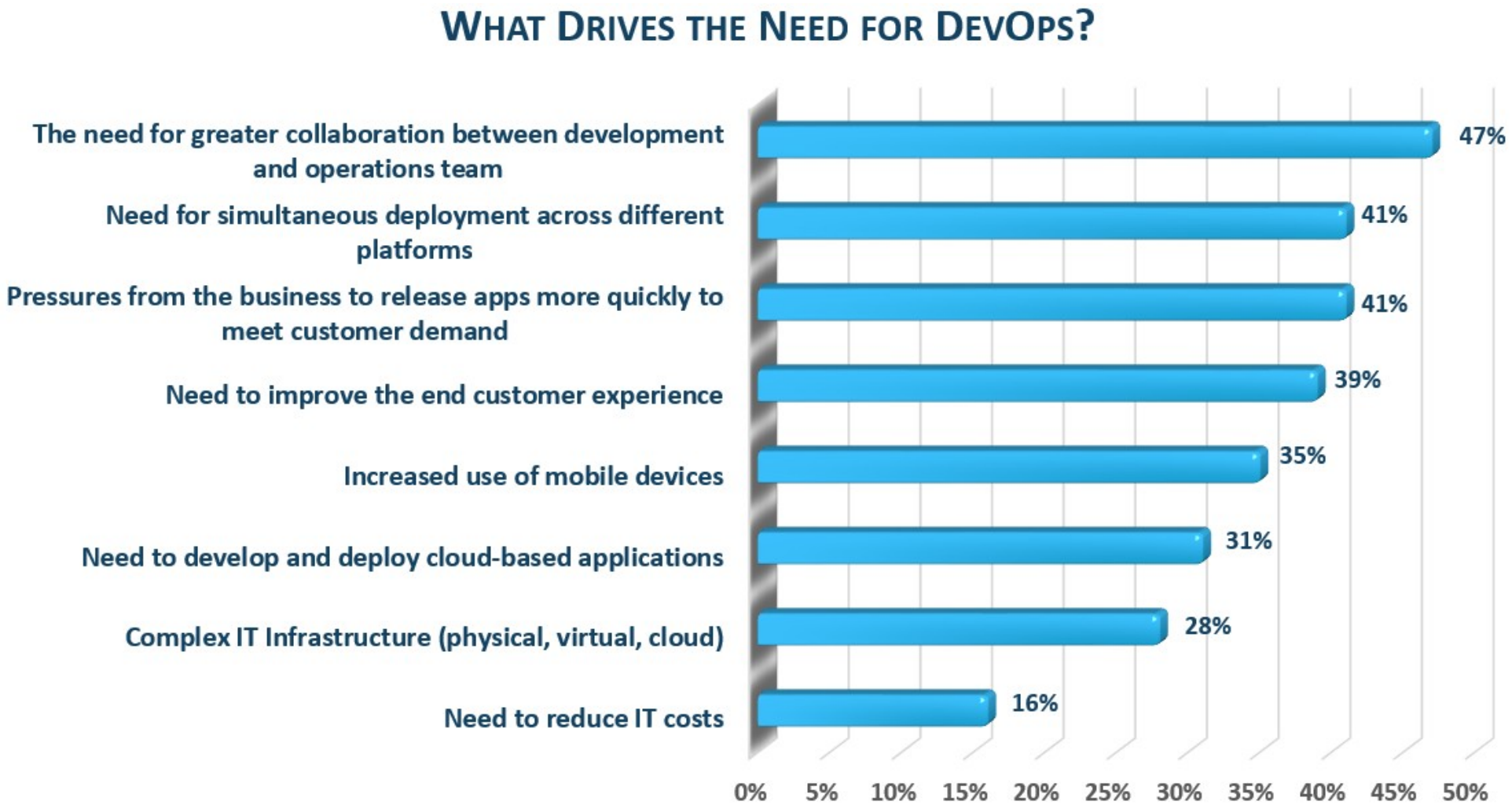
9. CI-CD product feature velocity is high. The high velocity improves the time spent investigating and patching defects.
10. Feature toggles and blue-green deploys enable seamless, targeted introduction of new production features.
11. You can introduce critical changes during non-critical (regional) hours. This non-critical hour change introduction limits the potential impact of a deployment problem.
12. 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.

https://help.mypurecloud.com/articles/benefits-continuous-integration-continuous-deployment-ci-cd/

CICD Benefits



CICD Drivers



IBM Internal DevOps

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

Challenges for CICD

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

CI and CD in Agile Development

- Continuous Integration and Continuous Delivery become essential ingredients for teams doing iterative and incremental software delivery in Agile Development
  - Developers share a common source code repository
  - Dedicated Continuous Integration environment
  - All code must pass unit tests
  - Integrate often
  - Regression tests run often
  - Code metrics are published
  - Every change to the system is releasable to production
  - **Automation is the key**

Continuous Integration

- Continuous Integration is a software development practice where members of a team **integrate their work frequently** , usually each person integrates at least daily - leading to **multiple integrations per day**
  - Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible (Martin Fowler)
- Goal is to merge and test the code **continuously** to catch issues early by **automating integration process**
  - Your project must have a reliable, repeatable, and automated build process involving no human intervention
- CI Server (Jenkins) is responsible for performing the integration tasks
- Automatic unit testing, static analysis and failing fast are core to CI

Continuous Integration Practices

- Single source repository for all developers
- Build automation
- Every change to VCS should make a new build
- Keep the builds fast and trackable
- Make the builds self-testing
- Test the builds in production-like environment
- Keep all verified releases in artifacts repository and available to everyone
- Publish coding metrics

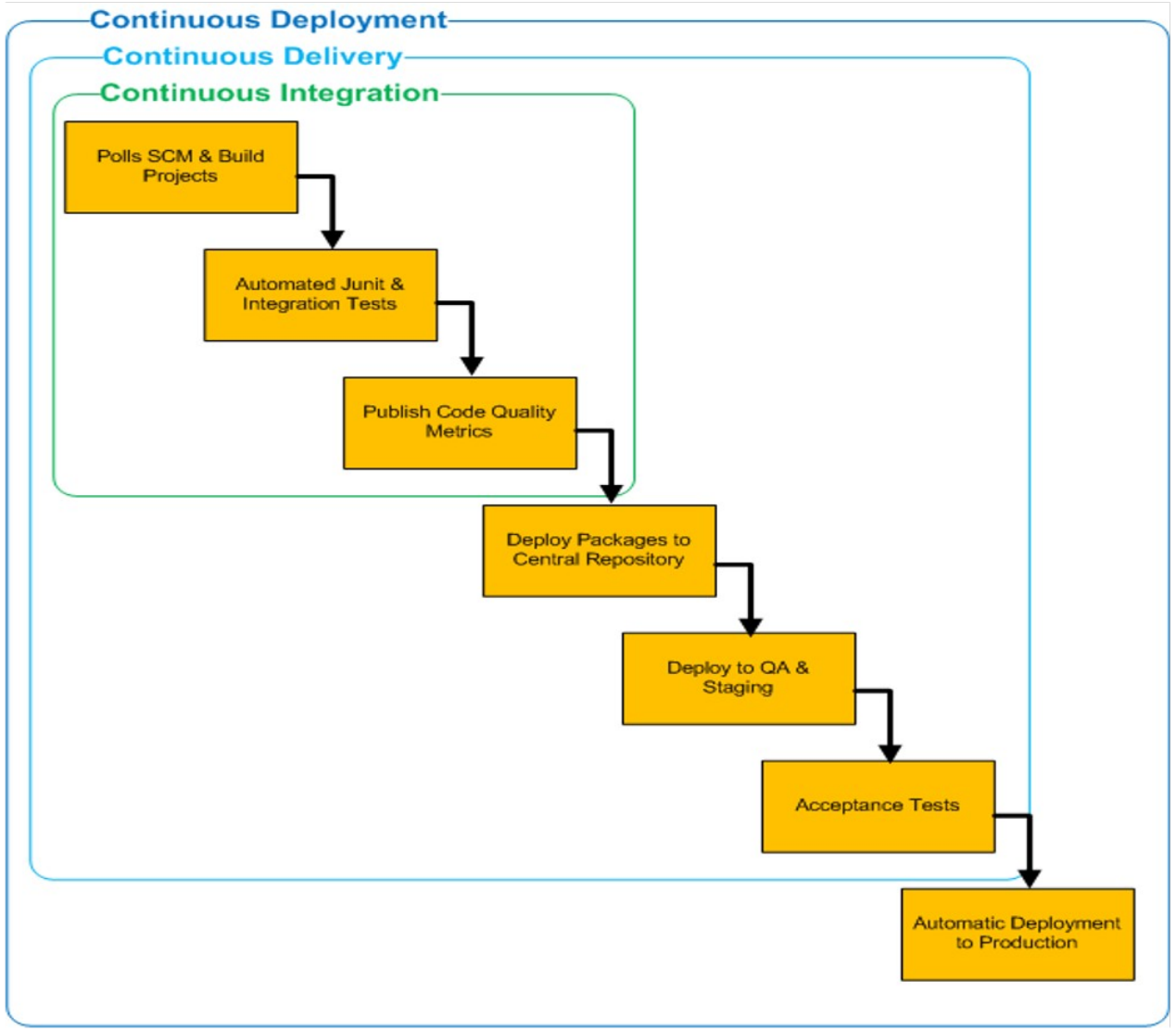
Continuous Delivery

- Continuous Delivery is a **natural extension** of Continuous Integration
  - Every change to the system has passed all the relevant automated tests and is ready to deploy in production
  - Team can release any version at the push of a button
  - But the deployment to production is **not automatic**
- The goal of CD is to put business owners in the control of scheduling of the software releases
- Continuous Delivery is a core principle of **DevOps**

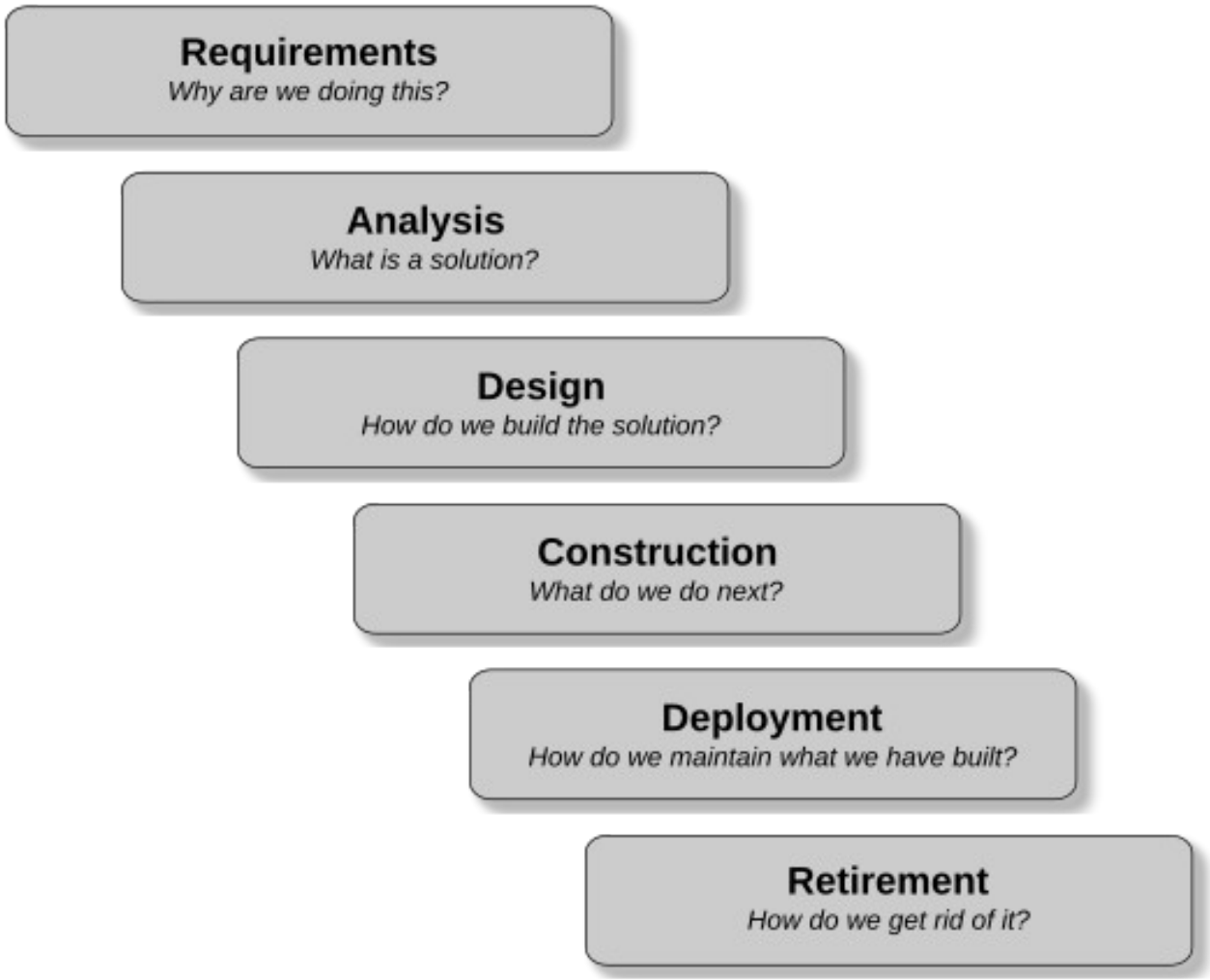
Continuous Deployment

- Continuous Development adds **automatic deployment to end users** in the Continuous Delivery process
- Continuous Deployment automatically deploys every successful build directly into production
  - Deploying the build to production as soon it passes the automated and UAT tests
- Continuous Deployment is not appropriate for many business scenarios
  - Business Owners prefer more predictable release cycles as opposed to arbitrary deployments

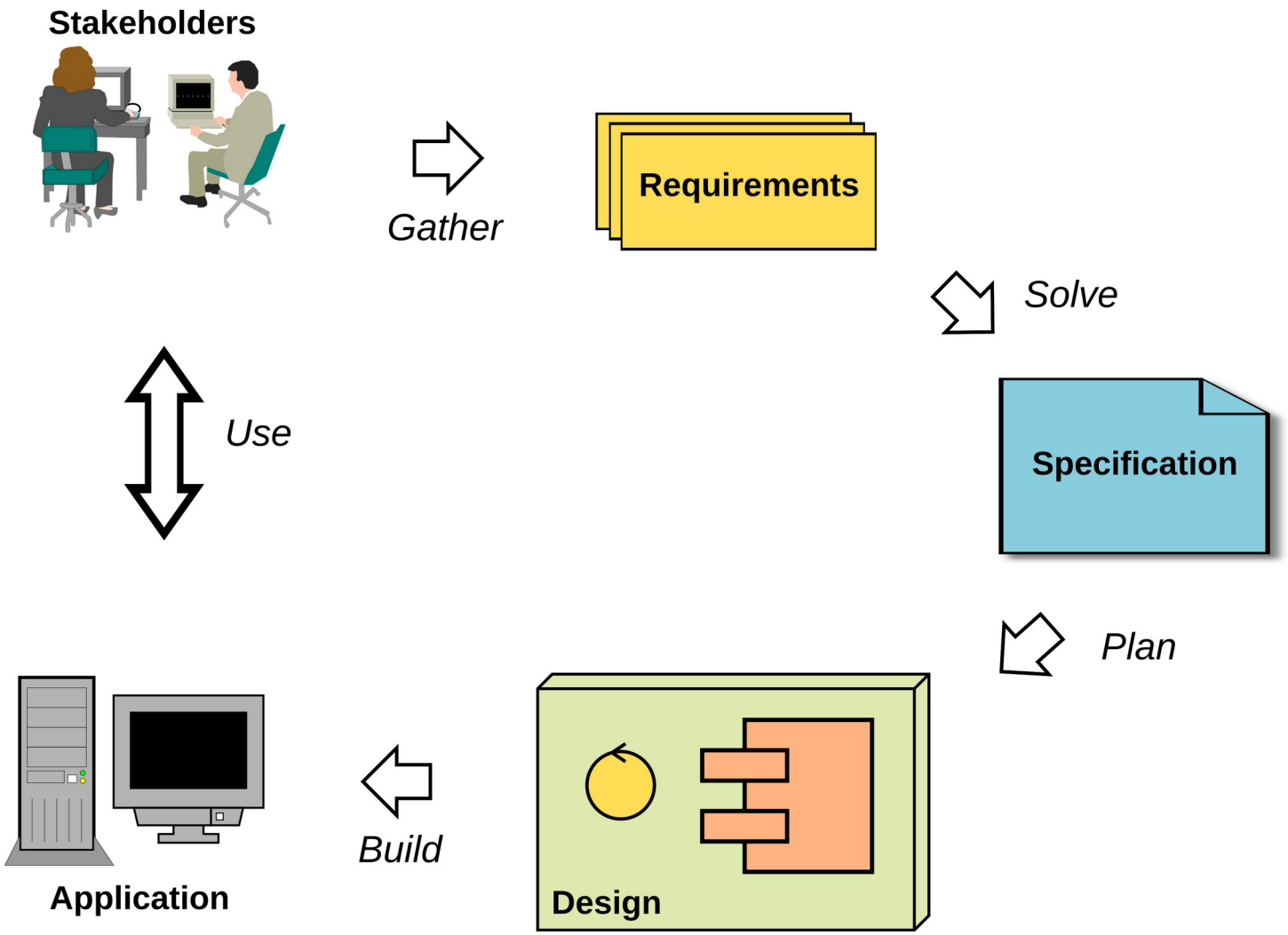
Continuous Integration, Delivery and Deployment



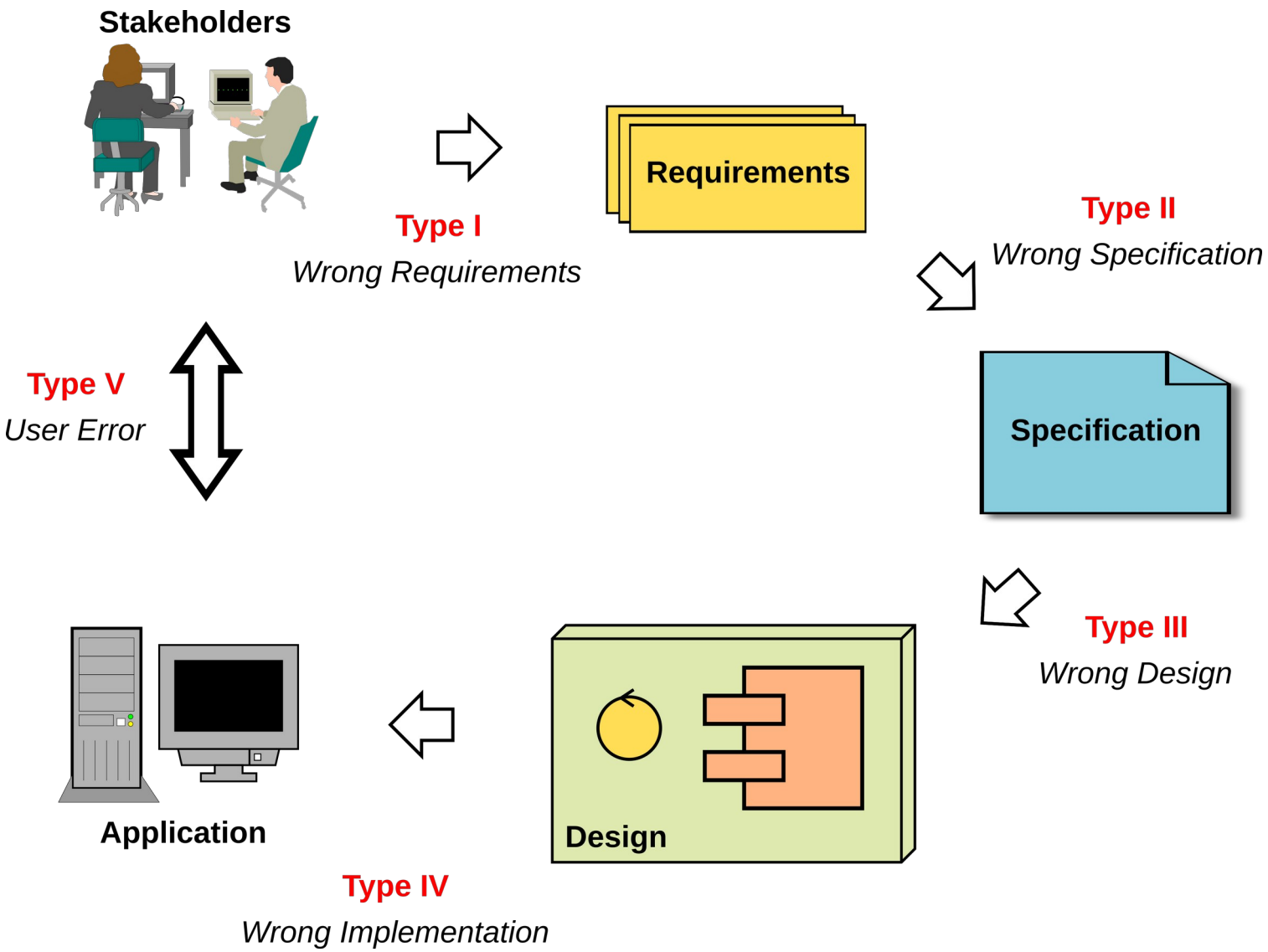
Engineering Process



The Generic Software Development Process



Where Things Go Wrong



Pipelines

- Processes can be ranked on a scale of “predictability
  - Depends on how much of the requirements we know in advance
  - If we know all the requirements, we can predict exactly what the final result should look like and behave
  - If we don’t know all the requirements, we have to continuously revise our target as we adapt to new and changing requirements
  - The fewer requirements we know or the more the requirements are in flux, the more adaptive the process needs to be

Predictive Processes

- All of the requirements are known at the project start
- Characteristic of projects that have high amounts of risk
- Usually implemented as a waterfall SDLC
- The final result or target can be accurately specified.
  - Very important for mission critical systems like software that runs a nuclear reactor cooling system or a heart pacemaker
- Often not suited for software that users interact with
- Predictive SLDCs start to become very inefficient and ineffective when requirements change quickly or are not completely known, especially true of new or innovative software

Adaptive Processes

- Many of the requirements are not known at the project start
- Very often we are focused on solving a problem but the nature of the solution is unknown so we must take a trial and error approach
- We may be deploying new technology where user requirements are not yet known
- Stakeholders have no idea what their requirements would be and often need to have a prototype to play with to start to identify their requirements
- Most commonly needed when requirements are fluid (like with user interfaces or automating interactions)

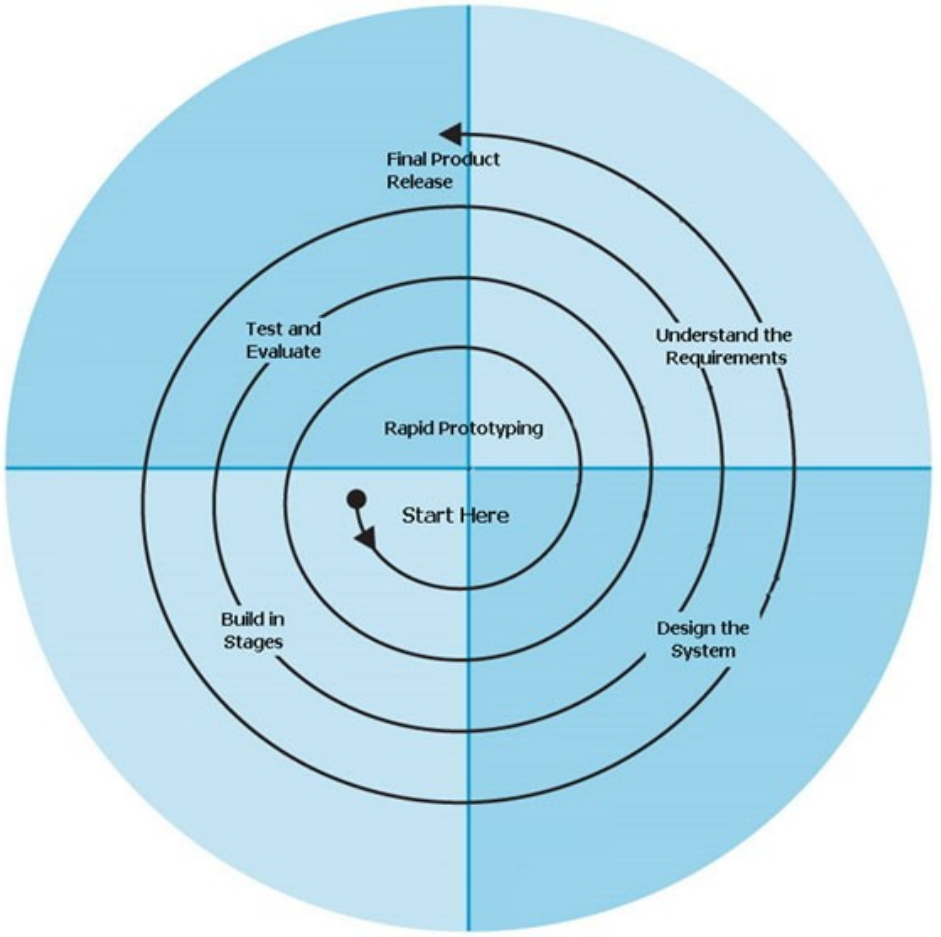
Non-Agile Adaptive SDLCs

Barry Boehm's Spiral methodology from 1986 defined a series of iterations with the objective of producing prototypes which were used to provide inputs into the subsequent iterations.

Another adaptive methodology is James Martin's Rapid Application Development (RAD) developed in the 1980s at IBM.

Both SDLCs was built around the idea that for some sorts of development, like working with user interfaces, the requirements are too fluid for a predictive approach.

The RAD approach, like the Spiral methodology, centred around getting a prototype into the hands of the users to start generating feedback that would be used to continuously develop the product.



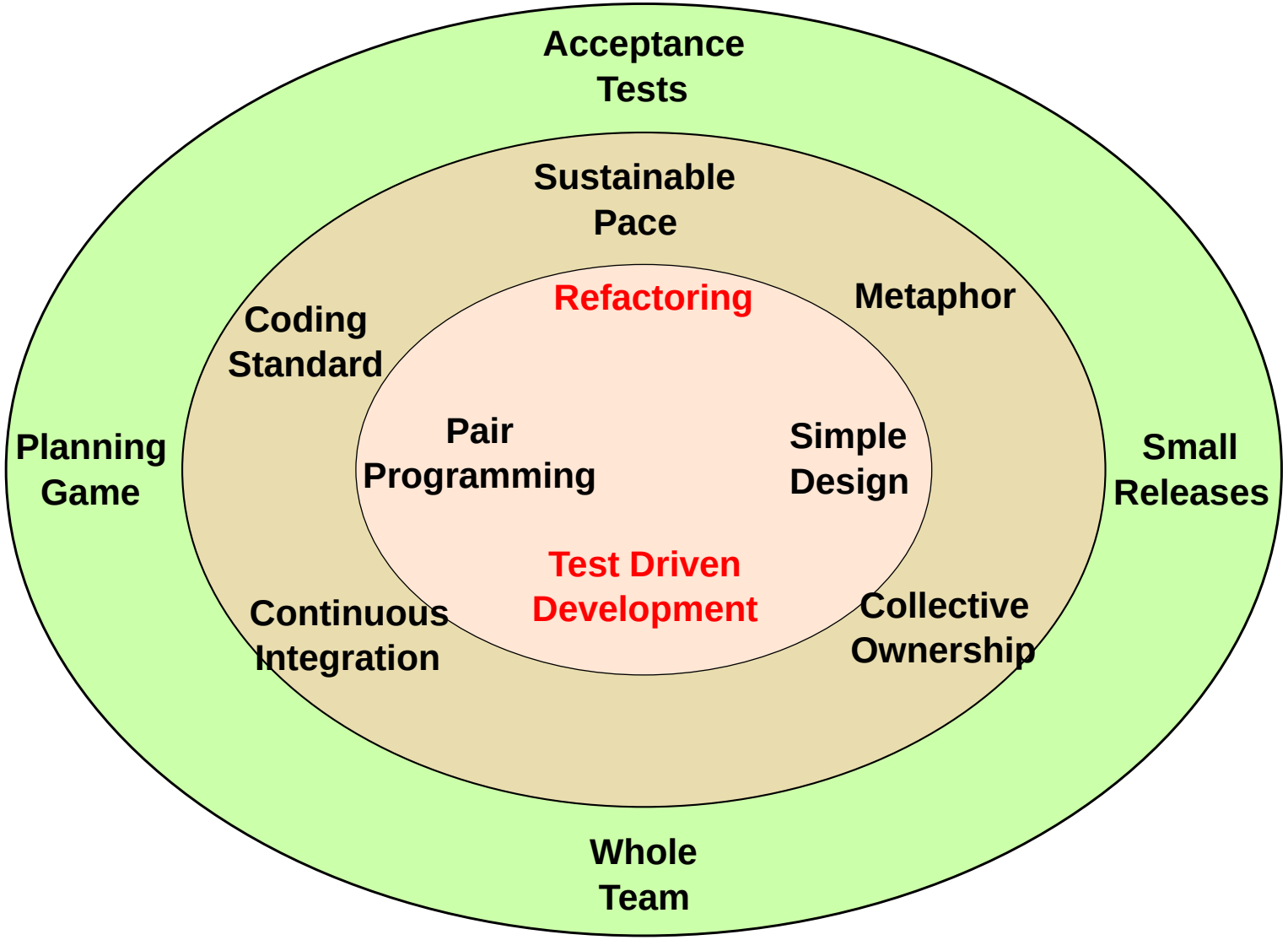
Agile Development Processes

- In the late 1990’s there was a rebellion against the high level of ceremony and formalism of RUP, which had become quite popular.
- Adaptive methodologies started to emerge that shared a common core of ideas:
  - Close collaboration between the development team and business experts
  - Face-to-face communication as opposed to written documentation
  - Frequent delivery of working prototypes to facilitate developer and customer communication
  - Small, tight, self-organizing teams
  - Sets of techniques to "craft" code and organize the team to anticipate the inevitable "churn" of constantly changing requirements

Extreme Programming (XP)

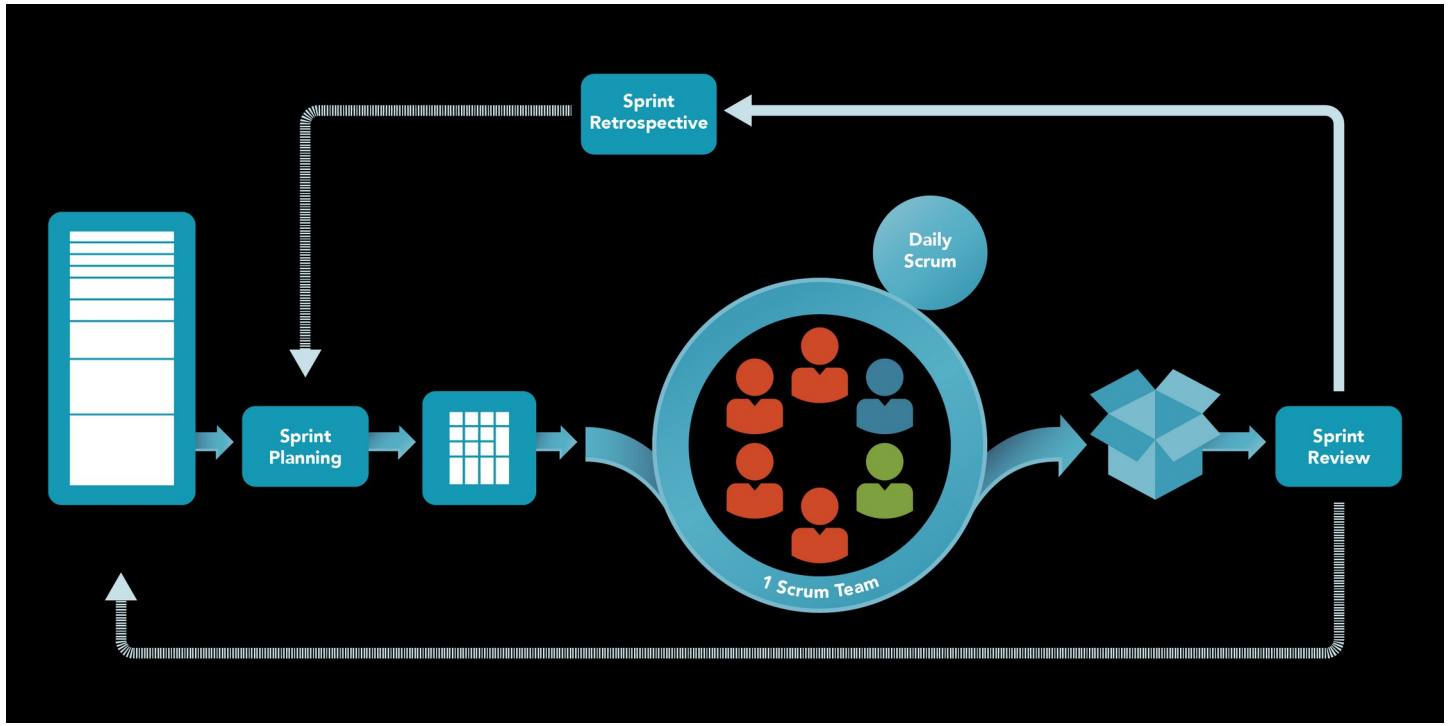
- Many modern Agile techniques originated in a methodology called Extreme Programming or XP
  - created by Kent Beck.
- XP is extreme means taking 12 development "best practices" to their logical extremes
- XP is intended to be easily used for projects of up to a dozen programmers and twice that with some difficulty
  - XP itself does not scale well
  - The way to do large scale XP development is within a project organized overall along more traditional models, but is then split into multiple smaller XP projects
- Influenced the development of the Agile Principles

The XP Onion


Acceptance Tests
Sustainable Pace
Refactoring
Metaphor
Coding Standard
Planning Game
Pair Programming
Simple Design
Small Releases
Test Driven Development
Continuous Integration
Collective Ownership
Whole Team

Scrum

- Scrum is not a method but is a framework for organizing software development activities in a quantitative manner
  - The use of Scrum improves management for any Agile methodology



End of Module