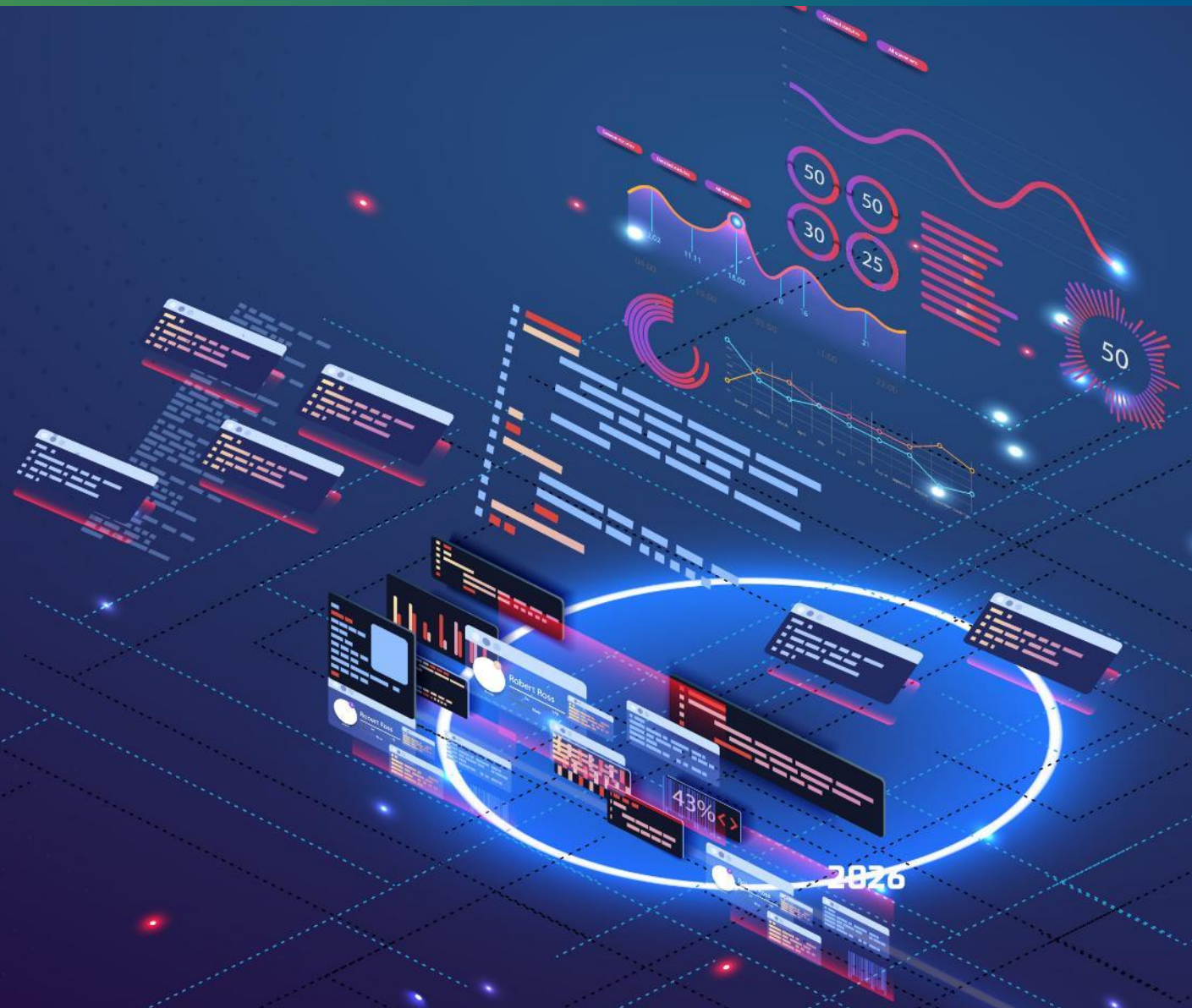
ISTANBUL
SOFTWARE
TESTING
CONFERENCE

# Old Dogs, New Tricks




Phil Royston, Tesena

#ISTC2026


tesena
SMART TESTING


Innovative Sy
tems
oducing new products a
ment Teams
dictable and
d unstable
s clear a
largely u
changi
s predictab
oesn't ex
own, but pre
many
adapt in re
is require

# Testing Conference


Old Dogs,
Old Tricks

# Big Old Trick

Requirements

Acceptance Testing

Syst Anal

tem ting

Verification

Validation

Detailed Design

Unit Testing

Coding

**Verification = building the system right**

**Validation = Building the right system**

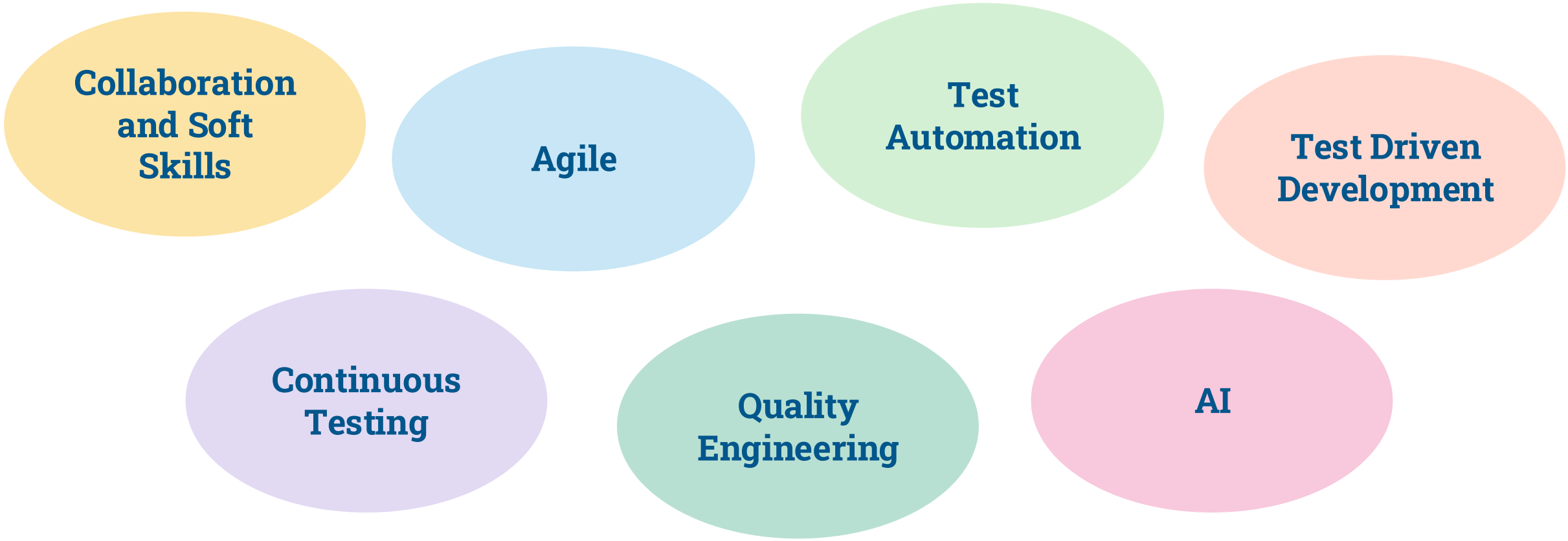
## The V-Model

# Early Days of Test Automation

CIO

Tester

# The new tricks you all have been talking about...




After a gap of many years, this old dog decides to go back to testing

**Except testing has become even more challenging than when I left...**

IT drives business value. It gives us our competitive edge.

Our clients are used to and expect constant innovation.

They are used to and expect cheap banking services.

Our clients expect our services to be highly available and high quality.

We need to be flexible to react to constant change in the market and in regulations.

Our competitors are more innovative and faster to market than we are!

I really need fast, cheap and good.

**Testing is viewed as a Bottleneck.**

**“Speed up testing, reduce costs but not reduce production quality.”**


The Agent of Change?


Introducing Change.
Pick Your Moment.

# Introducing Change

Means changing people.

You need to respect that that's going to be hard.

To change, people need to learn.

People only learn when they...

Are curious.

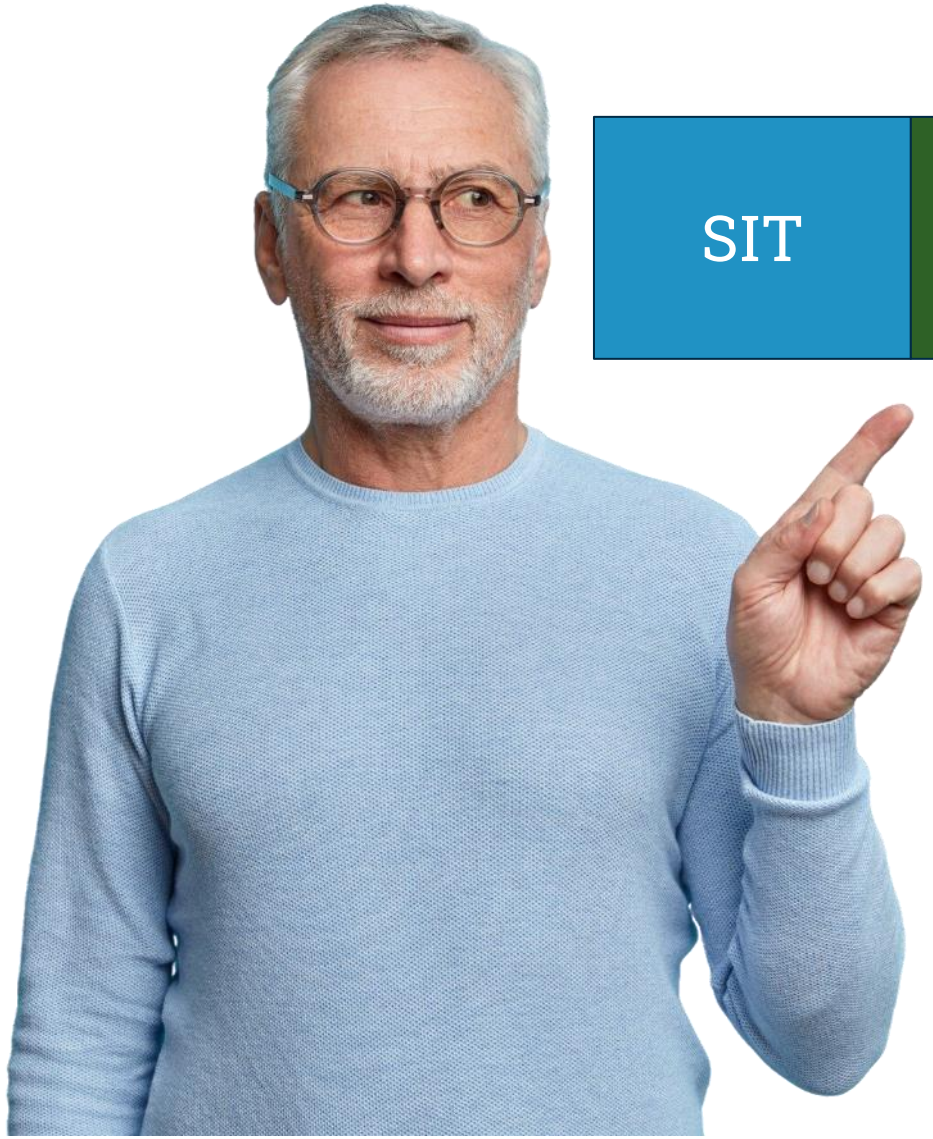
Think it is meaningful enough.

Are in awe.

Are in crisis.

Eat psychedelic mushrooms.

# What’s slowing testing down?



| SIT | ACCEPTANCE | REG |
|---|---|---|

Why this long acceptance testing phase in the release calendar?

# Testing Obstacles?

Tester’s Instinct

Big Releases

Unstable Test Environment

Fear of the Unknown

# Survivors of 35 Years of Reinvention

**THEN · pre-1990**

Mono-business communist banks. Zero competition. Paper-based services running on communist mainframes.

35 years

**NOW · 2025**

Ultramodern, digital, innovative, very competitive, highly regulated and well trusted banks.

**A 35-YEAR YEAR STORY OF**
**TURMOIL**
**PRIVATISATION**
**CRISIS**
**CONSOLIDATION**
**AND CONSTANT NEED FOR INNOVATION**

## ...LEAVING A LEGACY

### Large Business Portfolios

Multiple and deep business segments

### Complex Architectures

Mix of modern and ancient, multiple core systems from consolidation

# Why Big Releases?

Deeply coupled systems.

- One change can impact many systems.
- Handling cross impacts feels easier and safer in big release.

Quality Mindset

- Rollouts are complex and risk production stability
- Release less, less releases means releases have more scope

## Simplification

Major Banks go through transformation programs every 10-20 years to reinvent themselves.

# Testers Instincts

## Testing at the end

***We know we shouldn't do it, we do it anyway.***

***Some reasonable sounding "reasons"***

- **Realistic**
- **Efficient**
- **Stability**
- **Support**

# Testing at the End

**Every defect means we go back and…**

**REDO One or more of**

**Requirements, Design, Coding, Deploy and Testing**

**REWORK MEANS THE WORK YOU DID BEFORE IS WASTED because you have to do it again**

# Question

**What will you do if you find a regression, performance or security defect just a couple of days before scheduled rollout?**

# Why is the Test Environment Unstable?

Poor Quality is Destabilizing the Environment

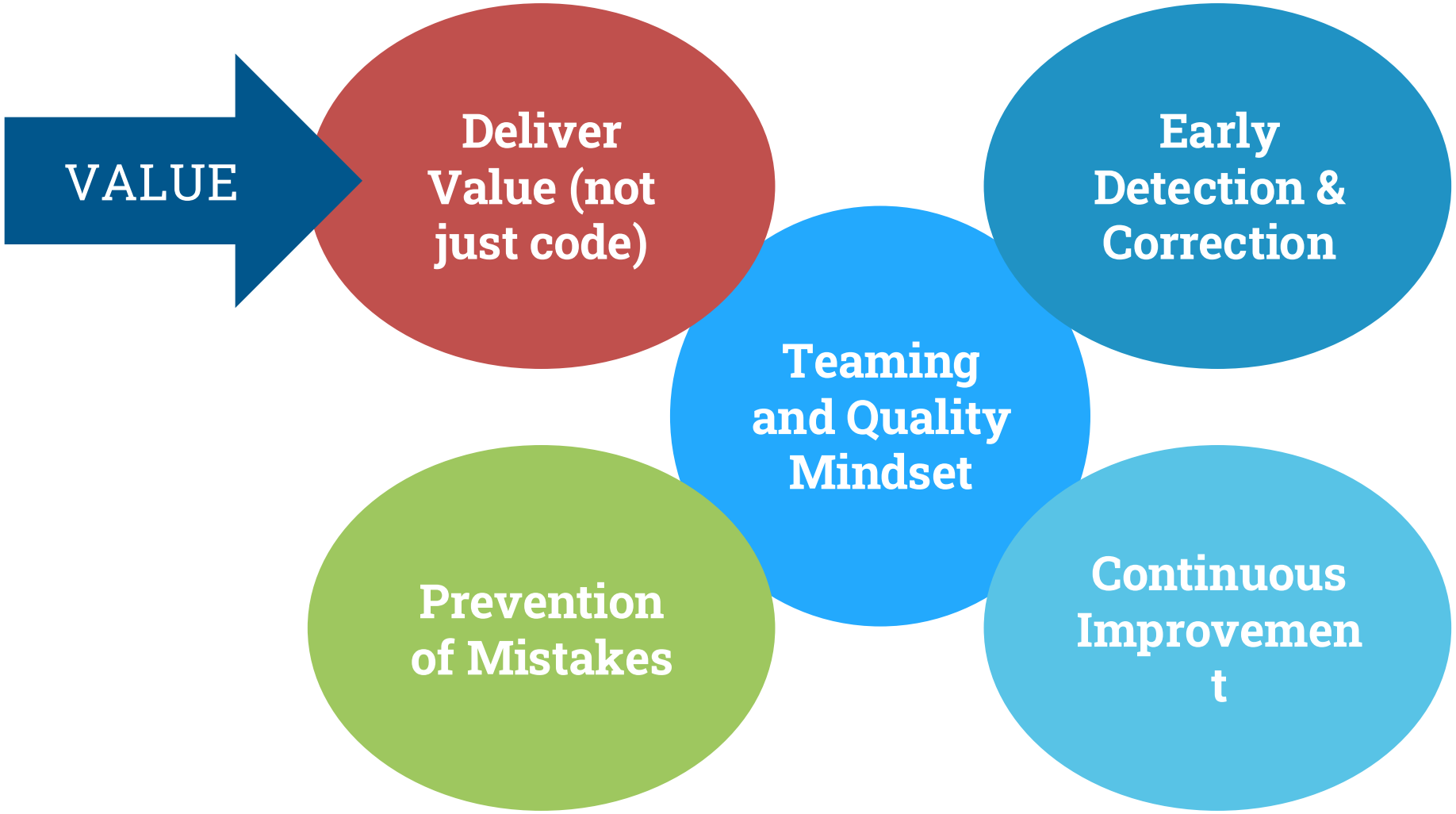
Broken, Inconsistent, Slow Deployments

# Fear of the Unknown

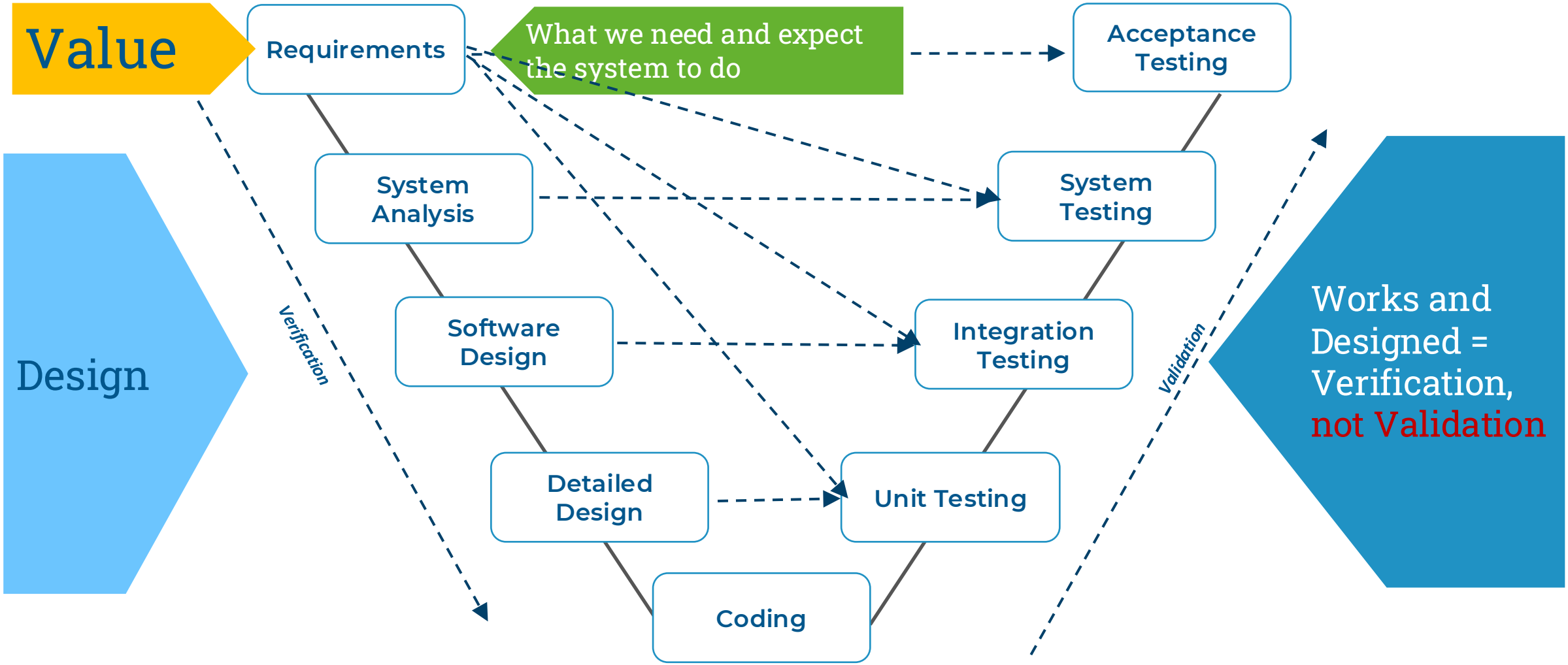
Regression

# New Tricks

# Quality Engineering Principles



# Where's the value in the V-model?



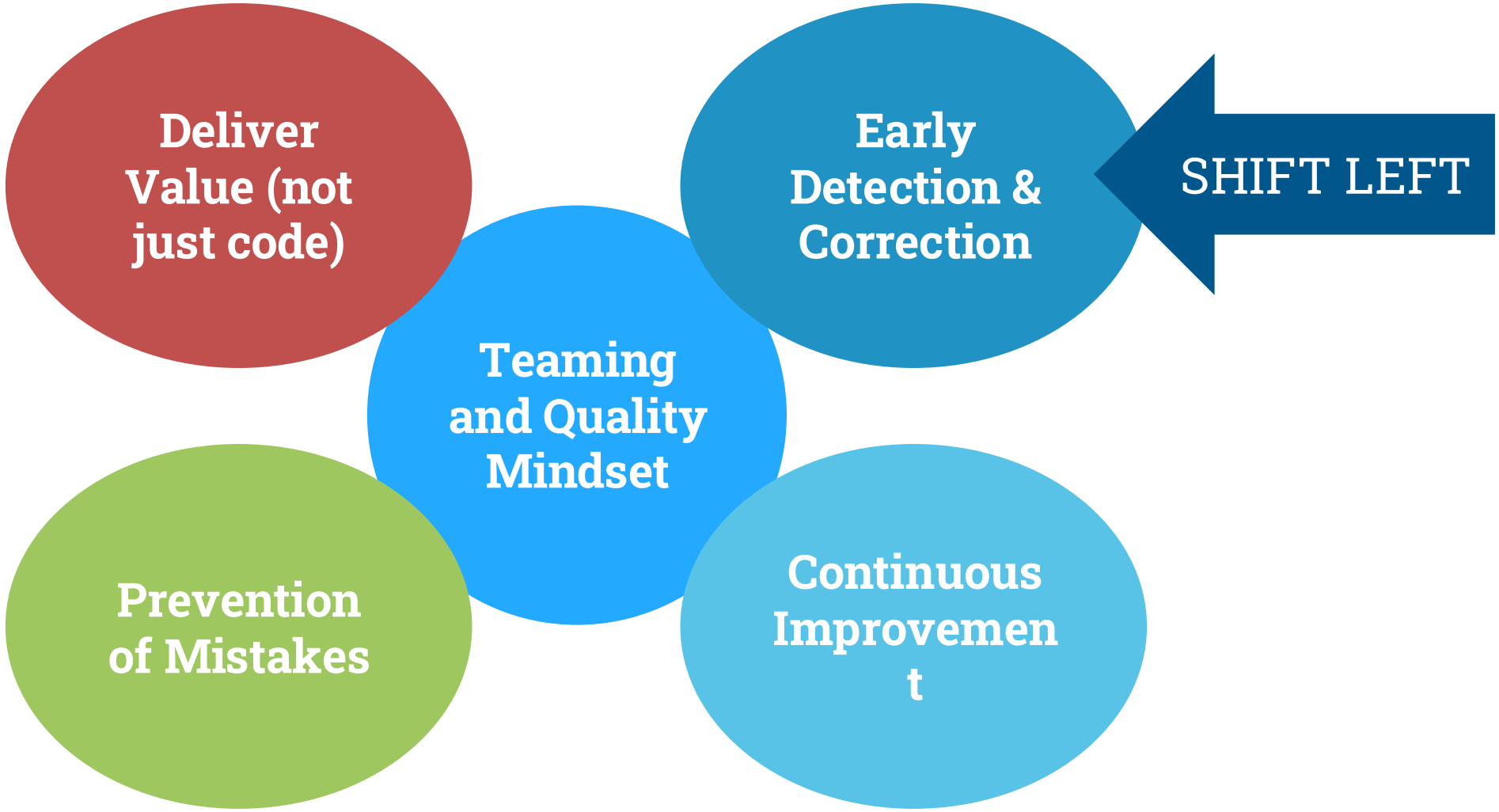

The V-Model

# Business Driven Testing

## Testable Business Requirements

## Early and Continuous Collaboration with Business

# Quality Engineering Principles



# Why Shift Left?

## Stop the spread

Dung beetle effect — mistakes get bigger as they roll downhill

## Easier to isolate

Faster feedback = faster root cause

## Flow

Context switching is expensive

## Still fresh

Faster to fix, less to re-learn

## Learning loop

See cause-and-effect immediately

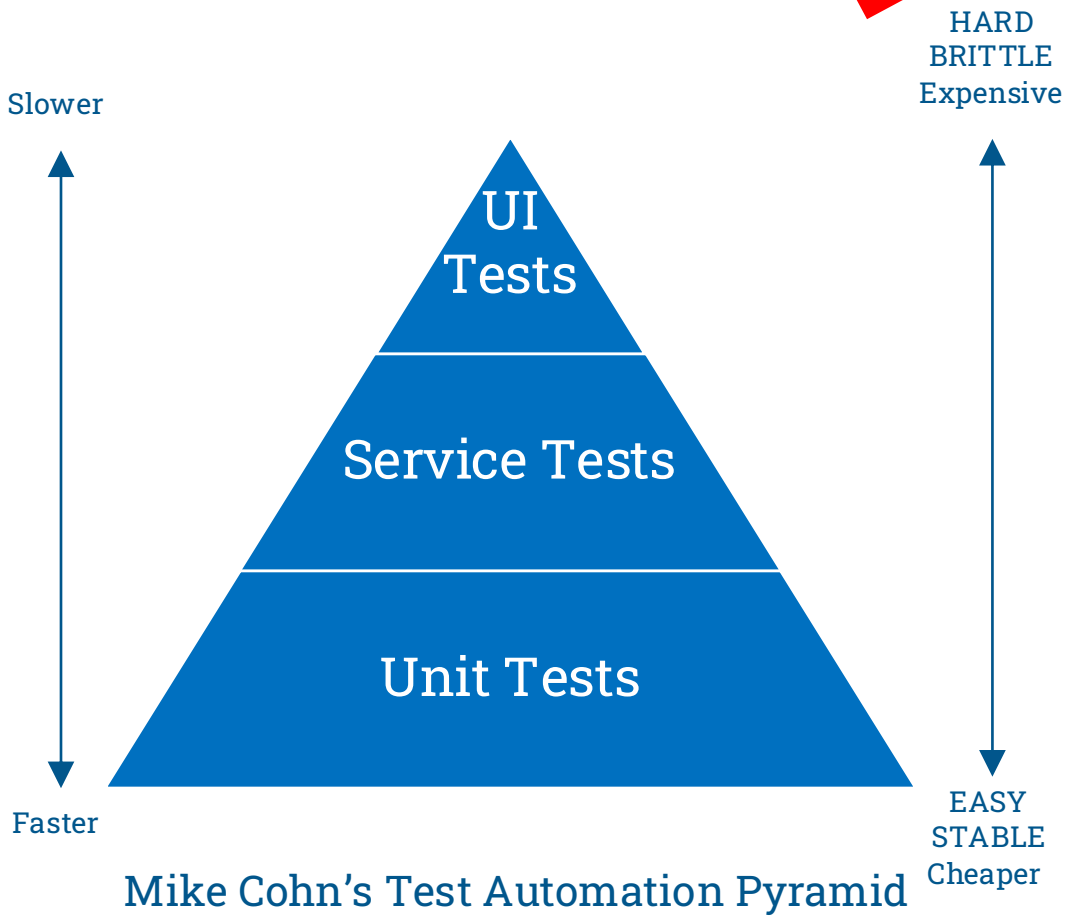
## Rework is waste

Late = expensive (testing at the end)

# Test Levels

## Is it just me?

# The Test Automation” Pyramid




Mike Cohn’s Test Automation Pyramid

Not only writing automation scripts

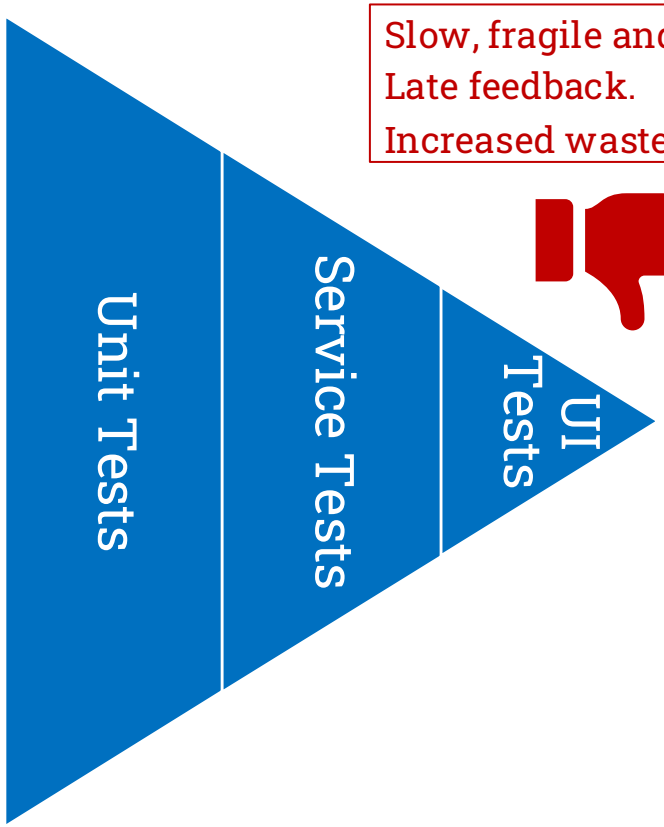
- Testing a system’s feature through the application screens is hard
- Testing through the service layer is a good compromise
- Testing a system’s feature isolated from everything else is easy

# Decoupled Testing

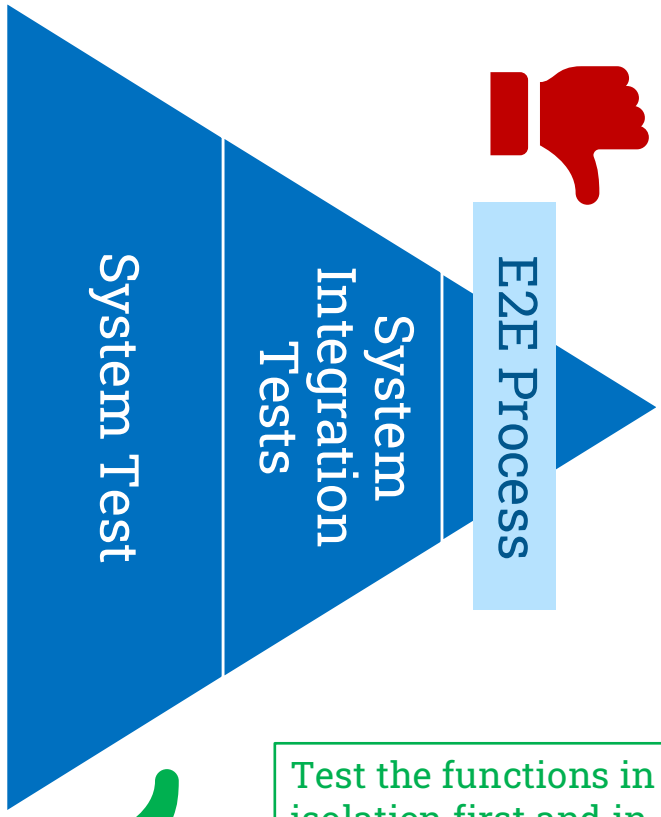
**Testing things in isolation means you won’t get blocked by late deliveries or bad quality of other things**

# Alt-Pyramids


Slow, fragile and expensive testing.
Late feedback.
Increased waste and rework.




Faster, robust and cost-effective testing.
Fast feedback.
Decreased waste and rework.




Test the functions in isolation first and in many combinations

# Golden Rules

## Always Test the Top of the Pyramid

It gives the highest confidence that the system actually works.

## Test the UI and the End-to-End Process

You need to prove the journey the user or business process takes

## Make it Effective and Efficient

Don't repeat the same test as you move right — each layer should add something new and you can reduce the test variants on the functions or features.

# The BIG "Buts"

**UNIT**

## Unit Tests Don't Fulfil Their Potential

- Developers have high code coverage in unit tests, but quality still BAD?
- Mocks or simulators are needed to isolate the unit — these are NOT REALISTIC

**SERVICE**

## Testing Services Doesn't Fulfil Their Potential

- We test interfaces a lot, but the integration between systems still doesn't work
- Testing one side of a conversation doesn't prove both sides are having the same conversation

**SYSTEM**

## The System Needs to be Integrated to Work

- It's not possible to test a system's function in because I need integration with other systems
- We cannot mock out the integration — mocks and simulators are NOT REALISTIC


I think I have a testing Ace up my sleeve
Contract-driven testing

# Contract-Driven Testing

Two teams stay in sync — without integration tests

1

## Agreement

*is made*

Consumer defines expectations on the Provider in form of tests

2

## Contract

*is published*

A shared file describing the agreement — the source of truth

3

## Provider

*verifies*

“Yes, my API fulfils what you asked for”
No broken integration deployed

4

## Consumer

*verifies*

Developers and testers can use reliable mocks created from the contract

# Ace Card Benefits

1

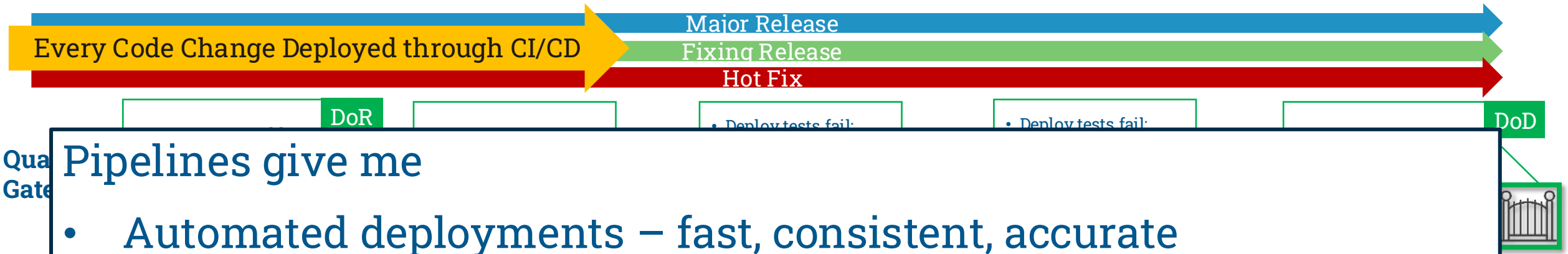
**Broken integrations don't get deployed to test environments — environments are more stable**

2

**Decoupled testing means more and better testing earlier**

> *Applying techniques like specification and example means we can include business-like use cases — so we can test that the value is being delivered early and providing even more realistic mocks*

# Pipeline Driven Testing




Pipelines give me

- Automated deployments – fast, consistent, accurate
  - More and more reliable testing
- Automate my automation
  - Early and continuous smoke and regression testing
- Possibility to create quality gates for fast feedback
  - Code is tested before it is integrated (contract tests pass)
  - Testing as part of deployment
- Gets the attention of IT guys (just say “CI/CD”)

# Test Automation.

# You've heard it all already.

# What tricks are we trying to apply?

- Shift Left
- Shift Down
- Pipeline Driven Testing
- Quality Gates with a Goal
- (Contract) Test-Driven Development
- Business-Driven Testing
- MAYBE
- Decoupled System Testing


Big Ships Don’t
Change Direction Fast

# A Steady Hand on the Tiller

# People Don’t Change Direction Fast


You are not Taylor Swift!

# THANK YOU

ISTANBUL
SOFTWARE
TESTING
CONFERENCE

# Thank you!