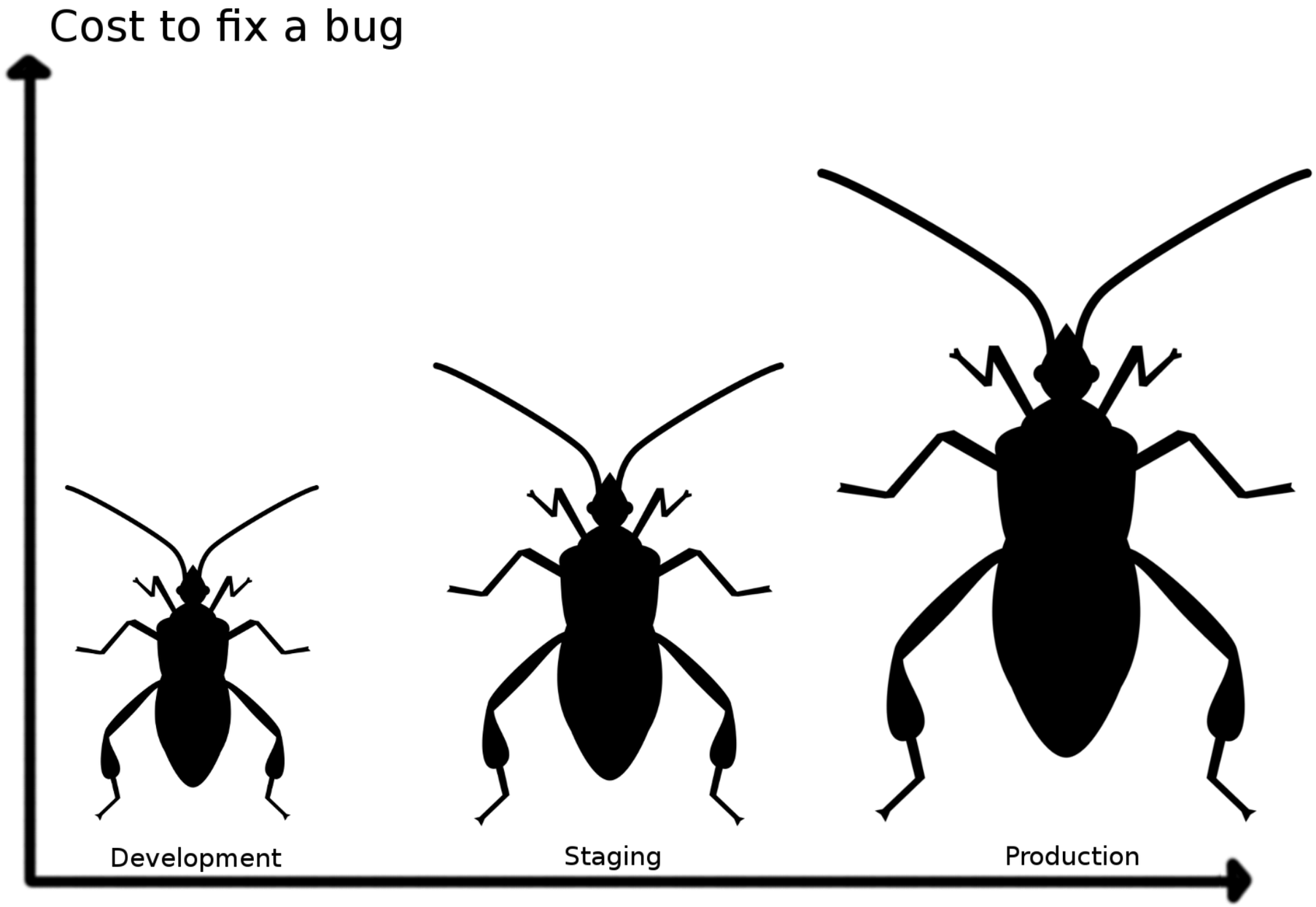
สยามชำนาญกิจ
SIAM CHAMNAN KIT

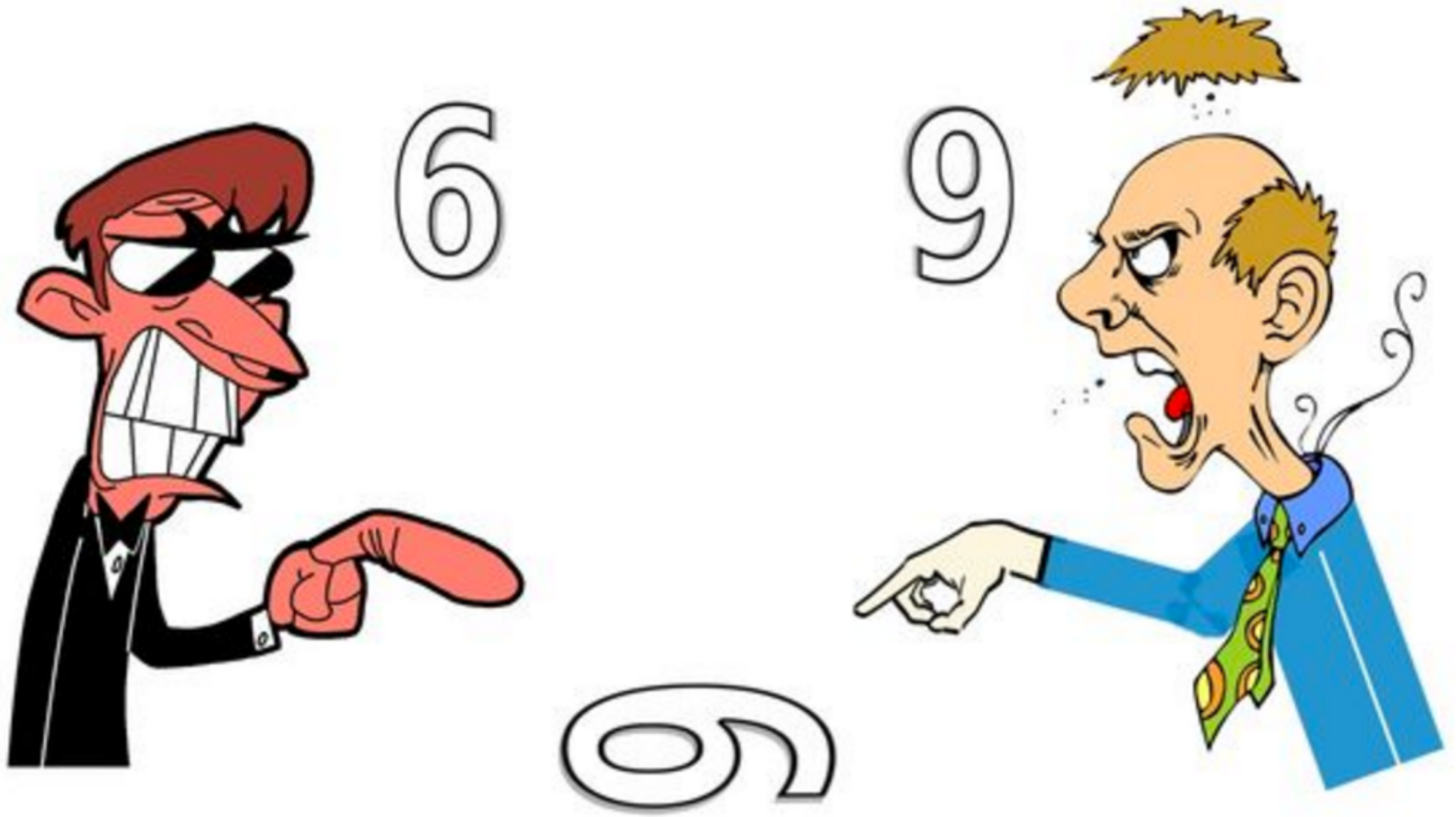
Automated testing

Automation Testing

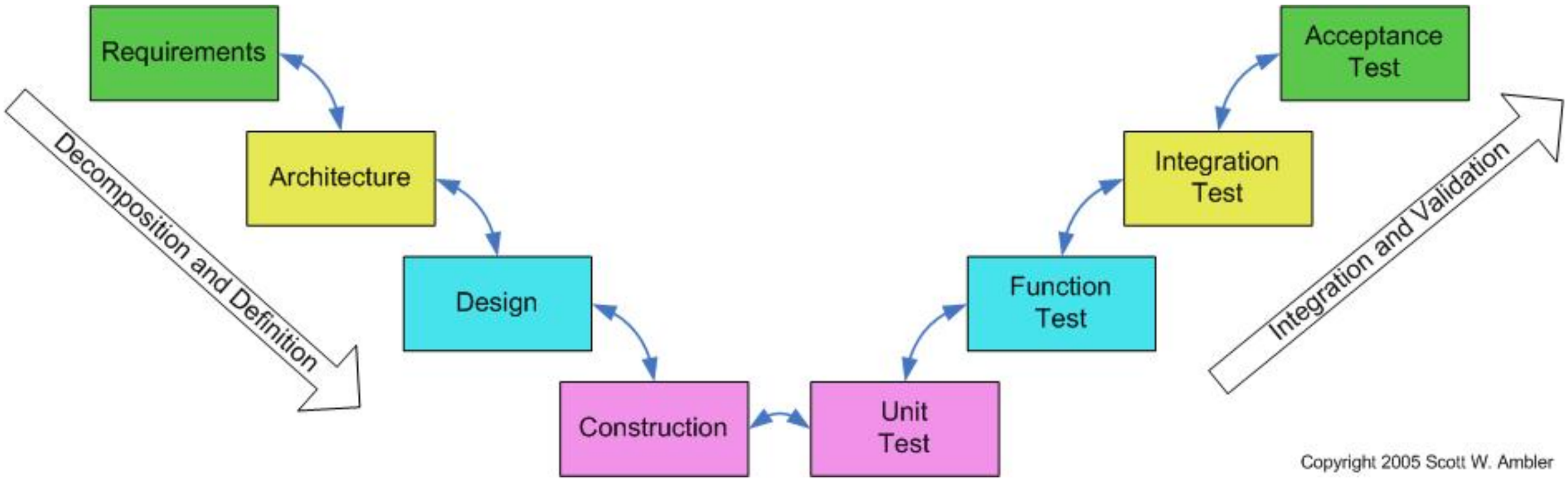
Cost of bug



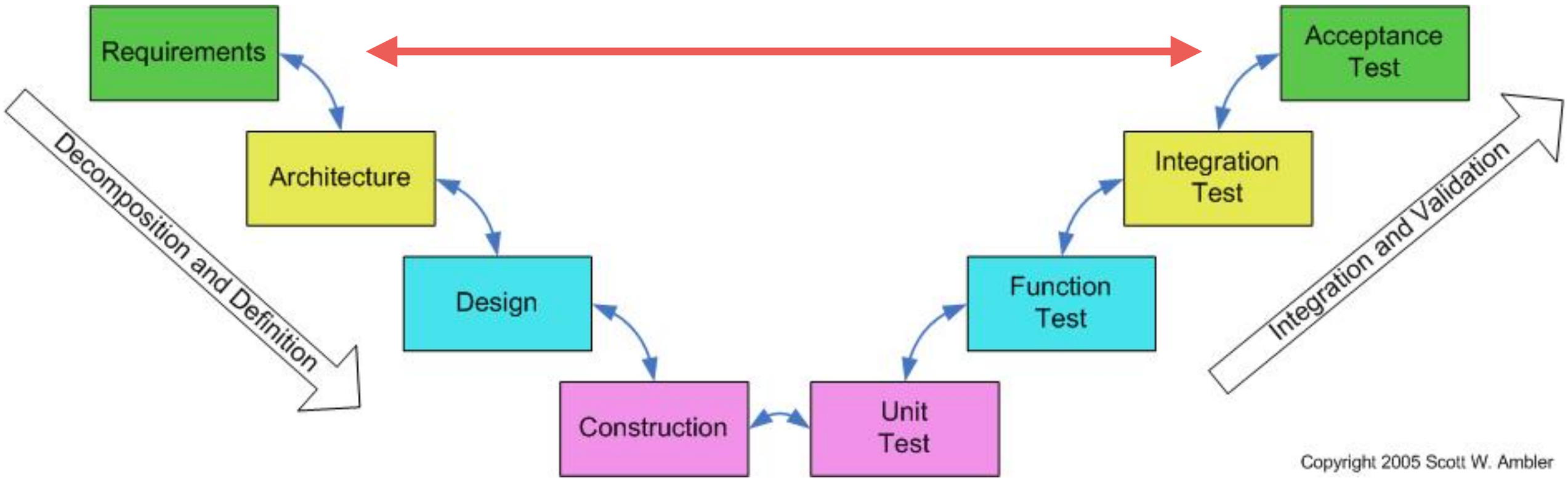
Developer vs Tester



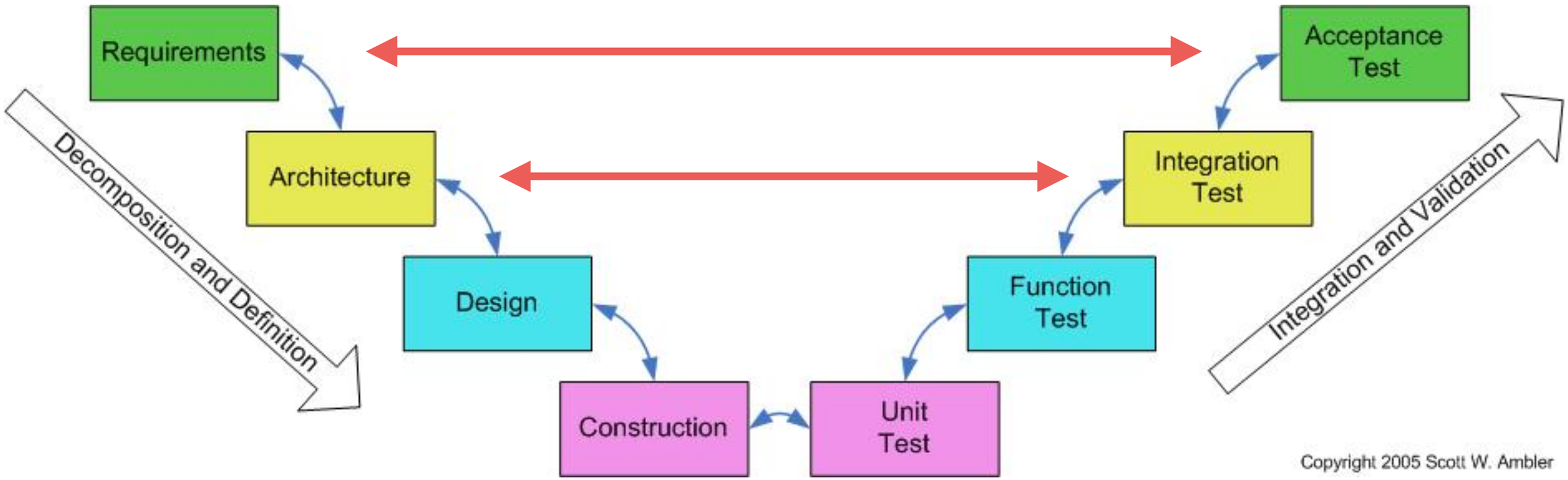
V Model



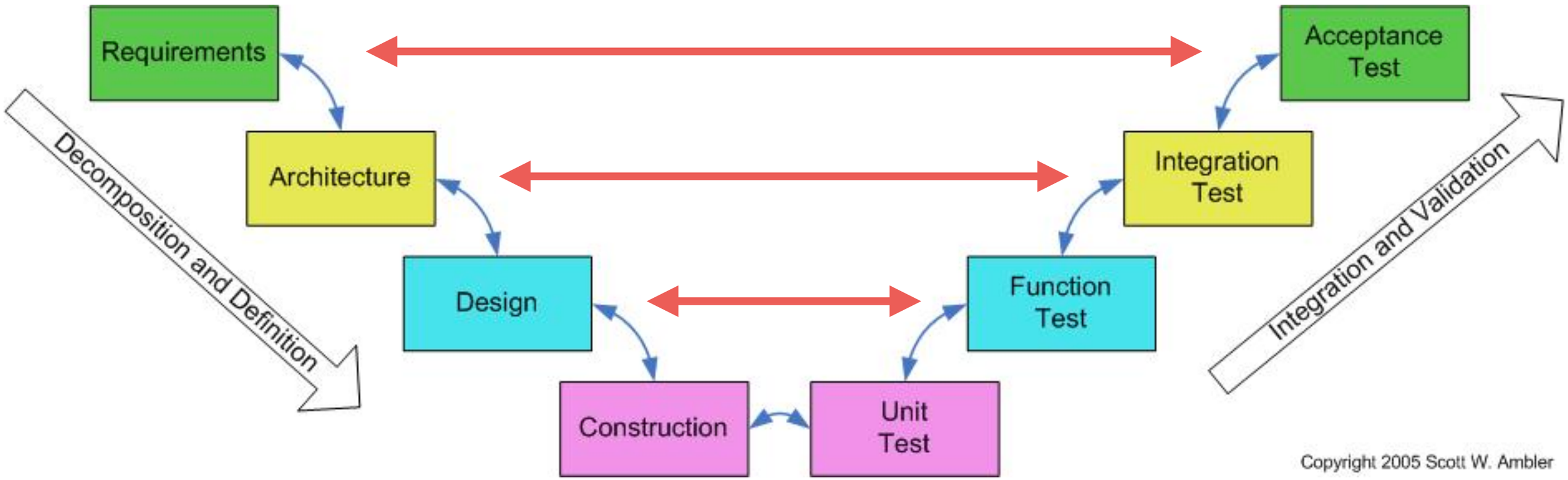
V Model



V Model



V Model



Need the new way !!

COVER FEATURE

Iterative and Incremental Development: A Brief History

Although many view iterative and incremental development as a modern practice, its application dates as far back as the mid-1950s. Prominent software-engineering thought leaders from each succeeding decade supported IID practices, and many large projects used them successfully.

Craig Larman
Valtech

Victor R. Basili
University of Maryland

As agile methods become more popular, some view iterative, evolutionary, and incremental software development—a cornerstone of these methods—as the "modern" replacement of the waterfall model, but its practiced and published roots go back decades. Of course, many software-engineering students are aware of this, yet surprisingly, some commercial and government organizations still are not.

This description of projects and individual contributions provides compelling evidence of iterative

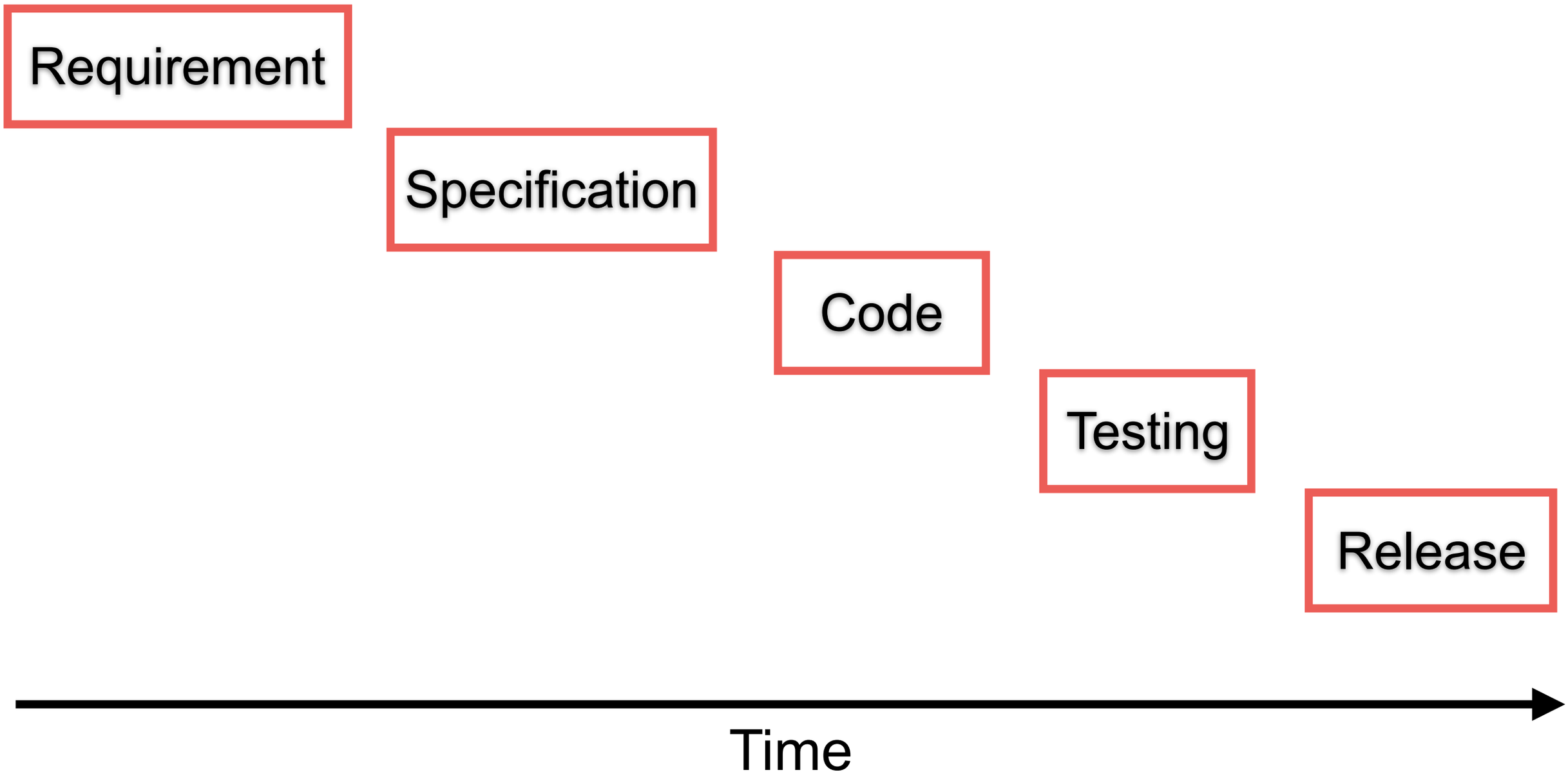
opment" merely for rework, in modern agile methods the term implies not just revisiting work, but also evolutionary advancement—a usage that dates from at least 1968.

PRE-1970

IID grew from the 1930s work of Walter Shewhart,[1] a quality expert at Bell Labs who proposed a series of short "plan-do-study-act" (PDSA) cycles for quality improvement. Starting in the 1940s, quality guru W. Edwards Deming began

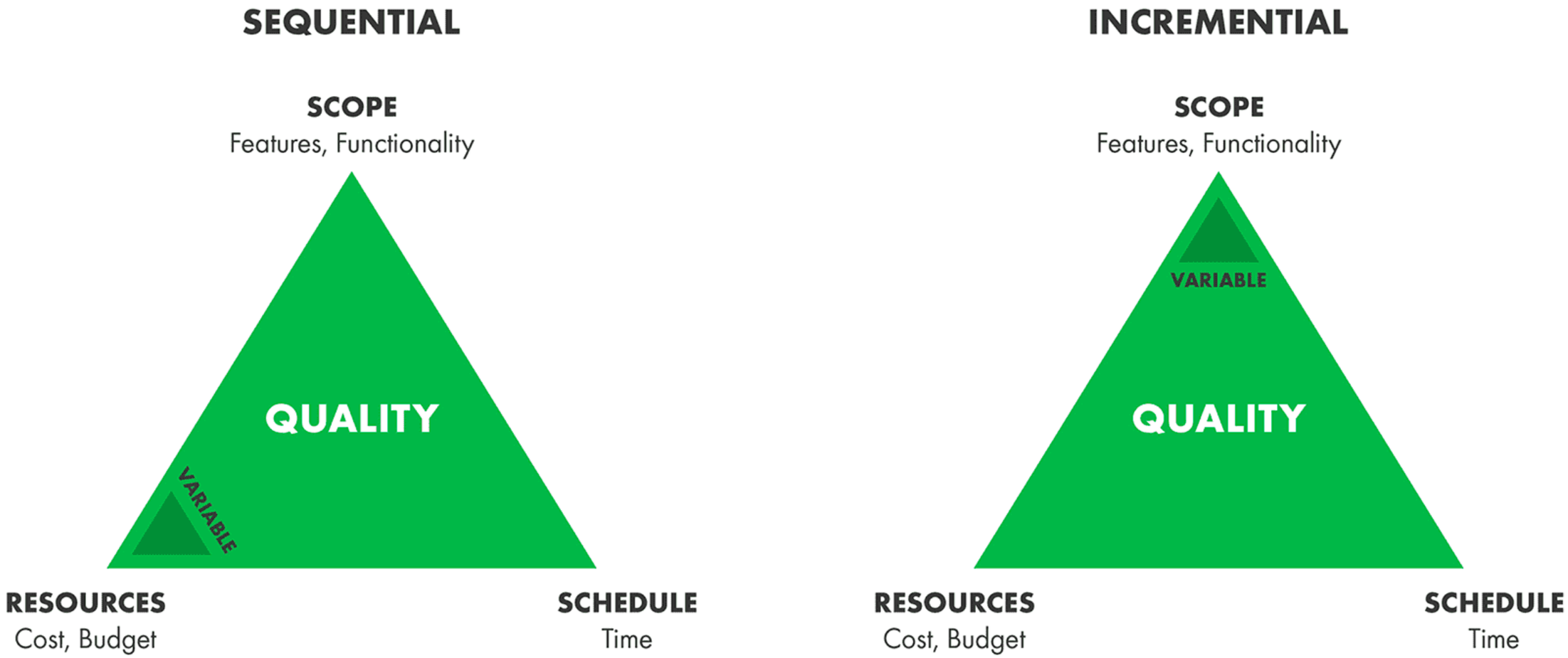
http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf

Sequential process



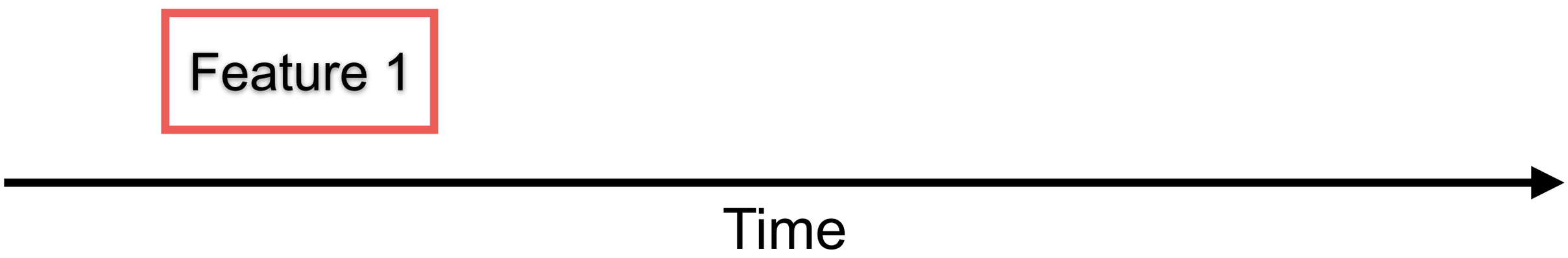
Iterative and incremental process

Time boxed delivery
Fix time and flexible scopes



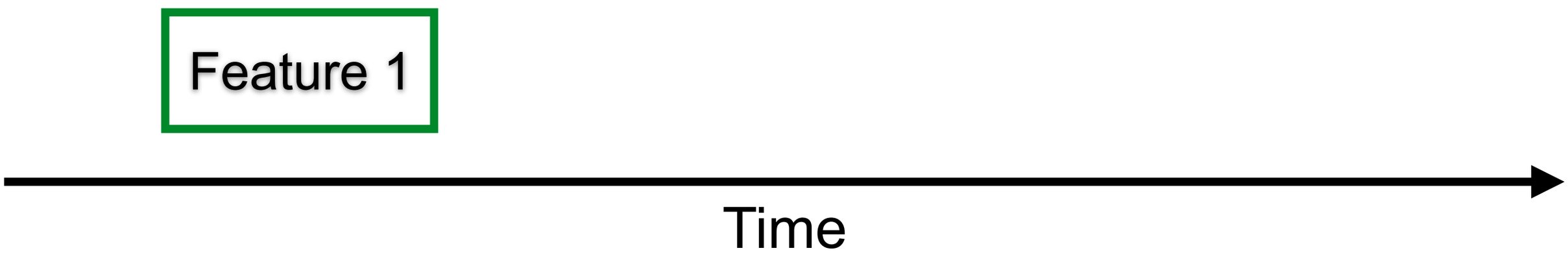
Business Technology Standard
www.managebt.org

Iterative and incremental process



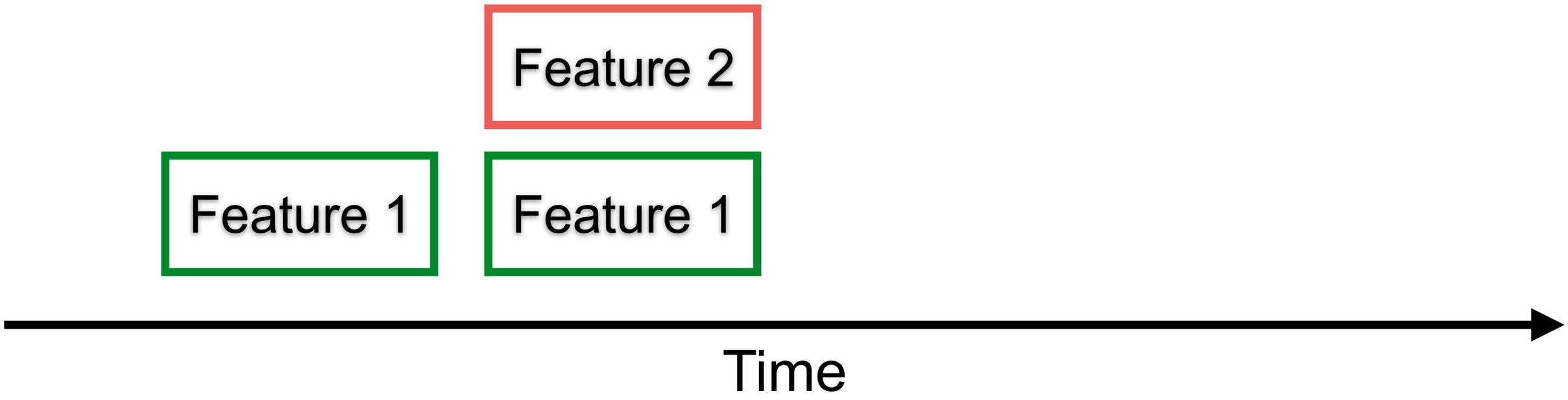
Iterative and incremental process

Done = coded and tested



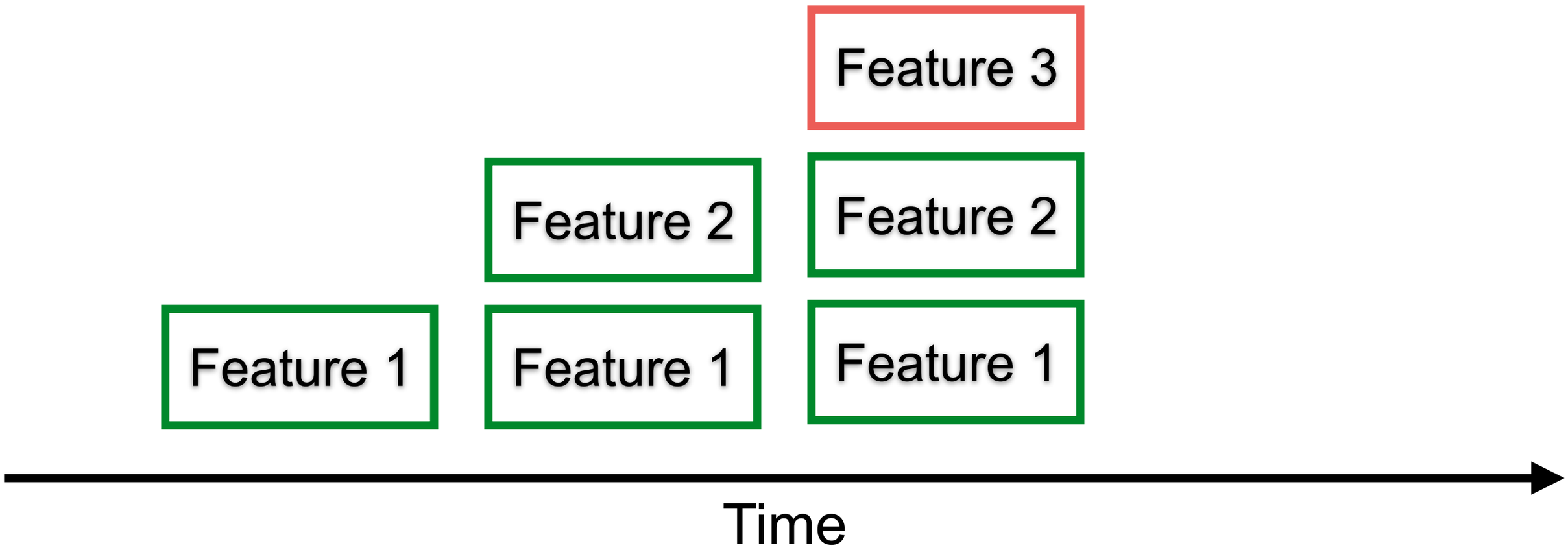
Iterative and incremental process

Done = coded and tested



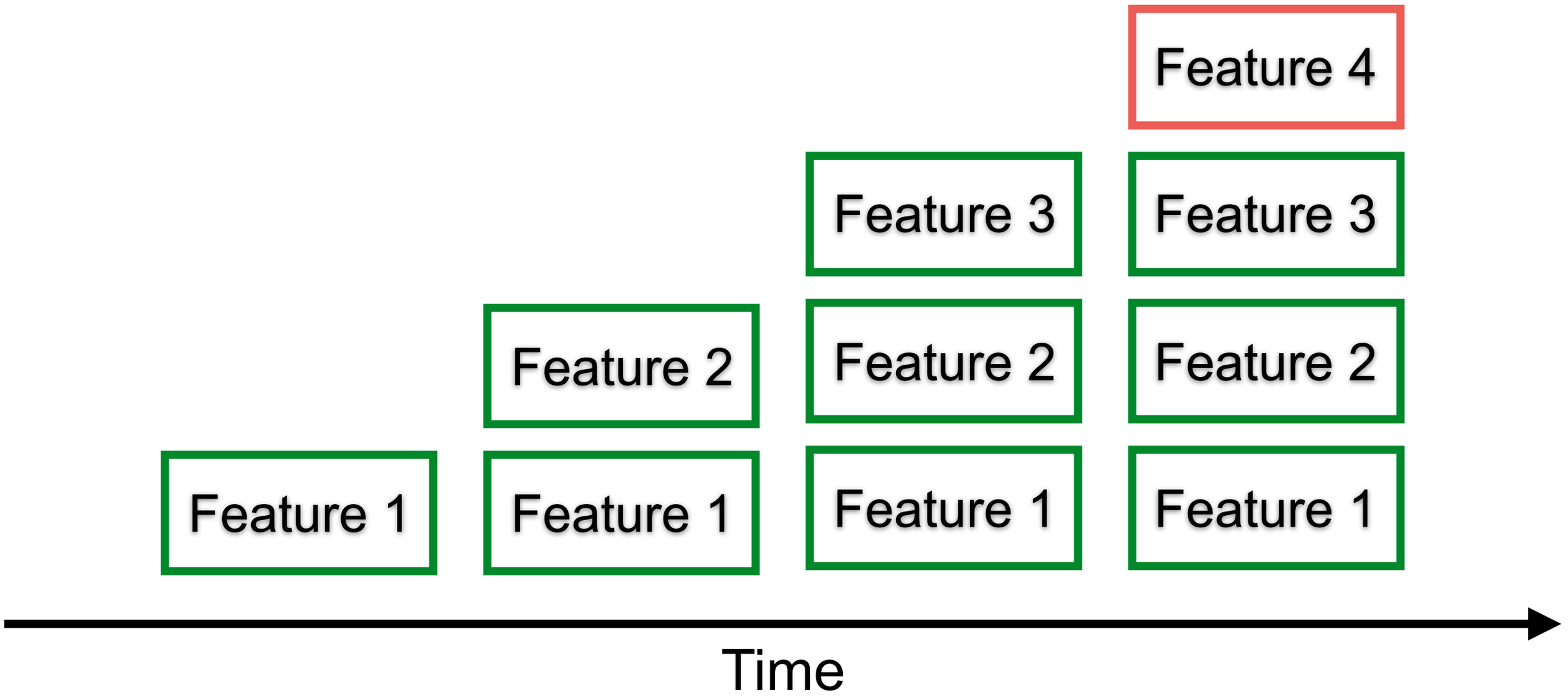
Iterative and incremental process

Done = coded and tested



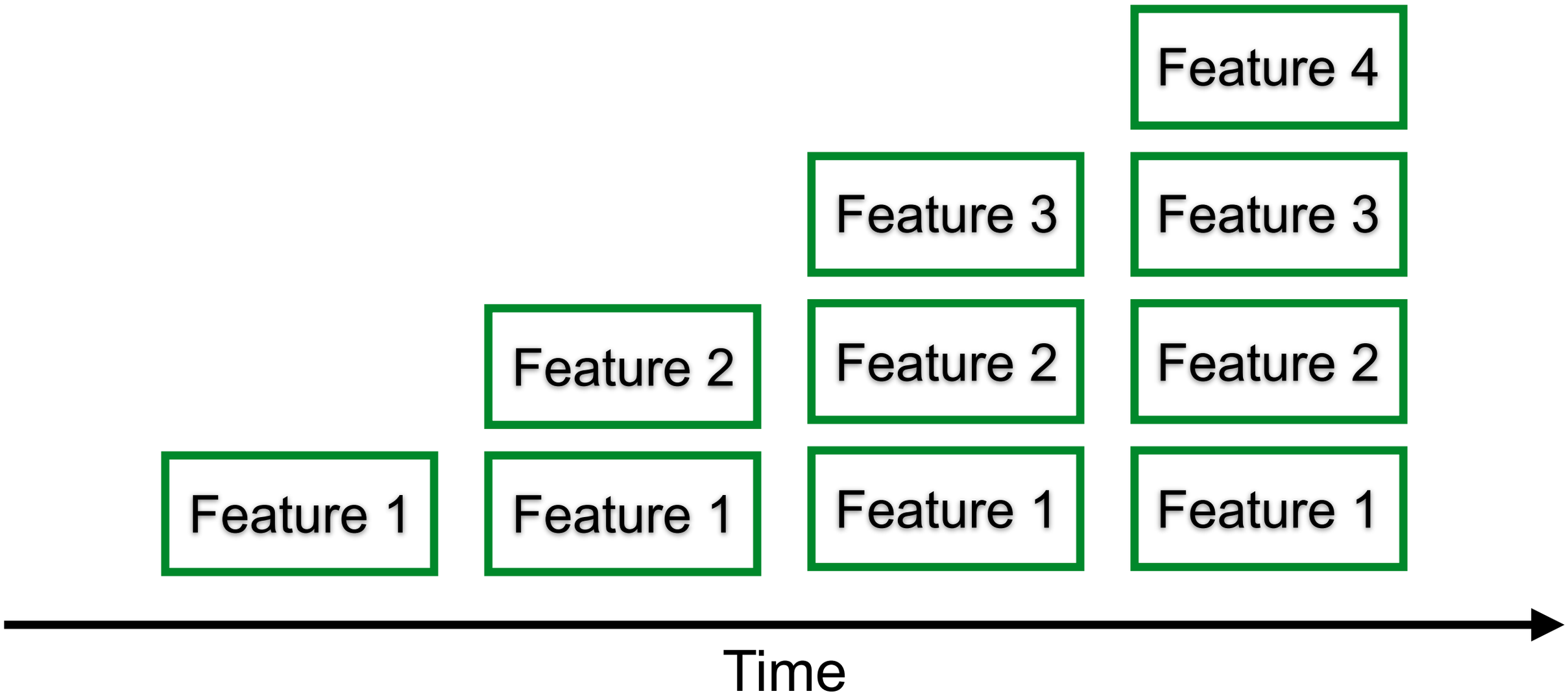
Iterative and incremental process

Done = coded and tested

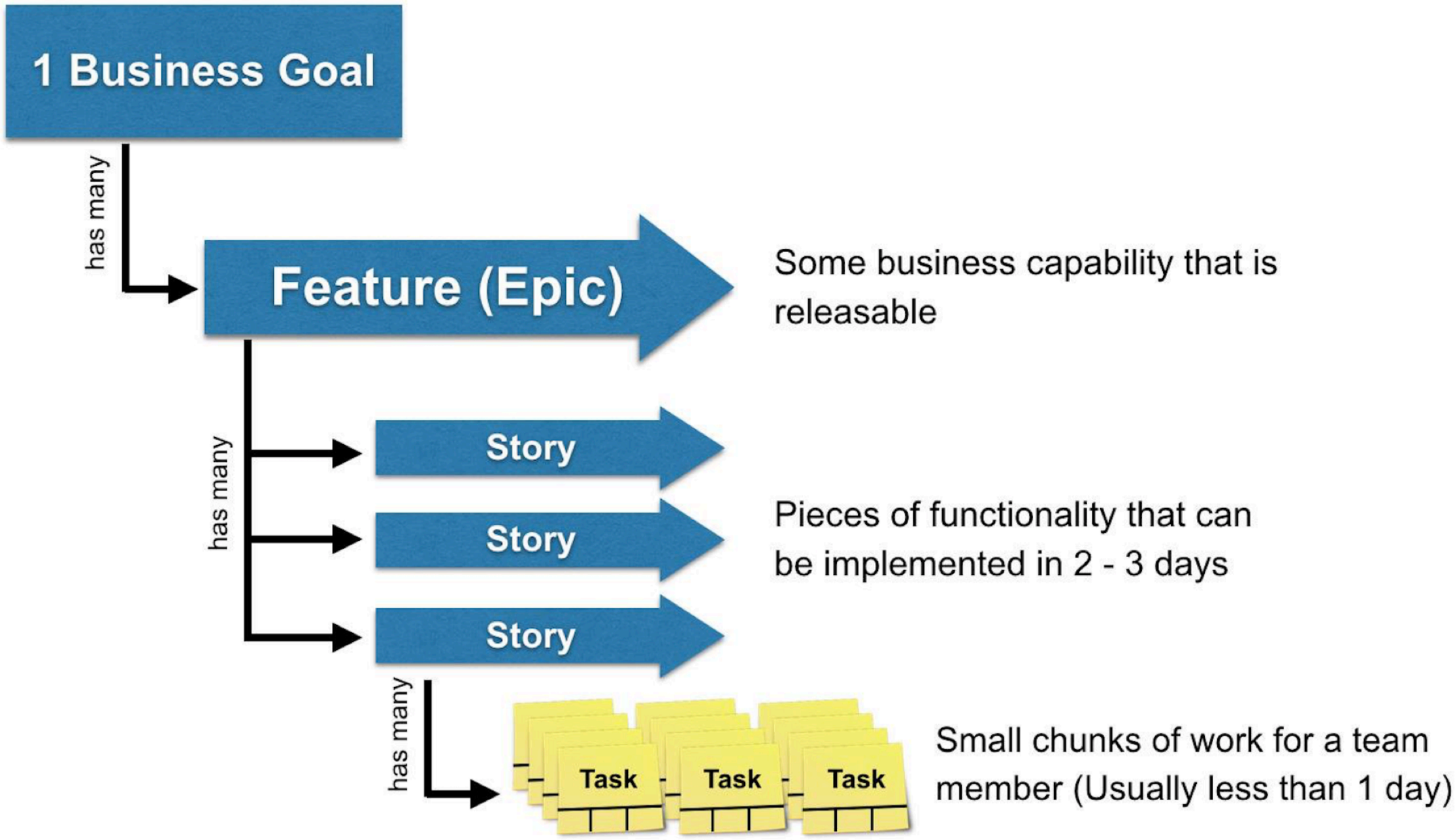


Iterative and incremental process

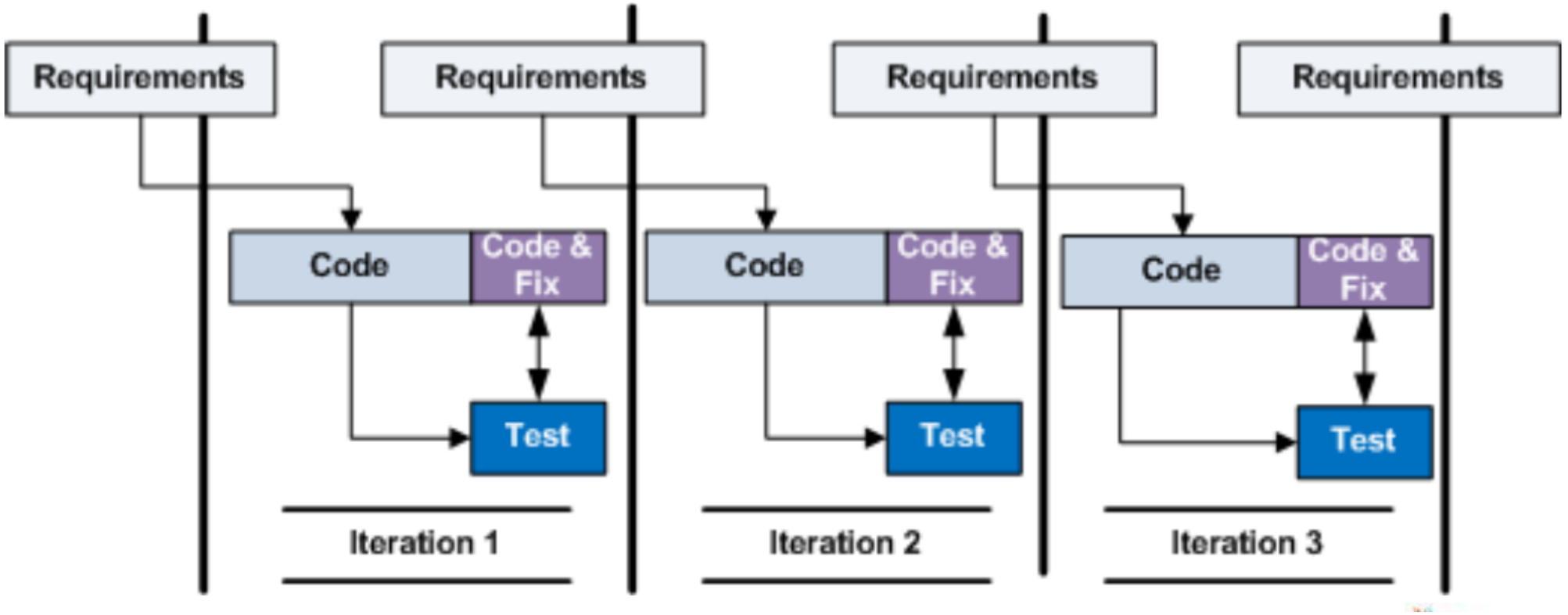
Done = coded and tested



Start with business goal to tasks



Mini Waterfall !!

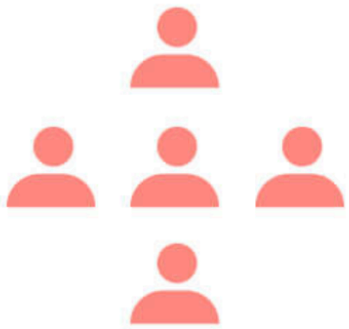


Whole team approach

Whole team approach

Functional

Common functional expertise



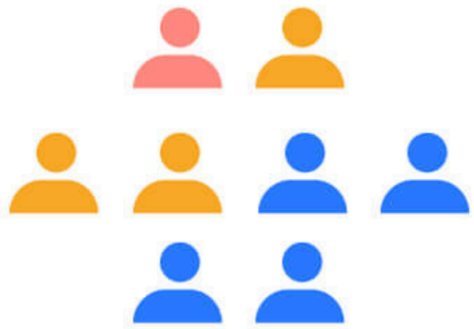
System analysts

Developers

Testers

Cross - Functional

Representatives from the various functions

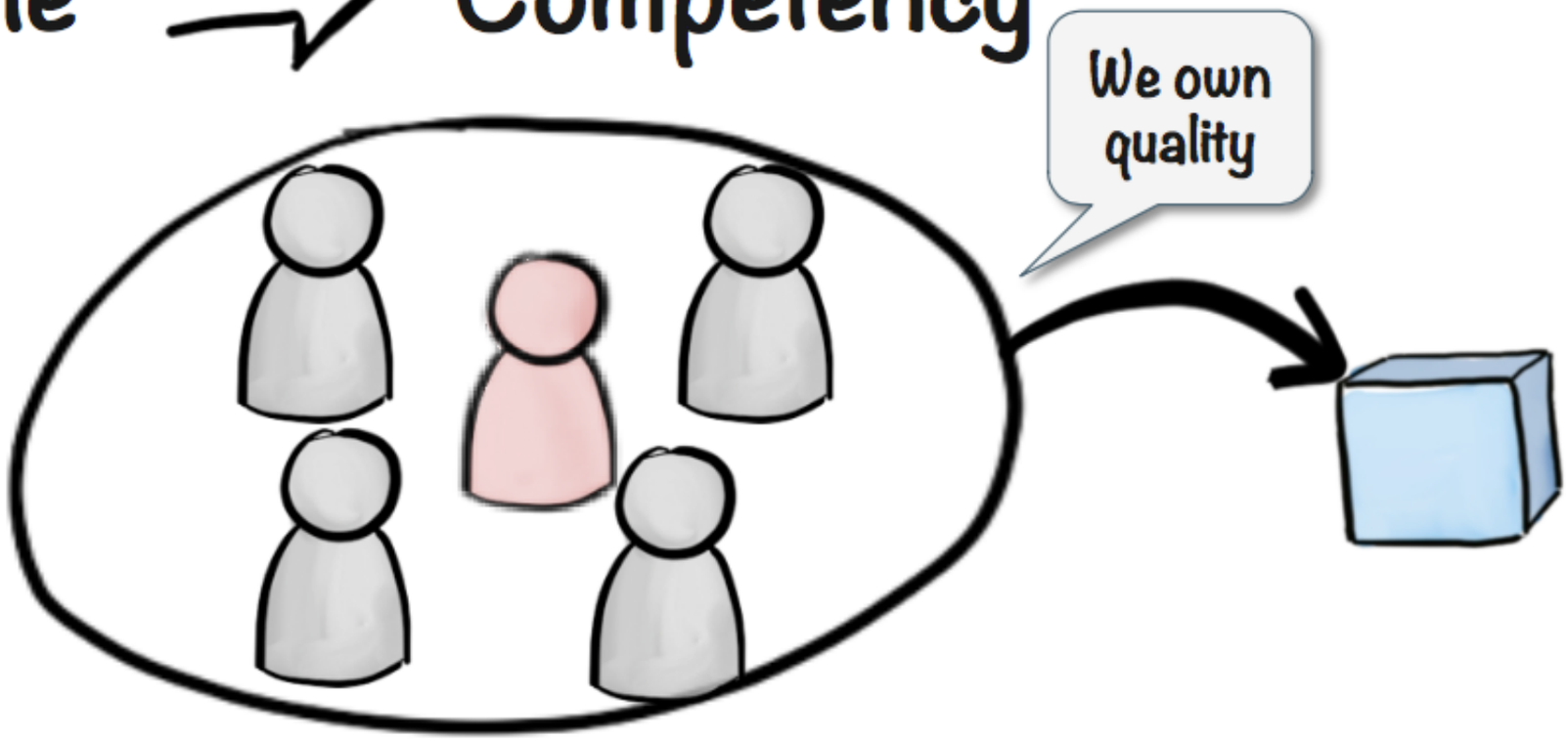


Development Team

Key success factors

Whole team solve problems
Whole team thinks about **testing**
Whole team **committed to quality**
Everyone **collaborates**

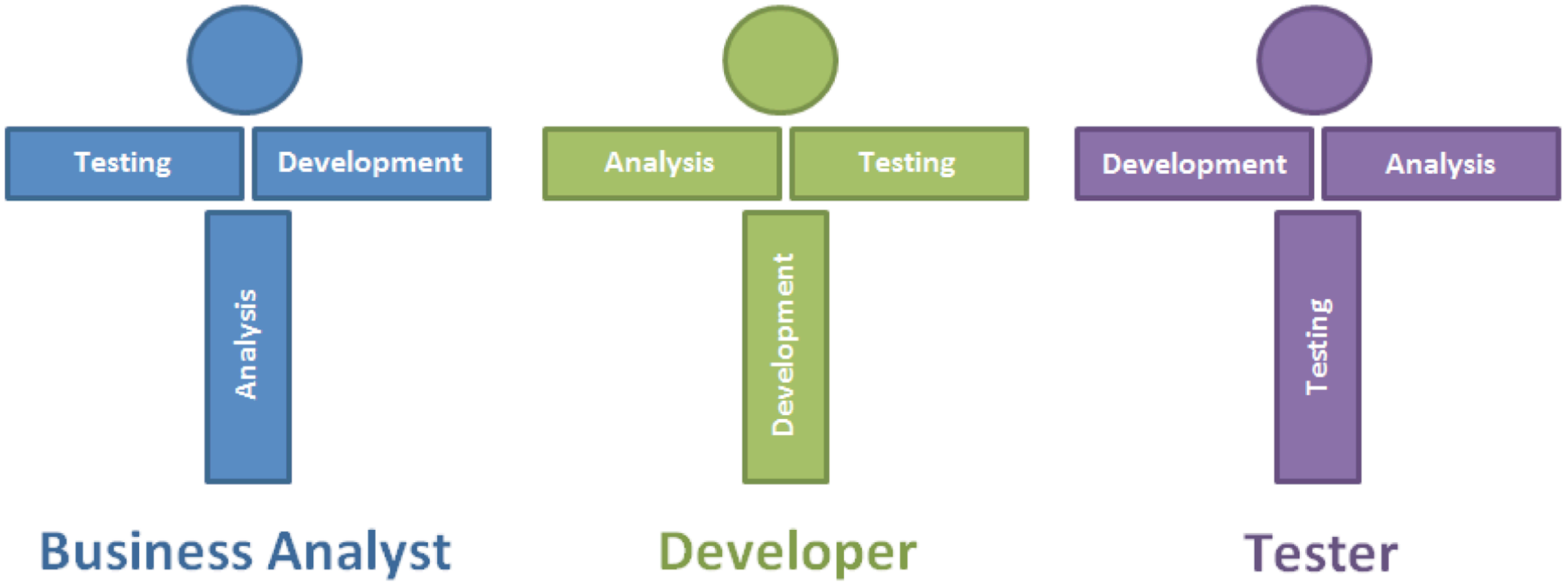
Role ⇒ Competency



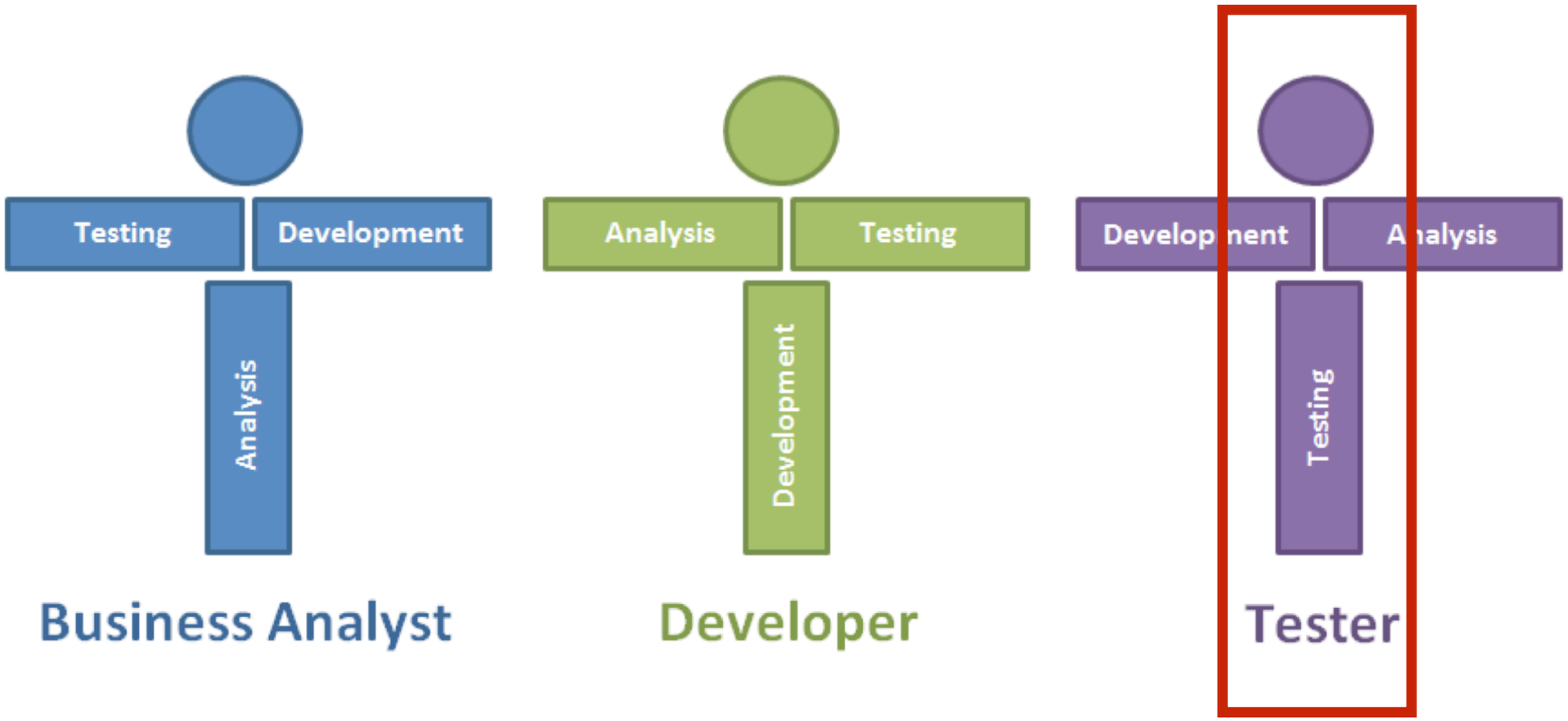
Cross-functional development team

QA = ~~Quality Assurance~~
= Quality Assistance

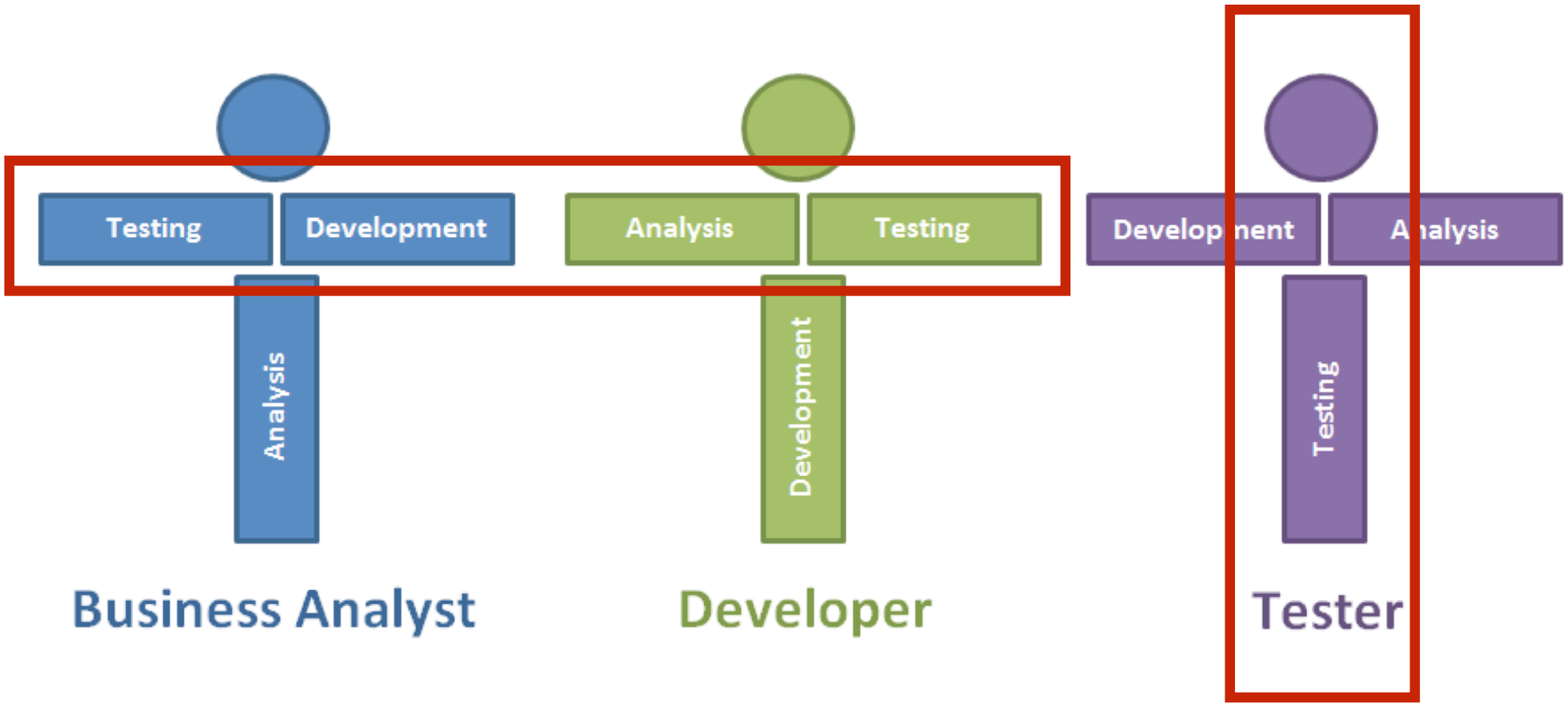
T-shaped skills



Quality assistance



Quality assistance



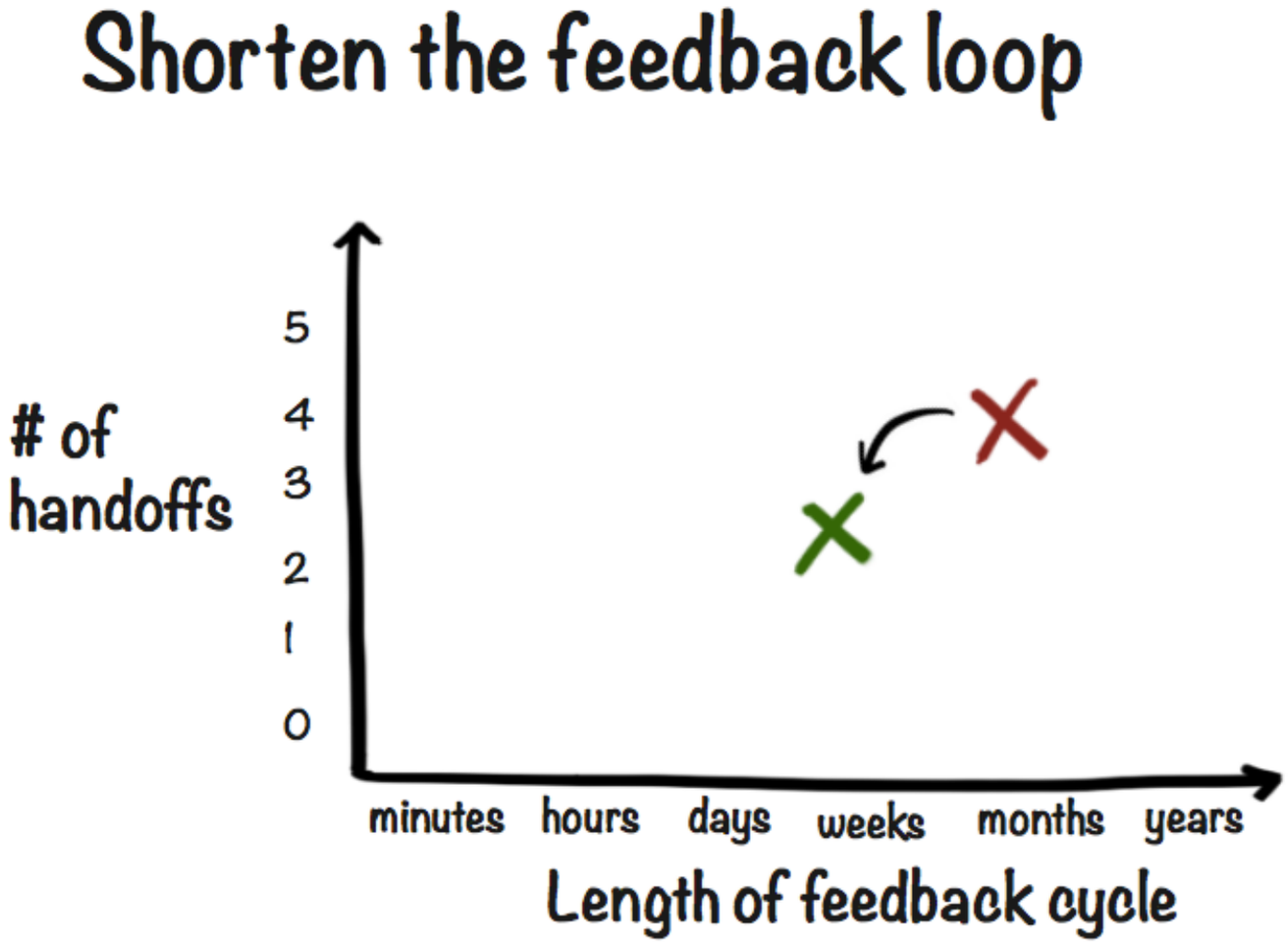
Test is activity

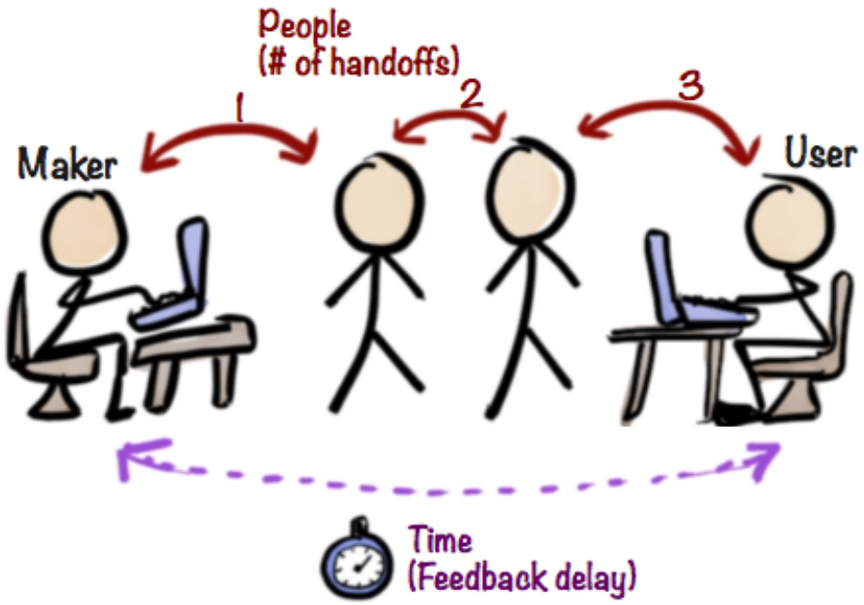
~~Test phase~~

~~Test team~~

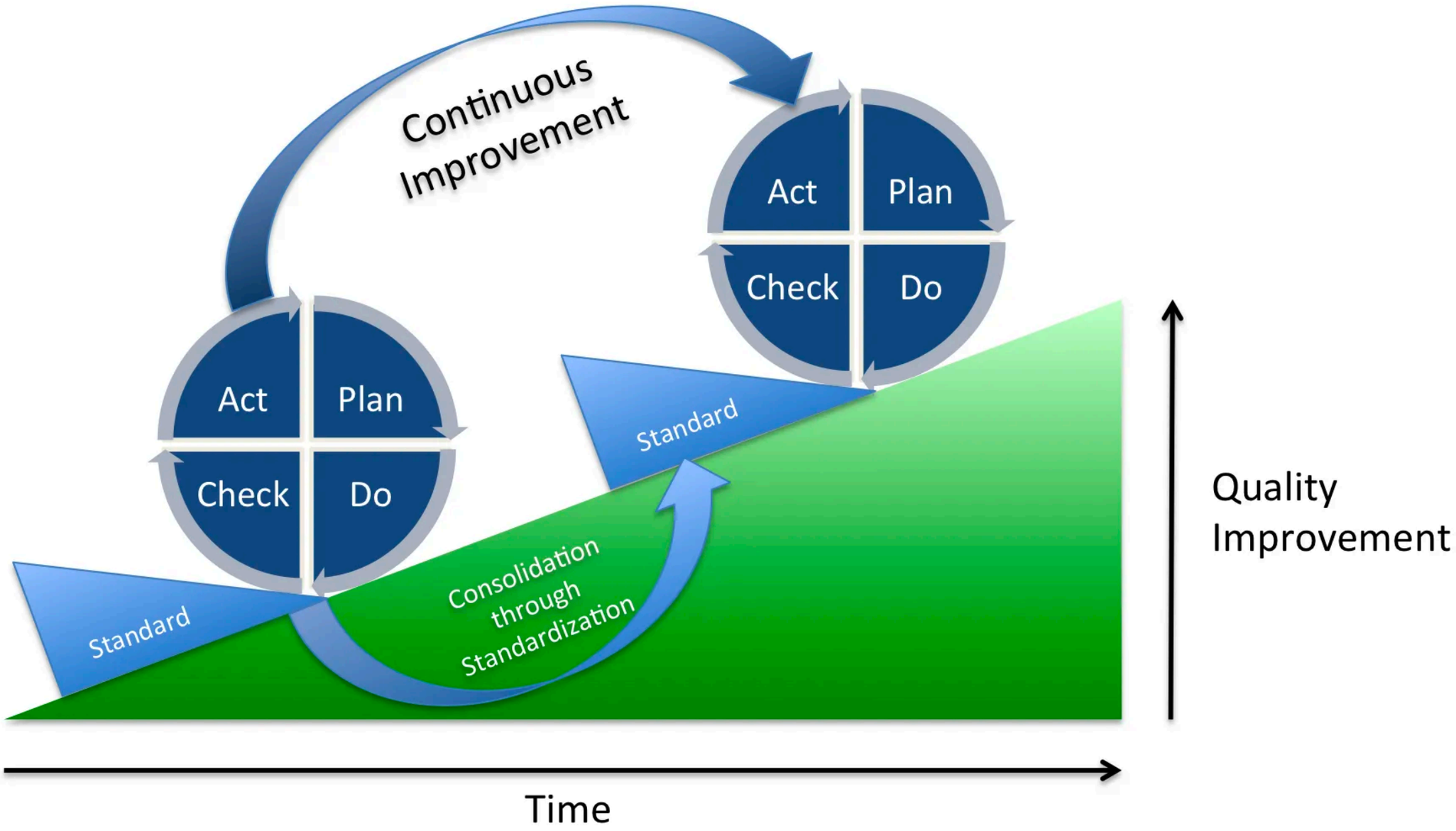
~~Tester role~~

Fast feedback loop

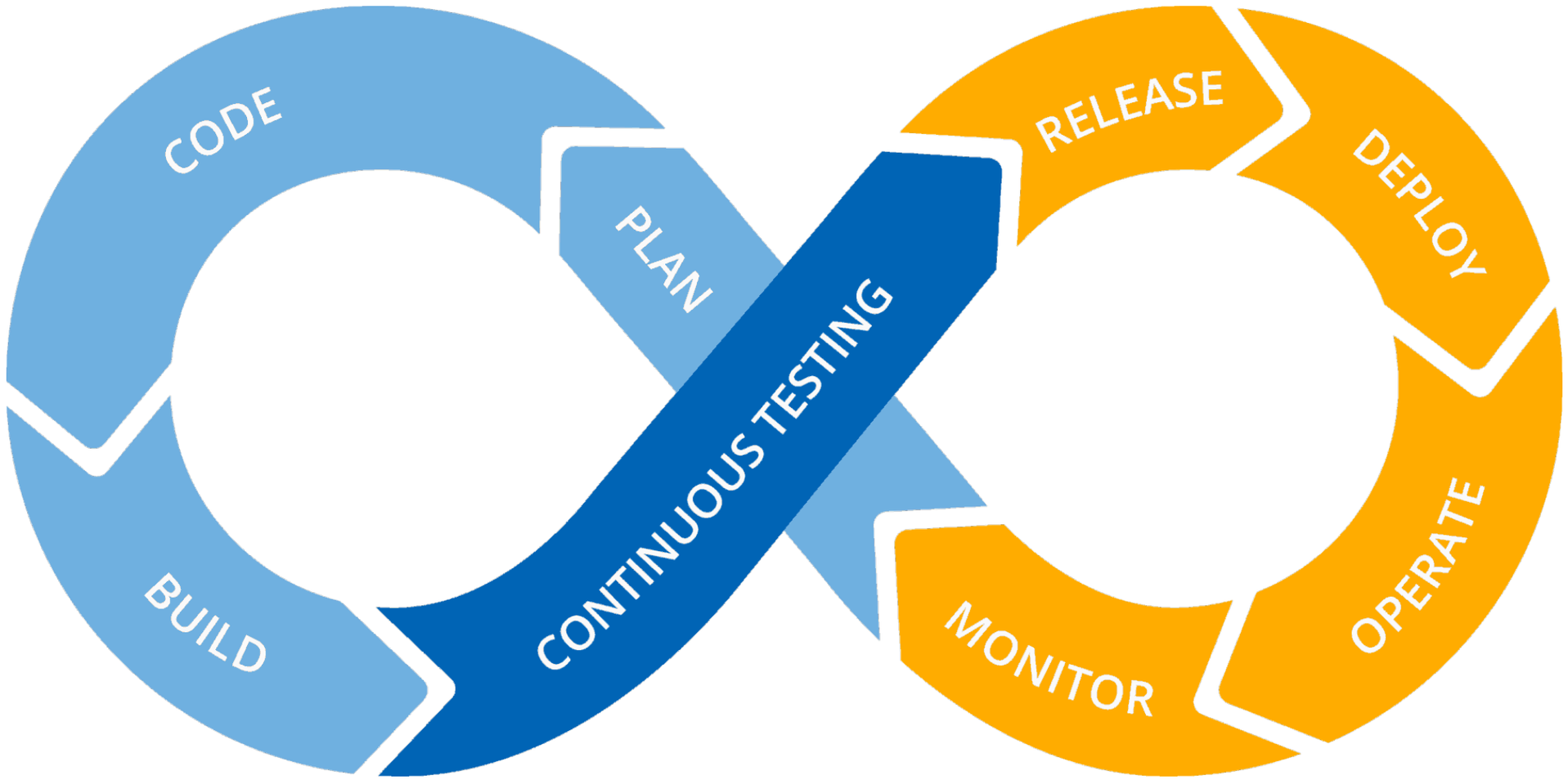




Continuous improvement

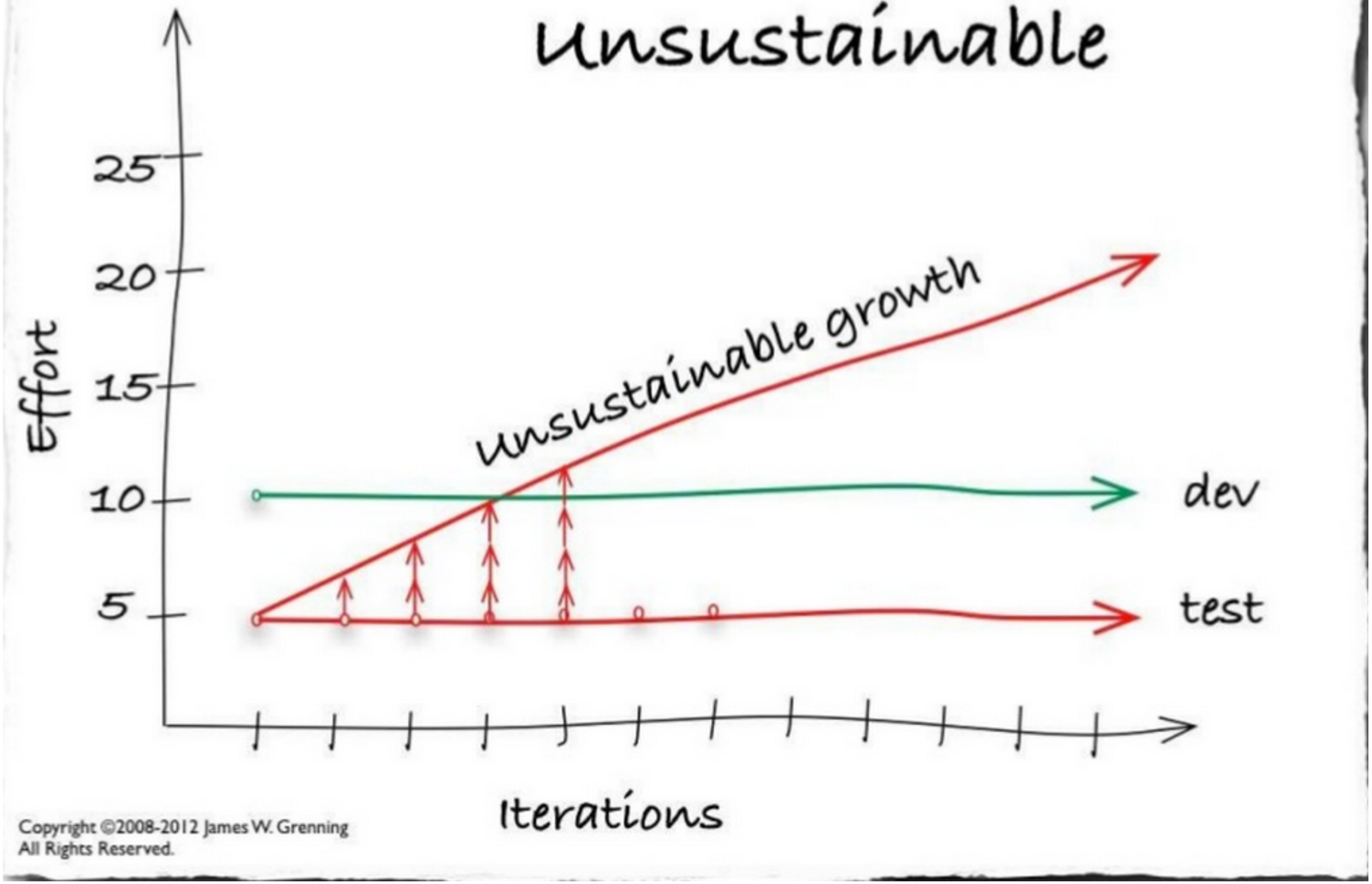


Continuous testing

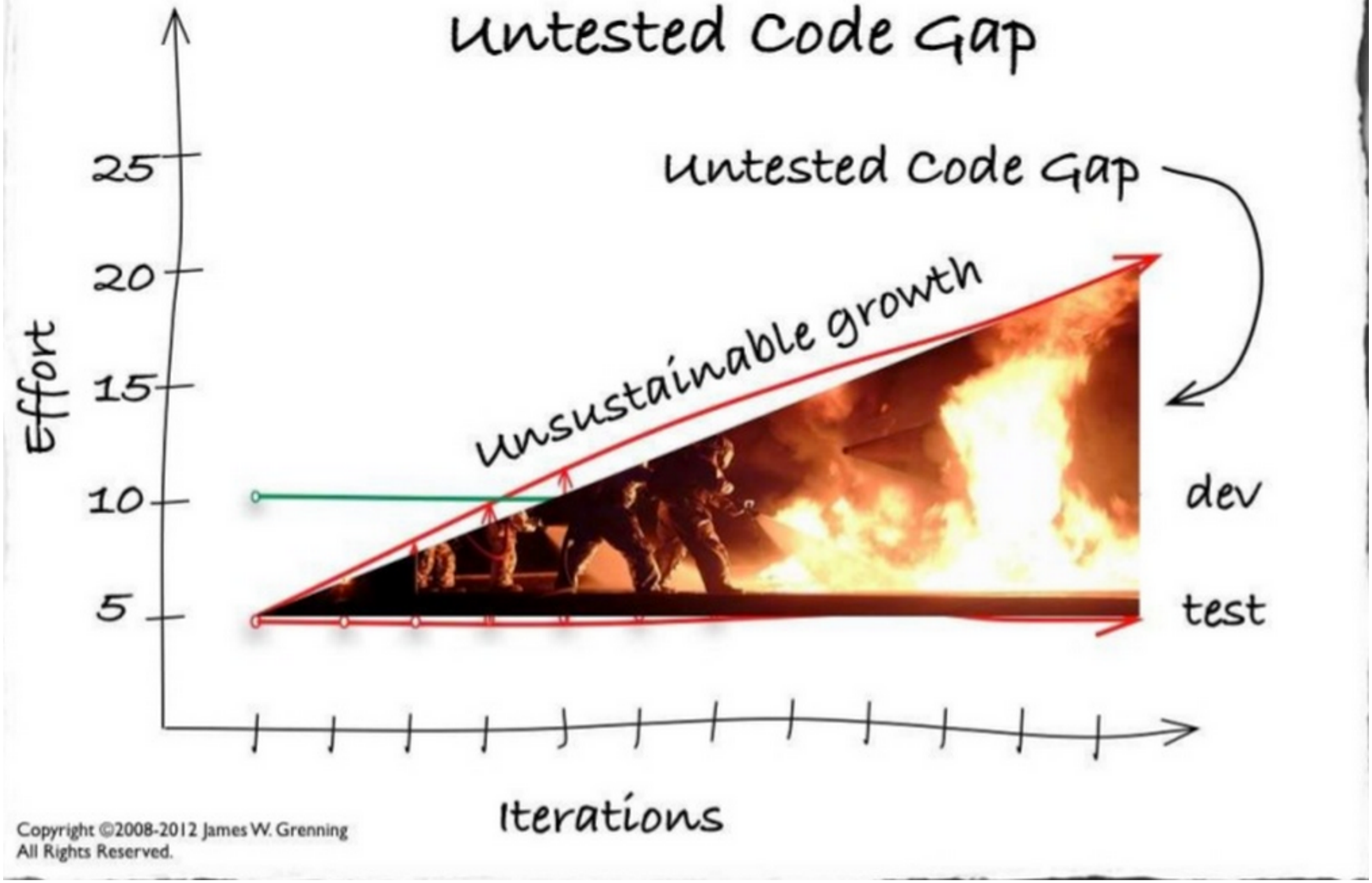


Manual testing

Manual Test is Unsustainable



Risk Accumulates in the Untested Code Gap



We need automation !!

Why we need automation ?

Manual checking take too long
Manual checks are error prone
Free people to do their best work (testing)
Living document
Repeatable and save time

BUT … why are not automating ?

Fear

Cost

Change frequently

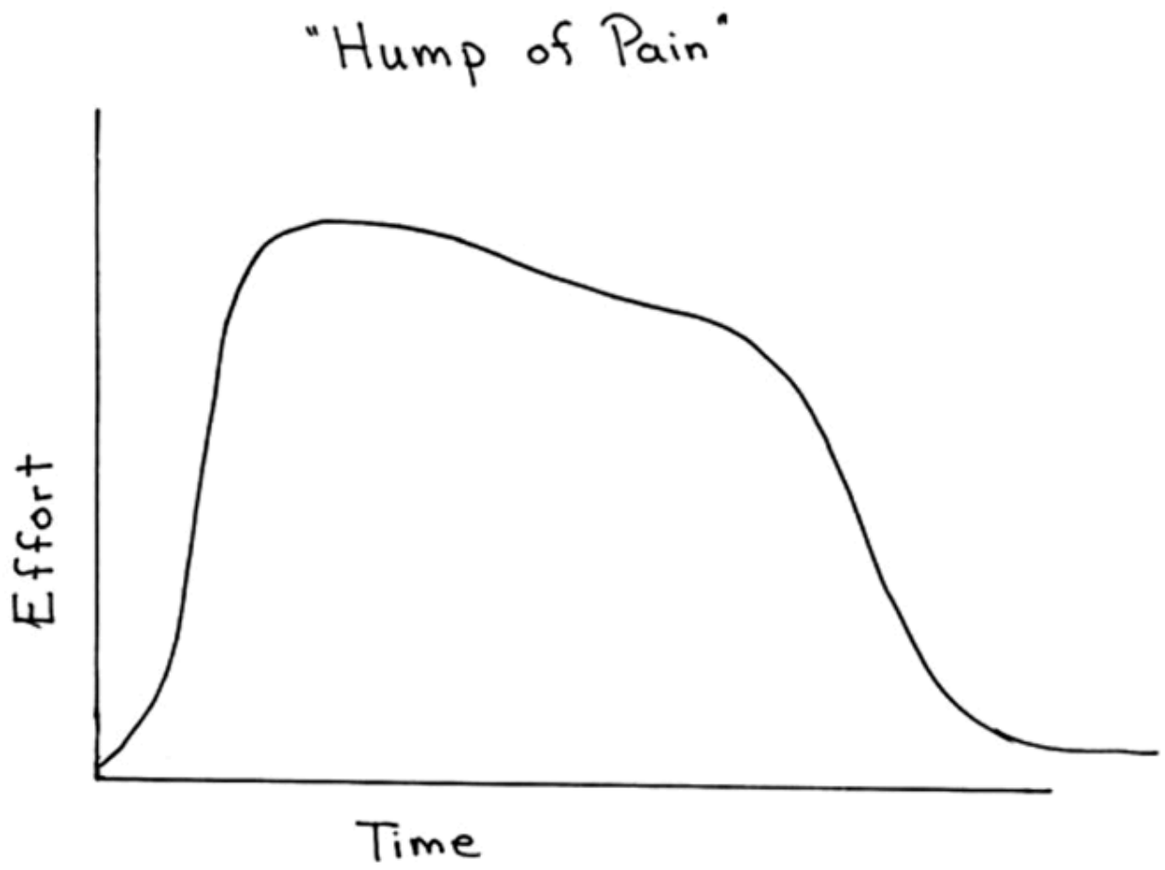
Deadline-Driven Development

Knowledge

BUT … why are not automating ?

Initial investment !!

Hump of pain

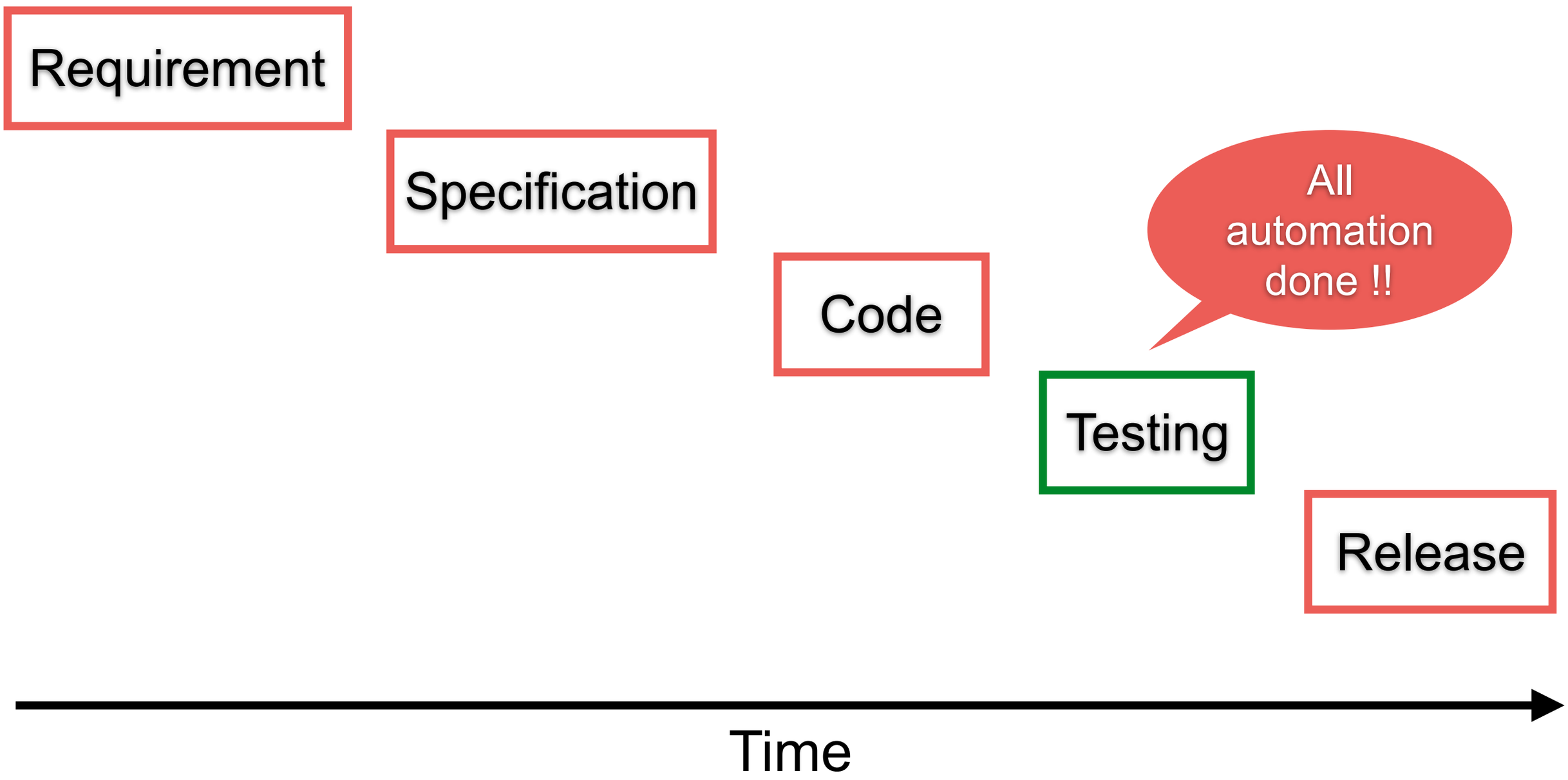


Initial investment !!

Working with legacy system
Side-effect after changed code
Need new tools, infrastructure and time

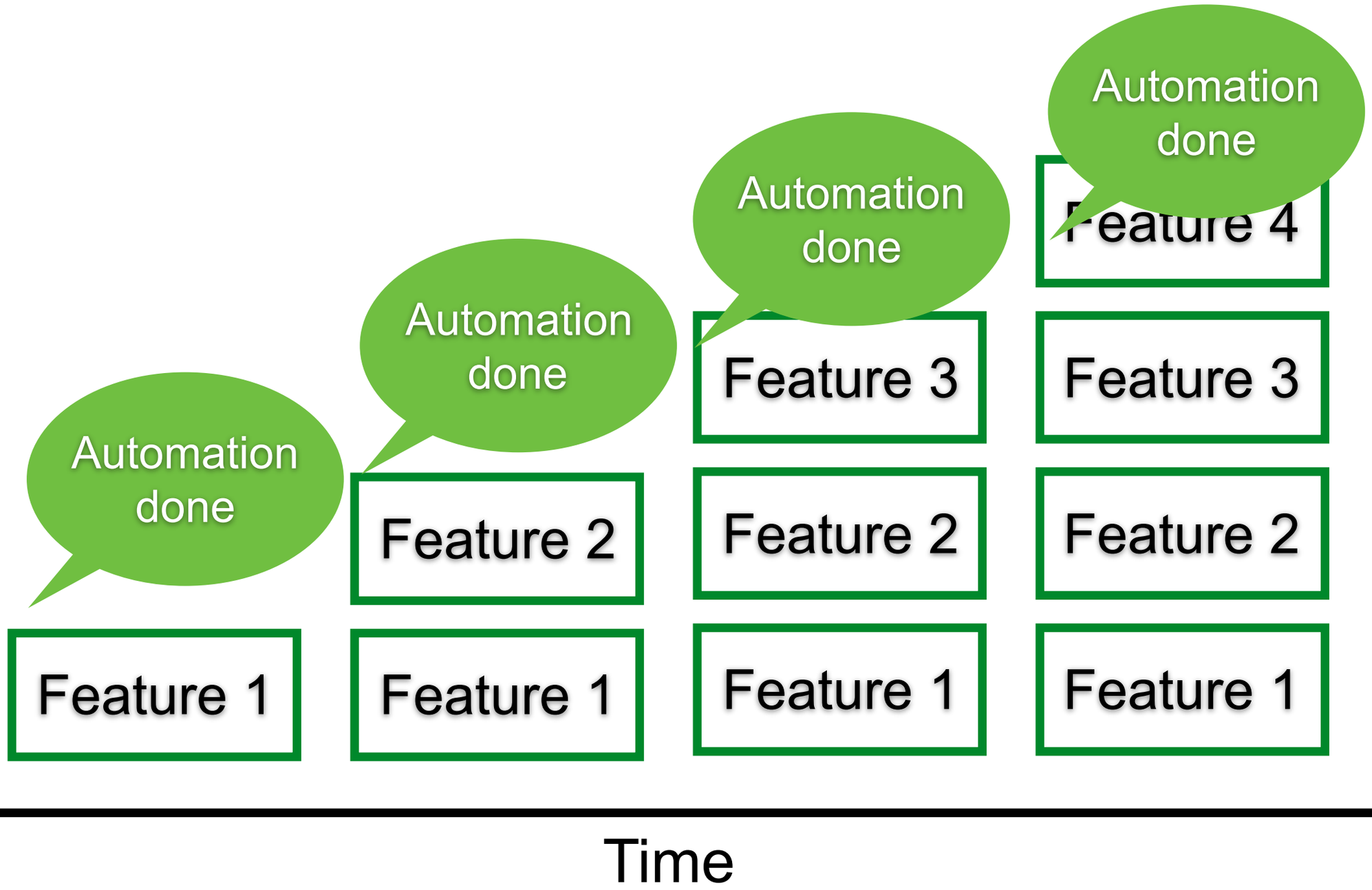
Automation test strategy

Sequential process

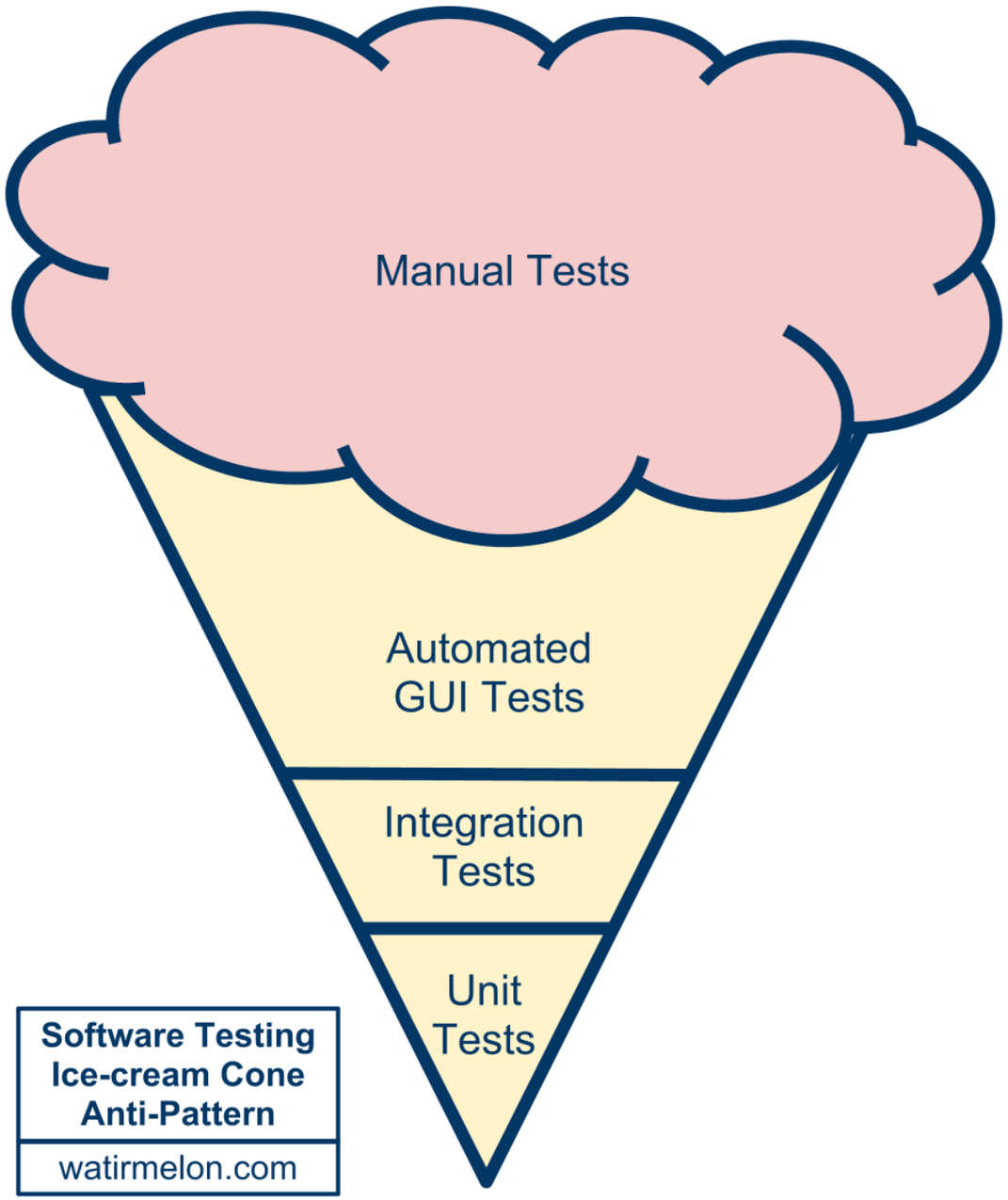


Iterative and incremental process

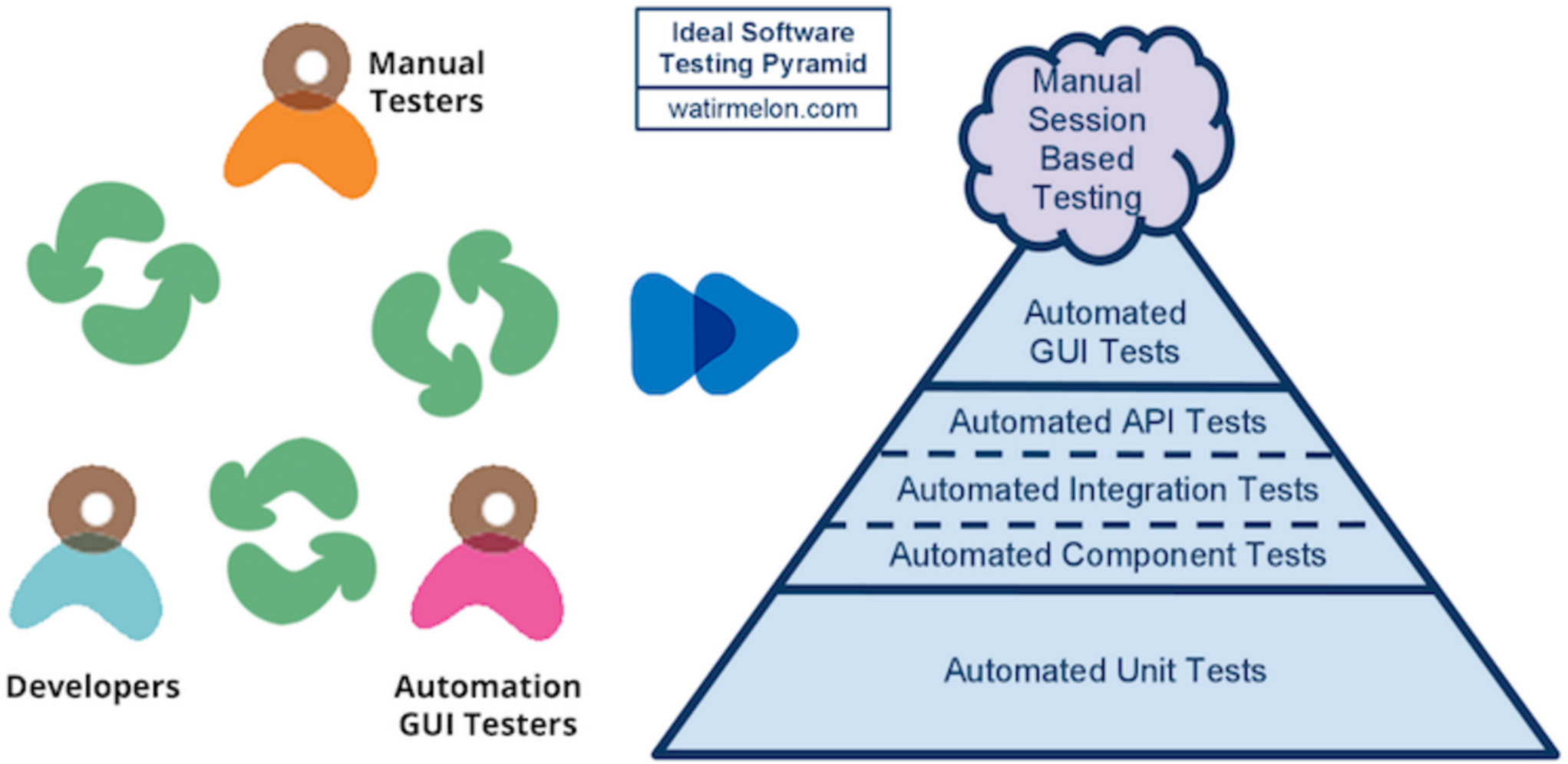
Done = coded and tested



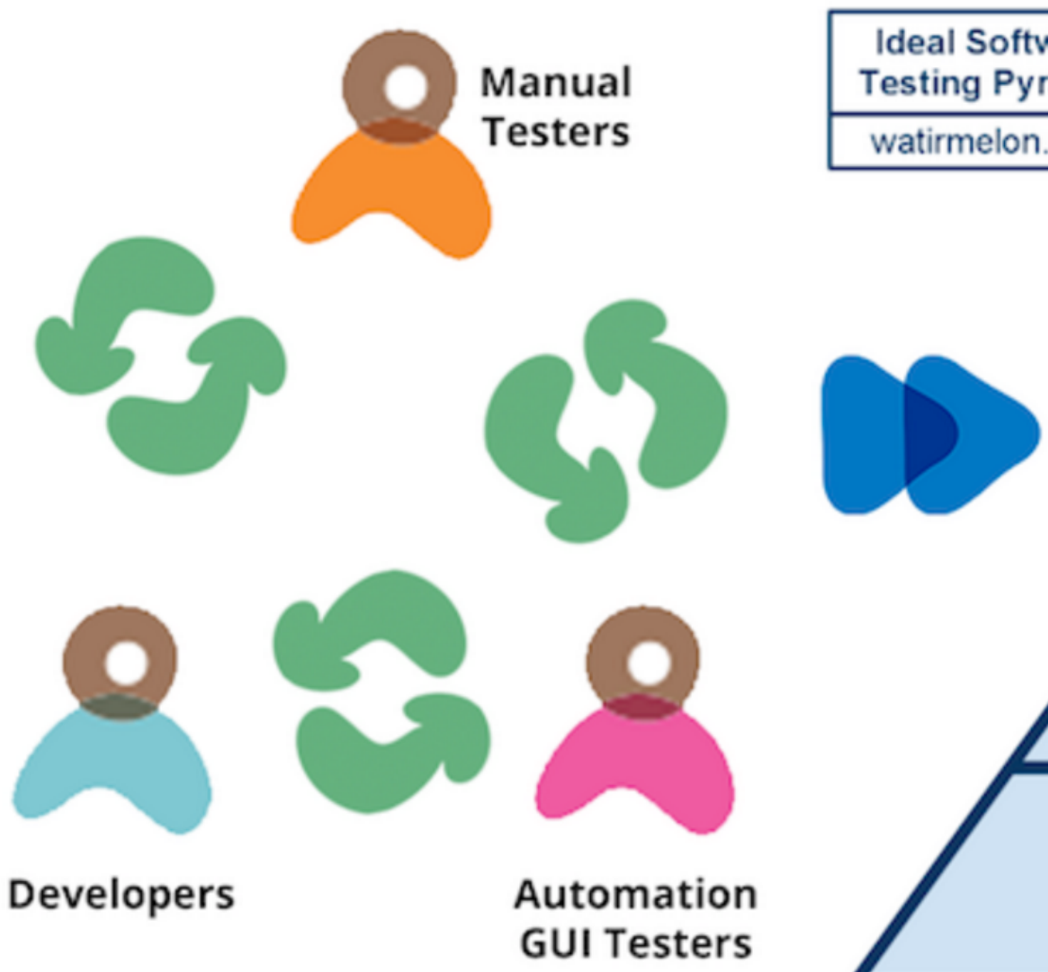
Ice cream testing

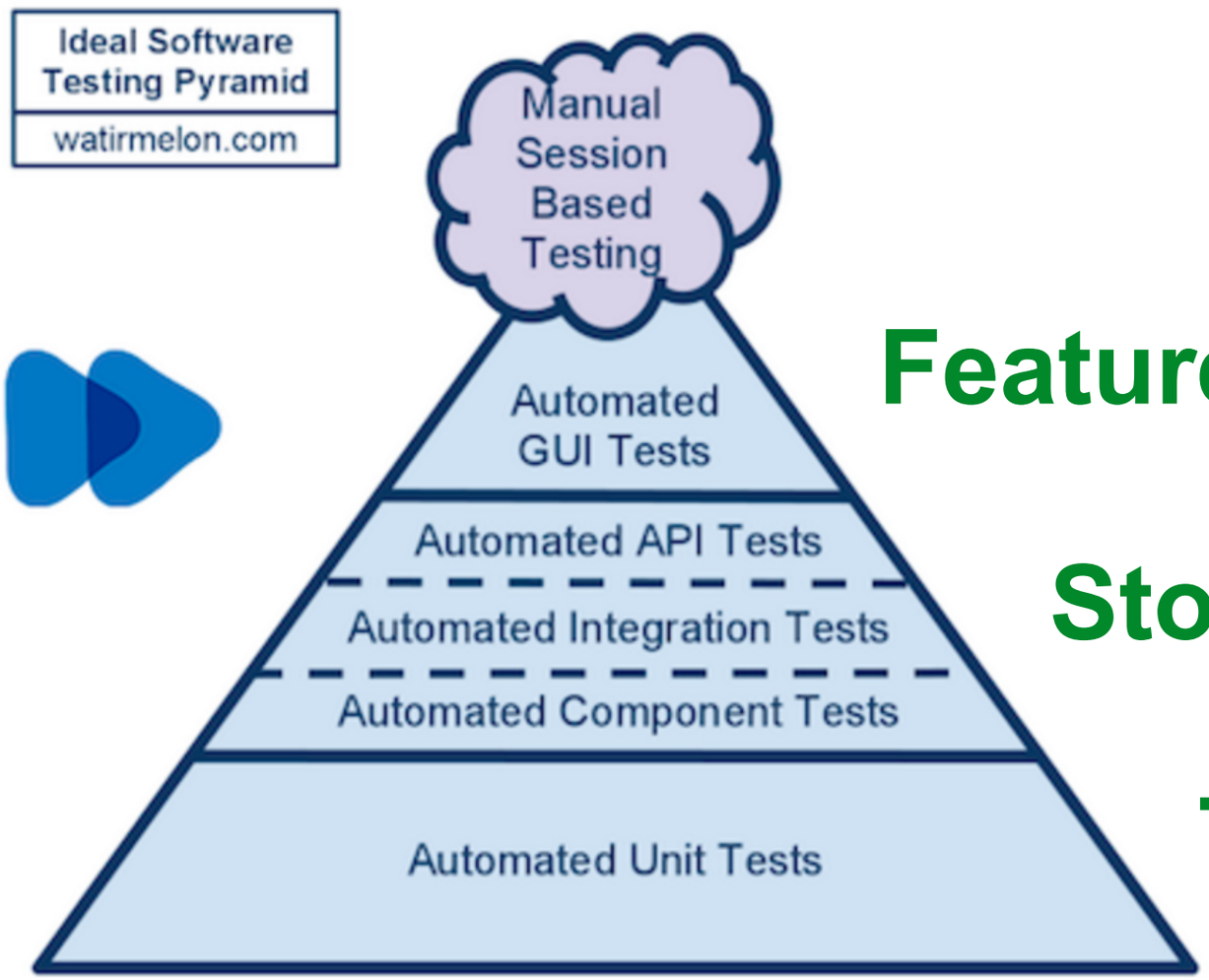


Pyramid testing



Pyramid testing



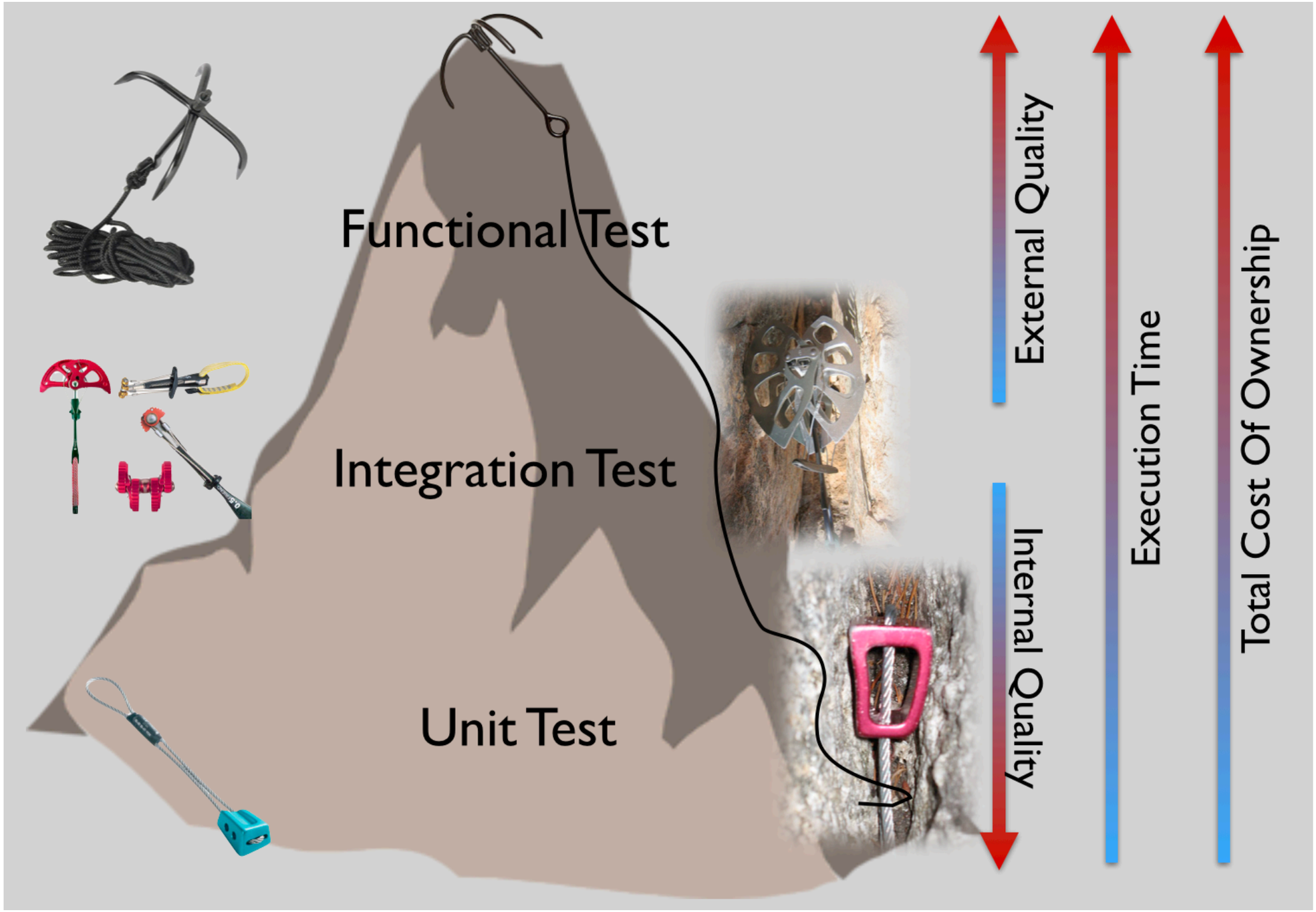


Feature

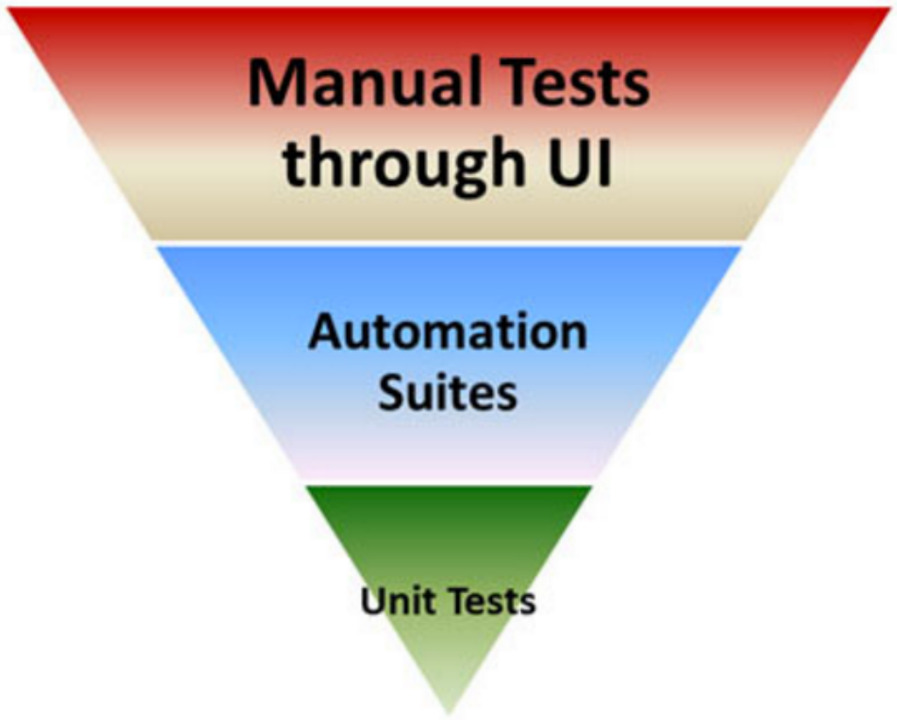
Story

Task

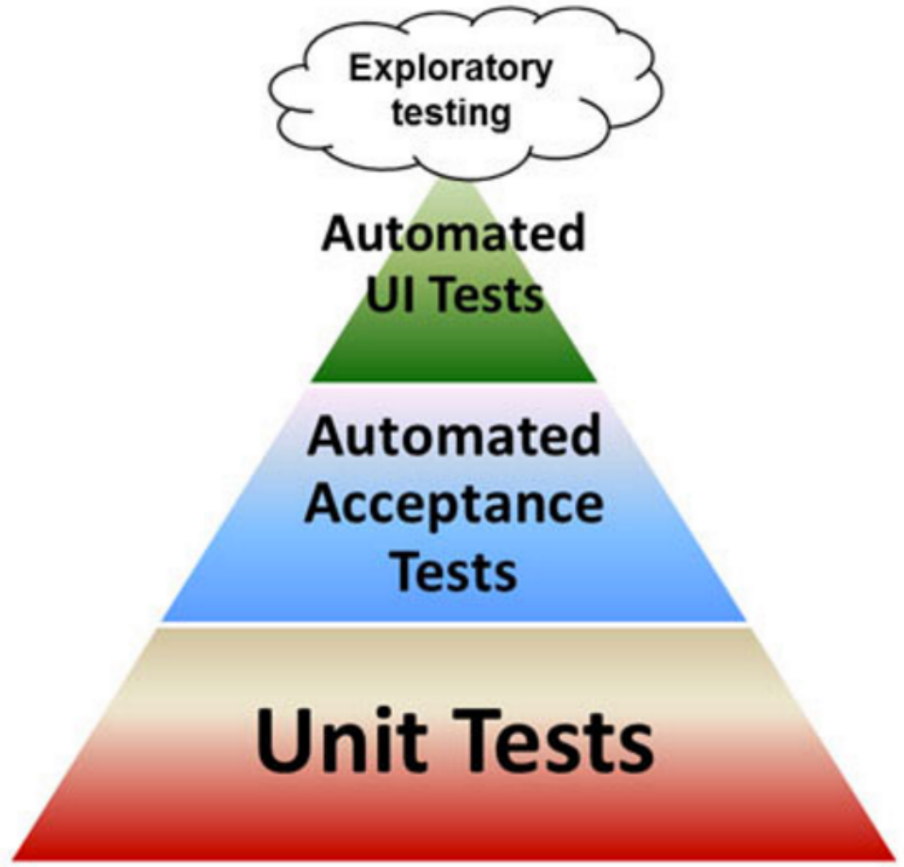
Pyramid testing



Prevention over find bugs



Traditional
(find bugs)



Agile
(prevent bugs)

Mind-set switch

Instead of

We are here to find bug

We are here to ensure requirement are met

We are here to break the software

Mind-set switch

Instead of

We are here to find bug

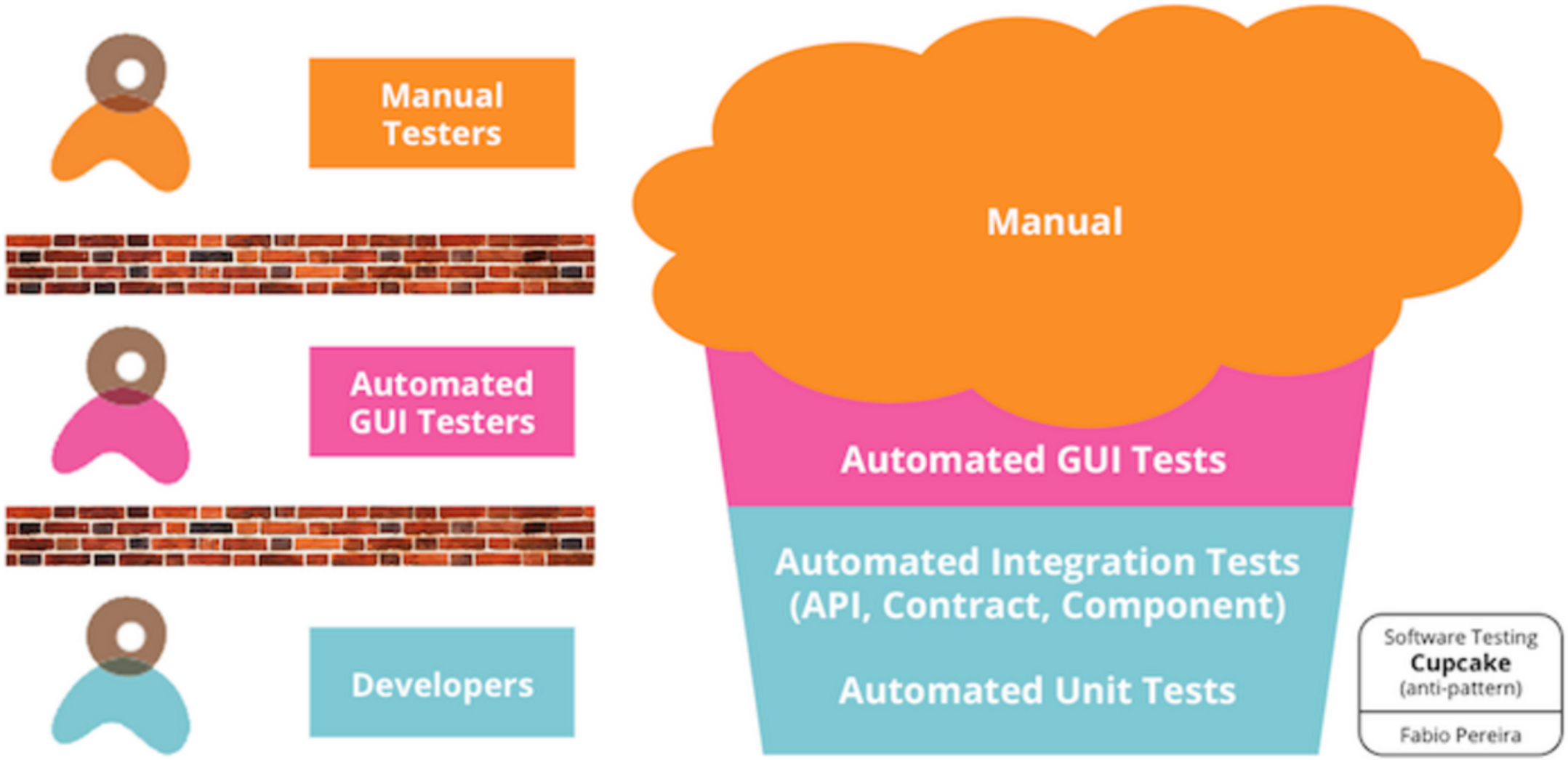
We are here to ensure requirement are met

We are here to break the software

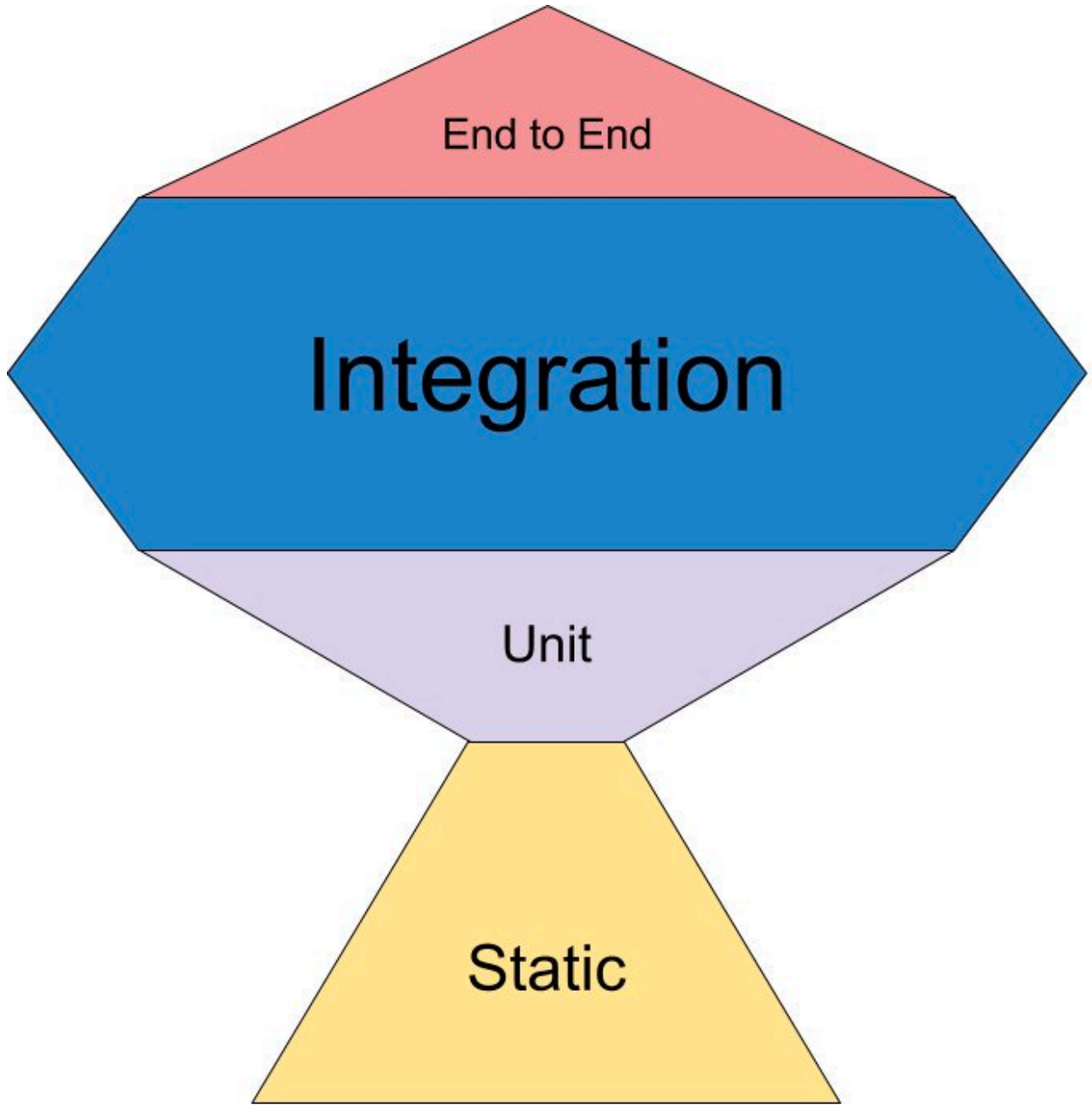
Think

What can I do to help deliver the software successfully !!

Cup cake testing



Trophy testing



Important to remember

Important to remember

The automation pyramid is a **tool**
To talk about **automation tasks**,
How to **prioritise** them and **who will help**

Way to make automation **visible** to the whole team

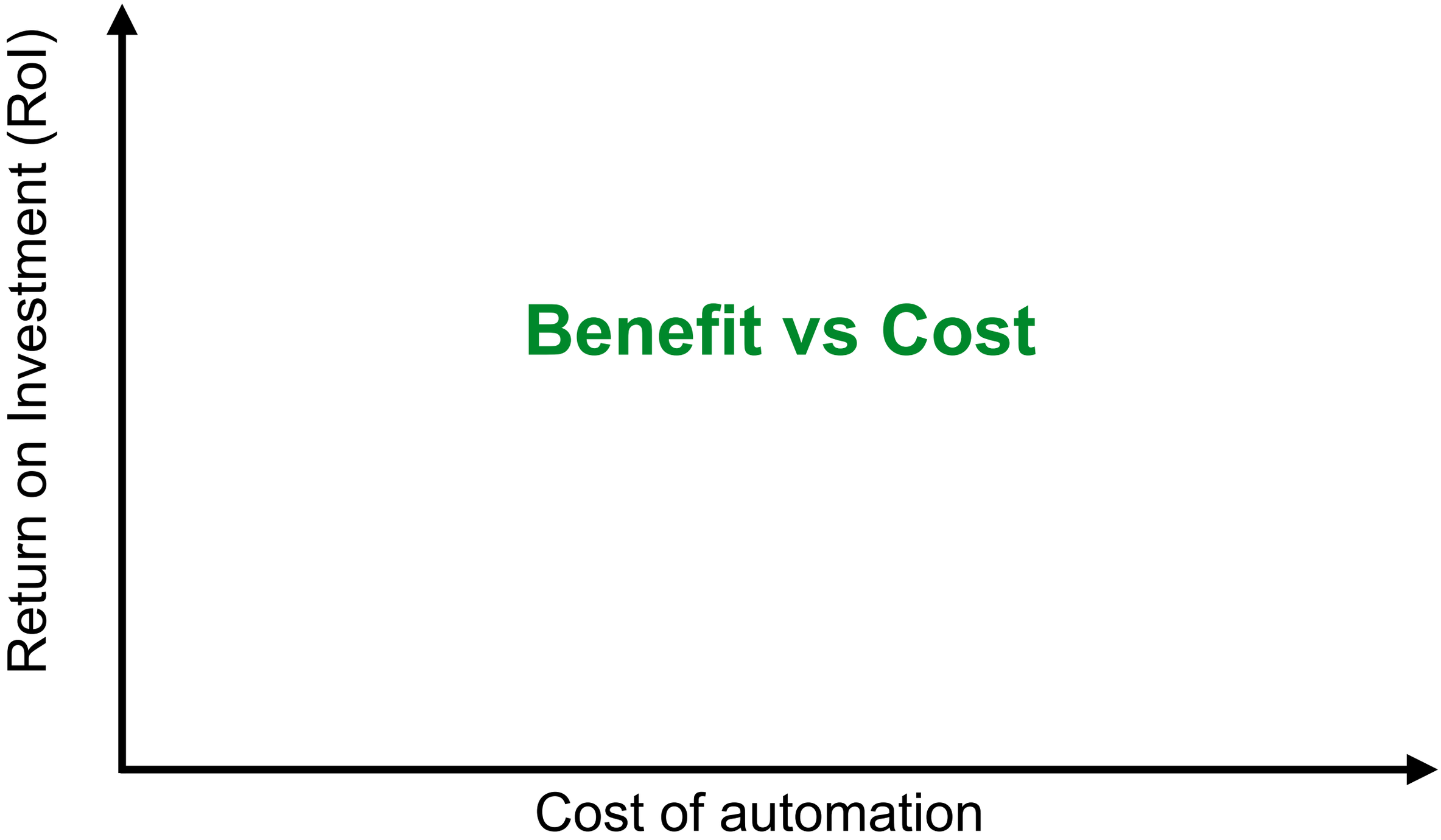
Where to start ?

What are the **trade-offs** between automating and manual testing ?

What are the **biggest obstacles** ?

Time/Tools/System/People

Automation decision guideline



What should not try to automate ?

Tests are really expensive

One-off tests

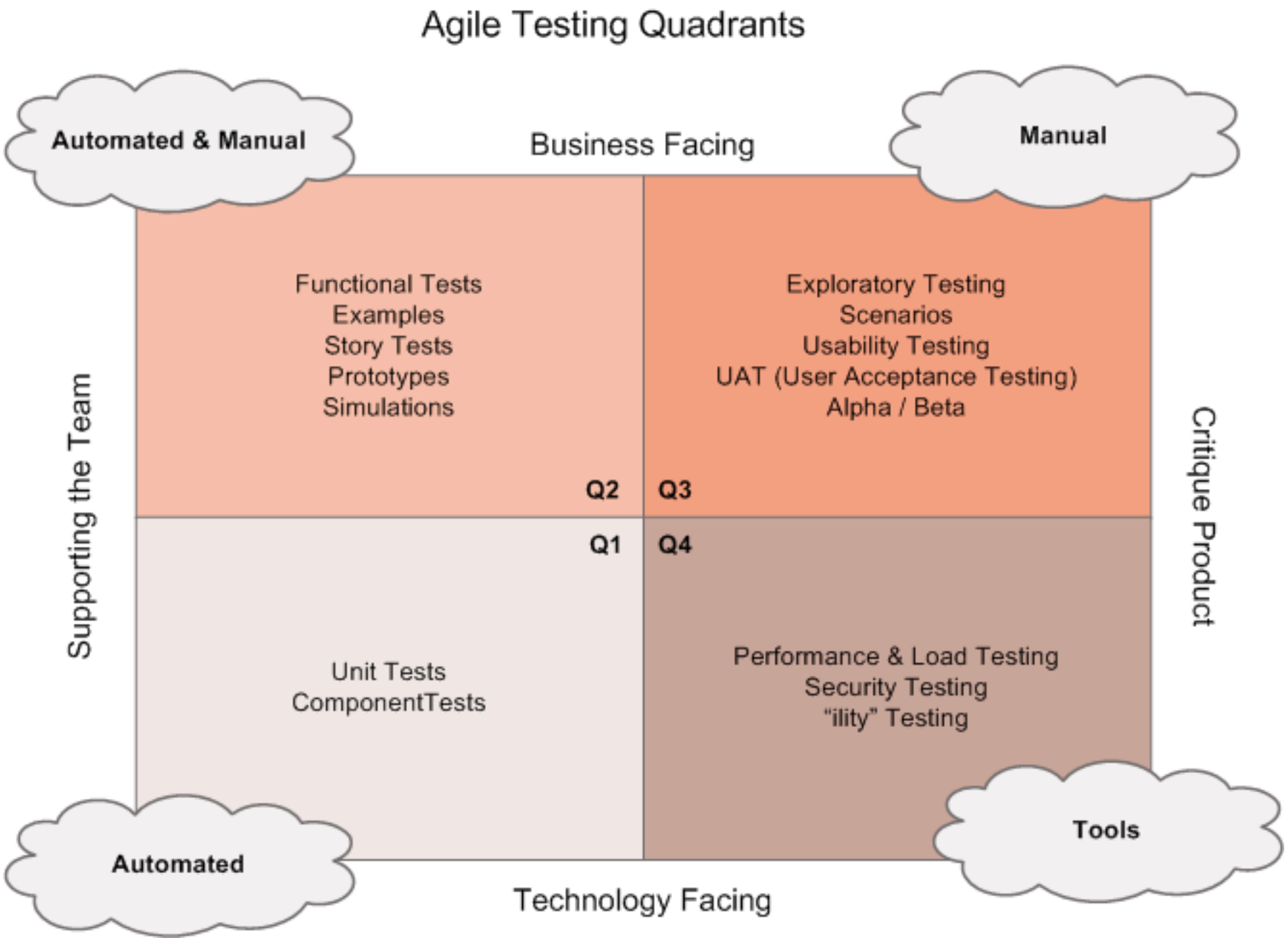
Usability tests (look and feel)

What should be careful ?

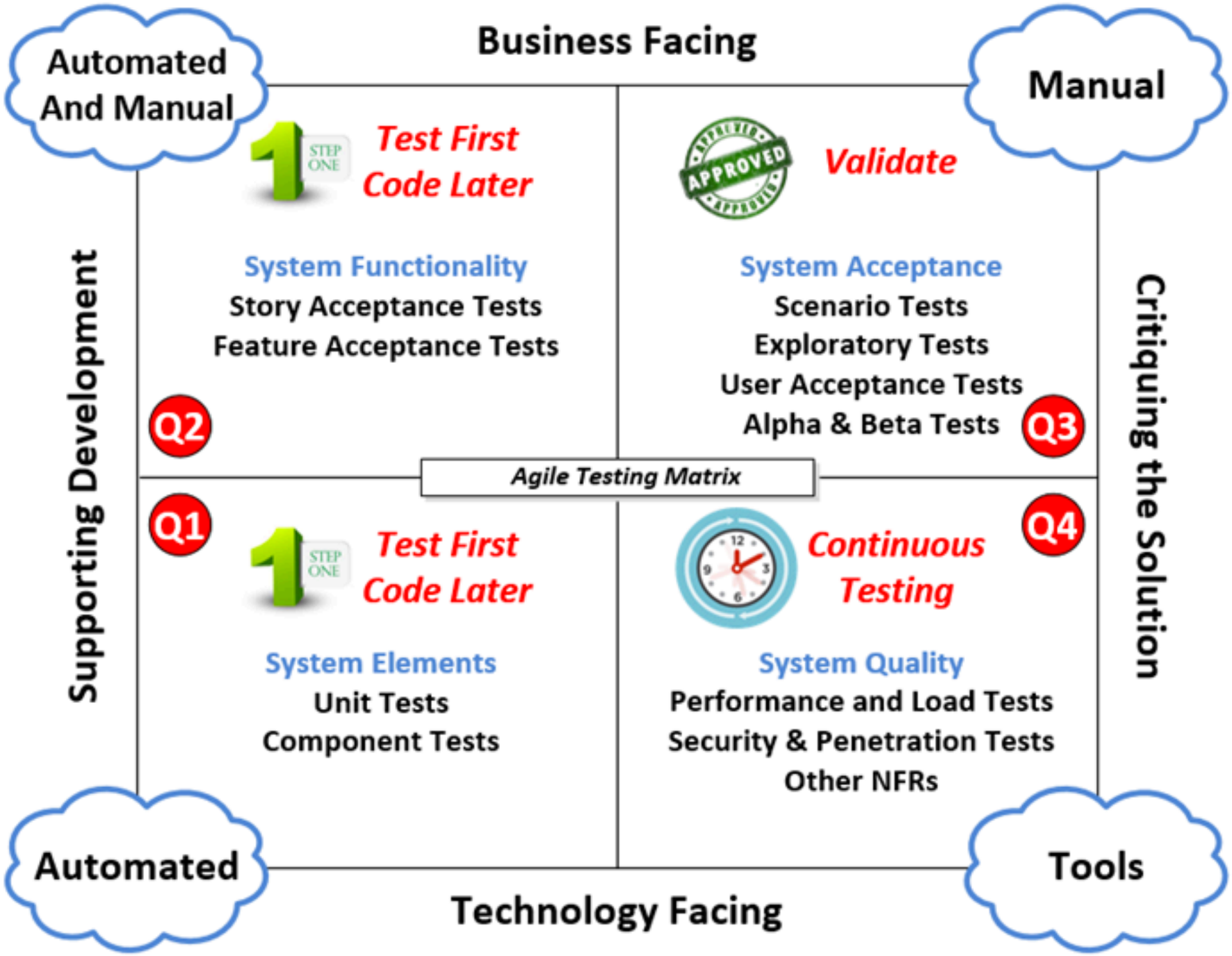
Automating end-to-end tests
User Interface are slow
Working with database
Working with external system
Automating every paths

Testing Quadrants

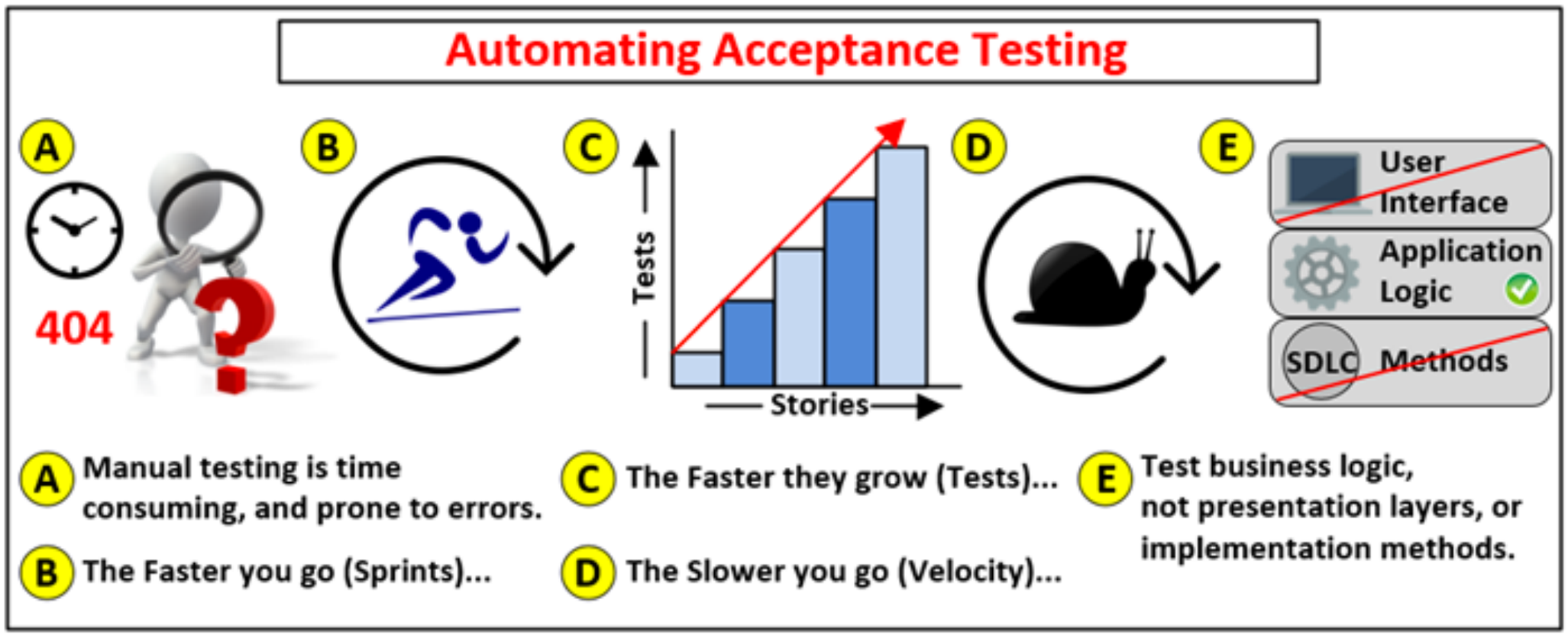
Agile Testing Quadrant



Agile Testing Quadrant



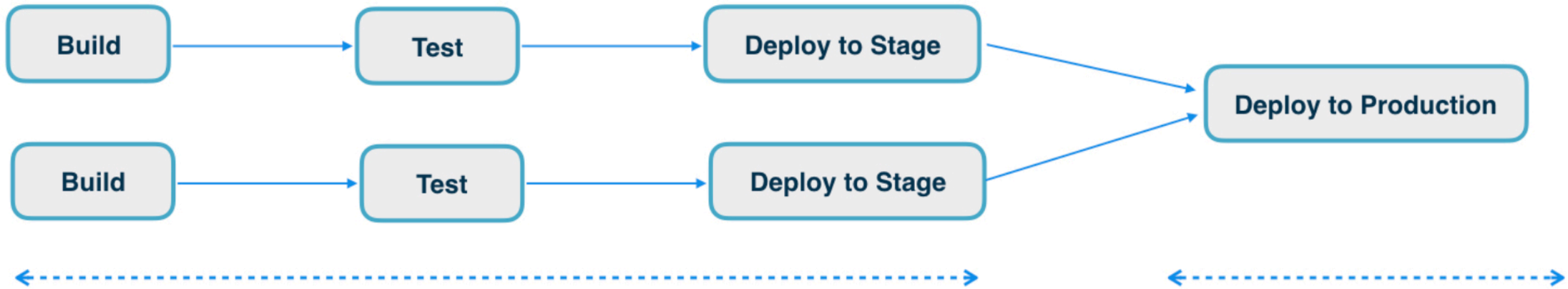
Test automation is essential

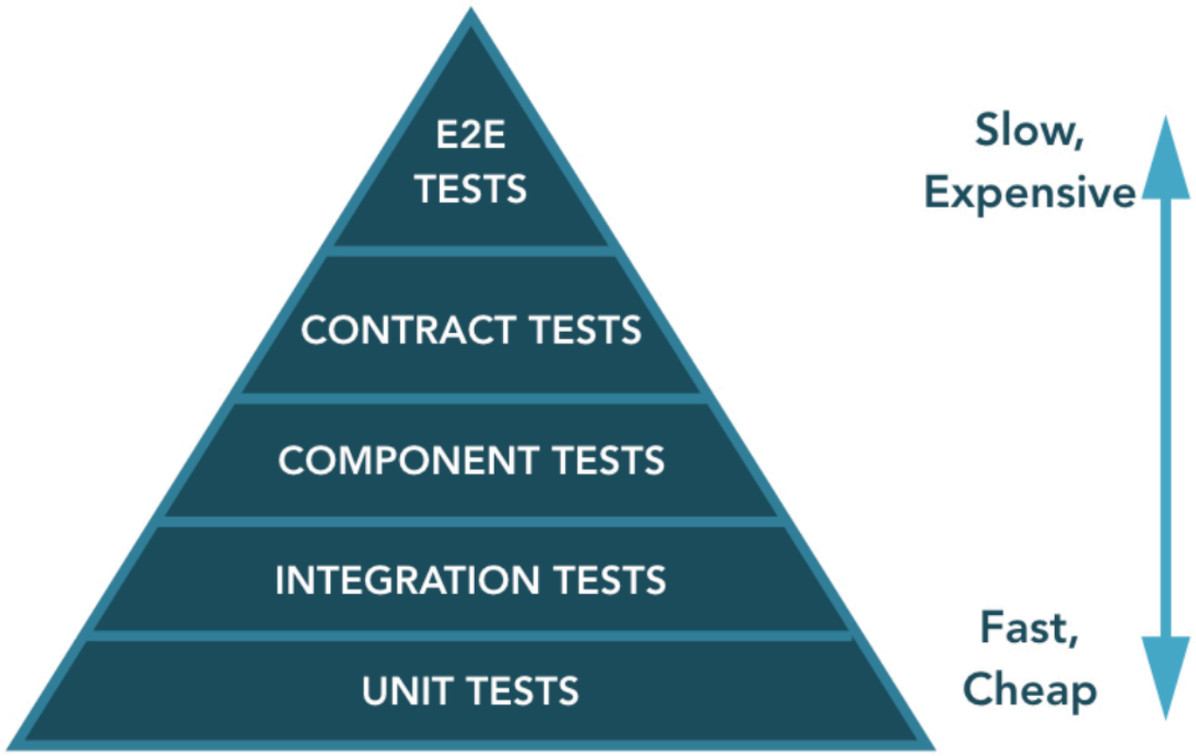


Test strategy

https://martinfowler.com/articles/microservice-testing/

Test strategy

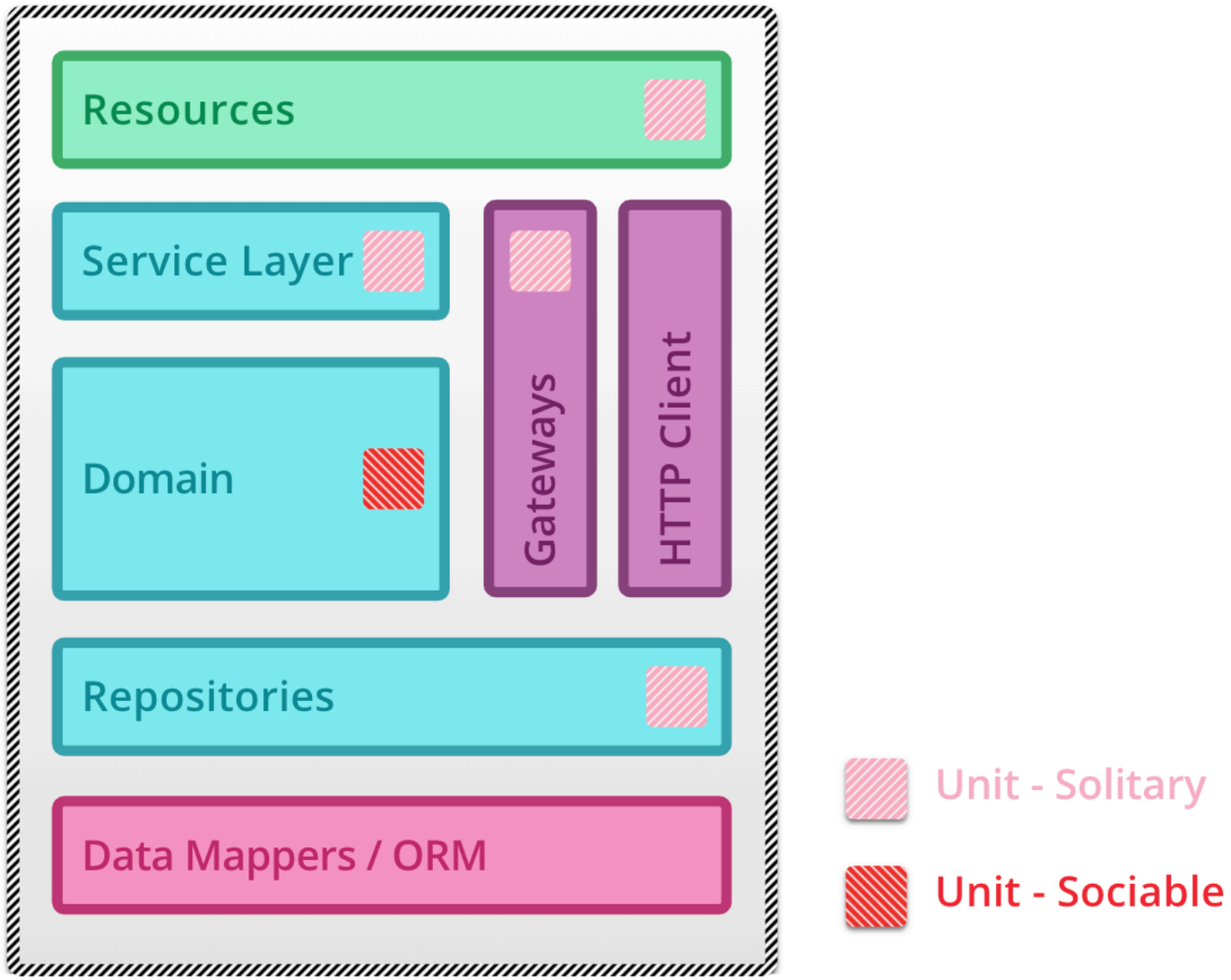




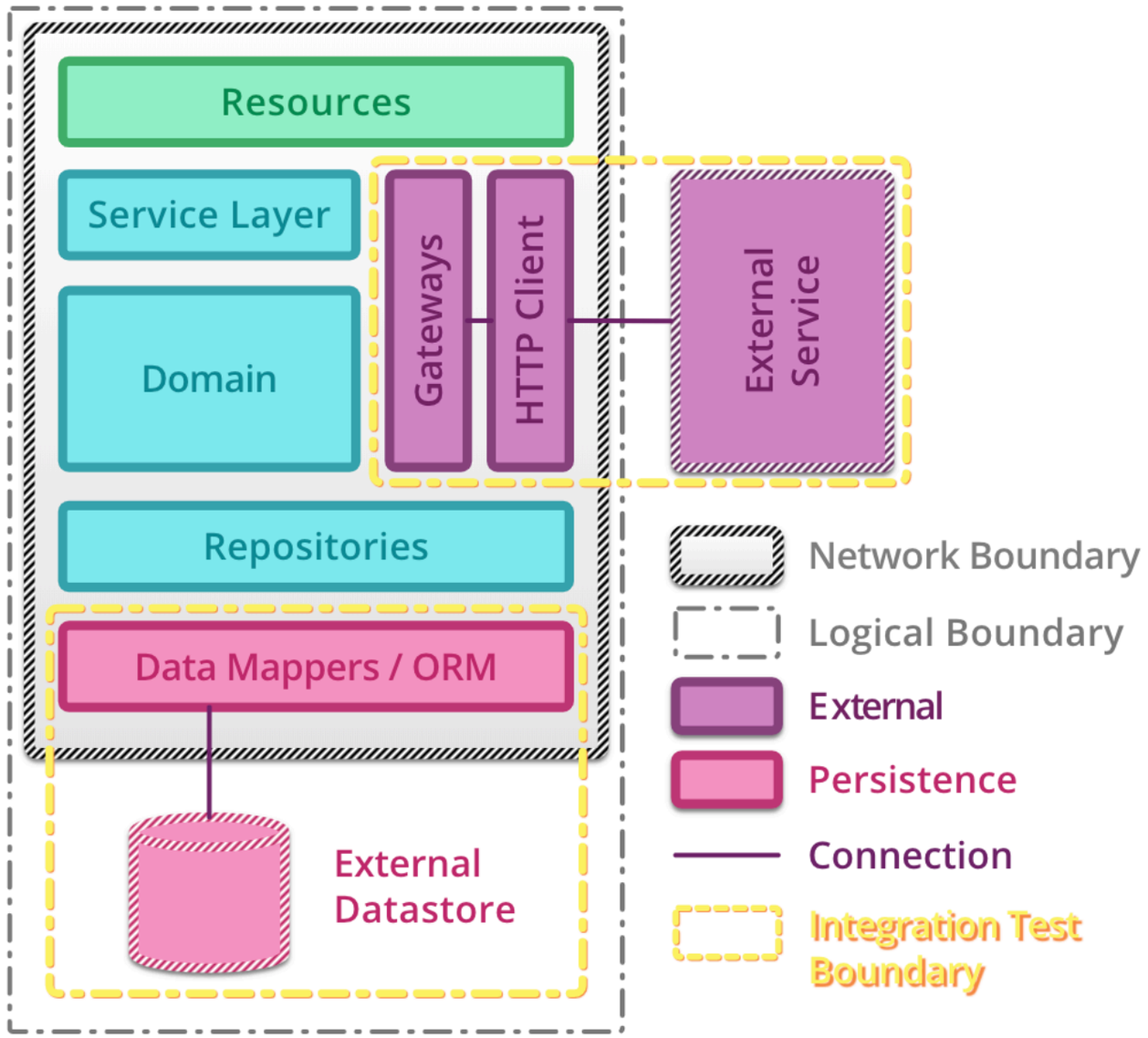
Monitoring
Distributed Tracing
Fault-injection Testing

Canary Deployments
Blue-Green Deployments
A/B Testing

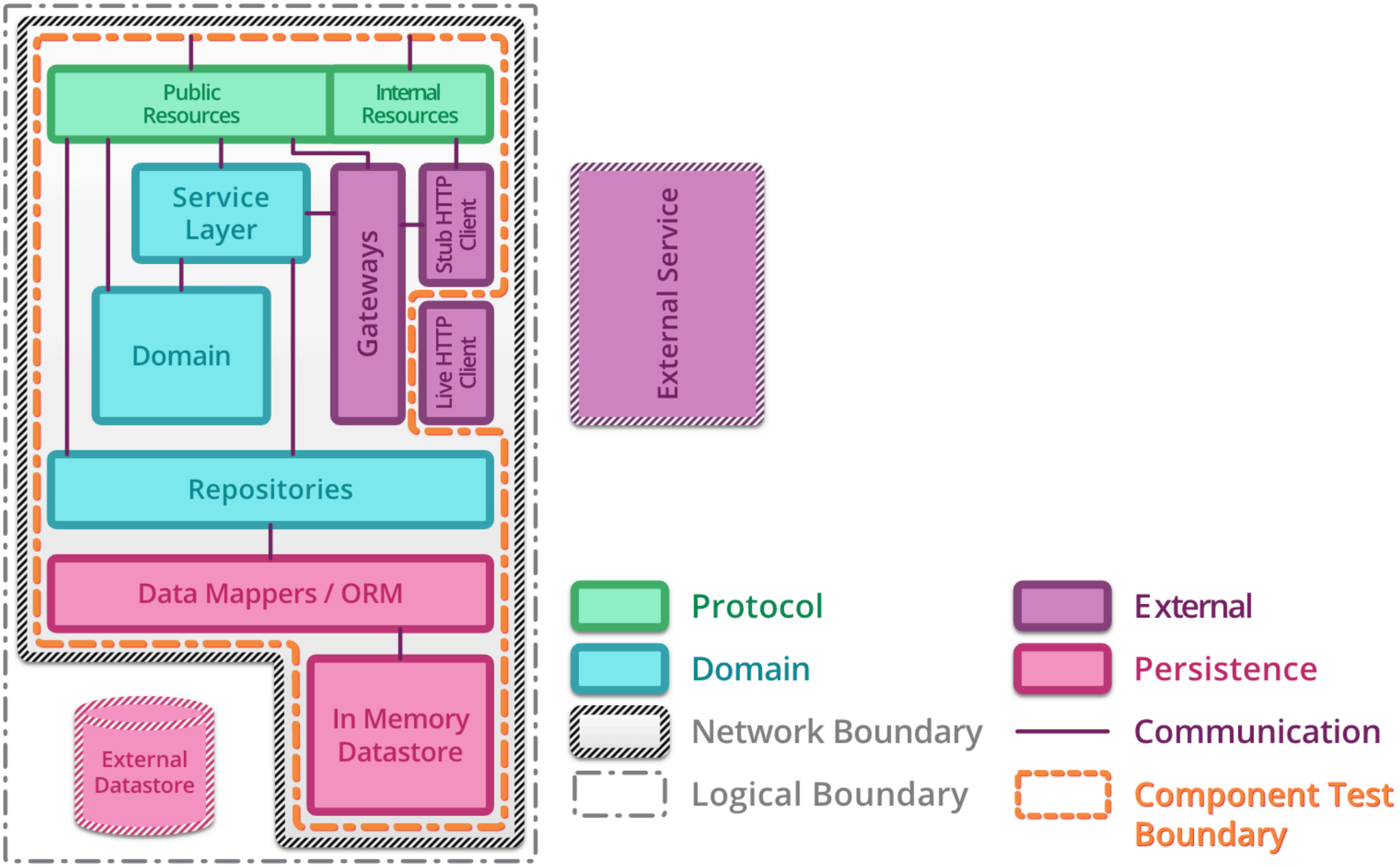
Unit testing



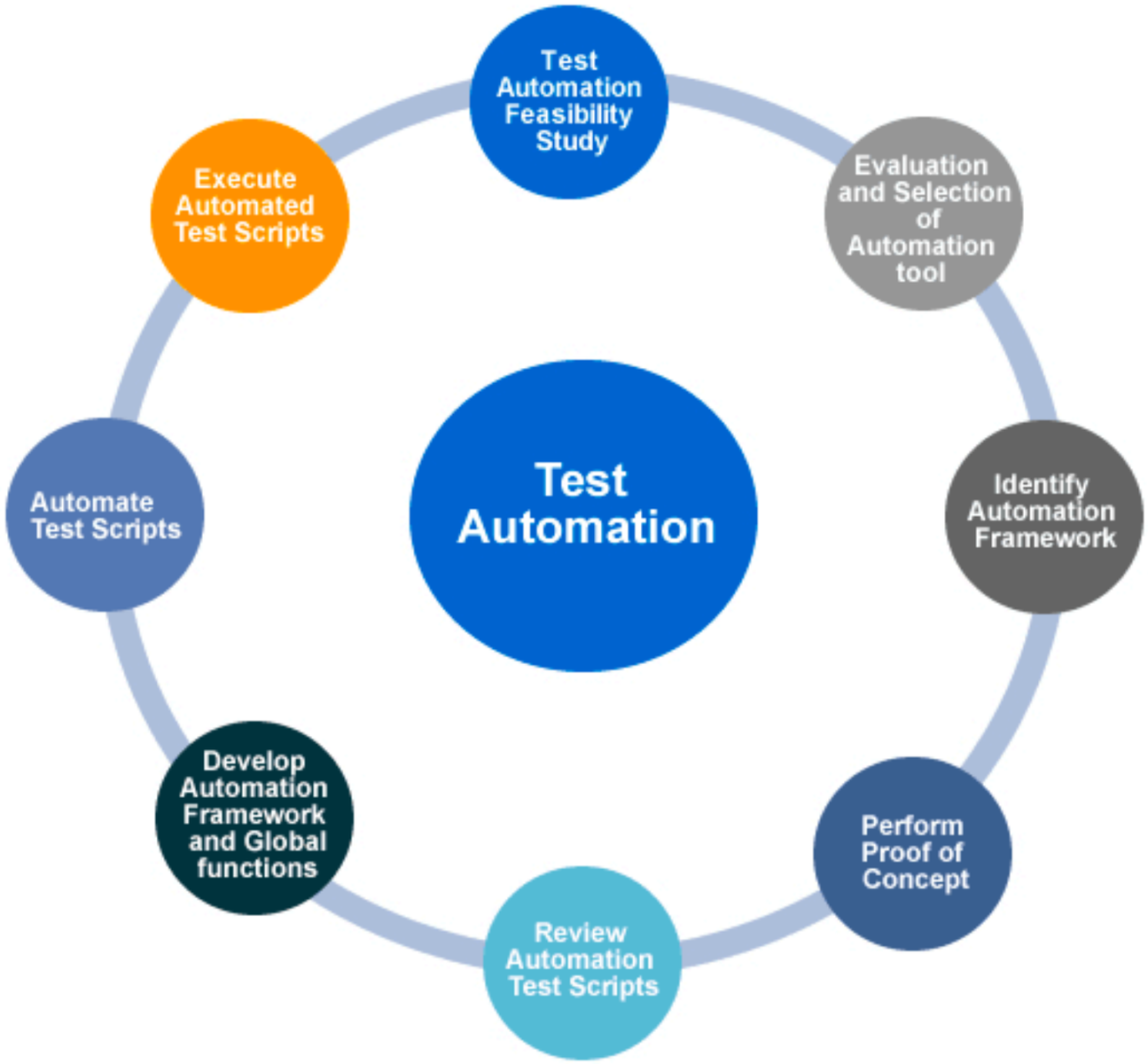
Integration testing



Component testing



Test Automation selection



Acceptance Tests

=

Business Criteria

+

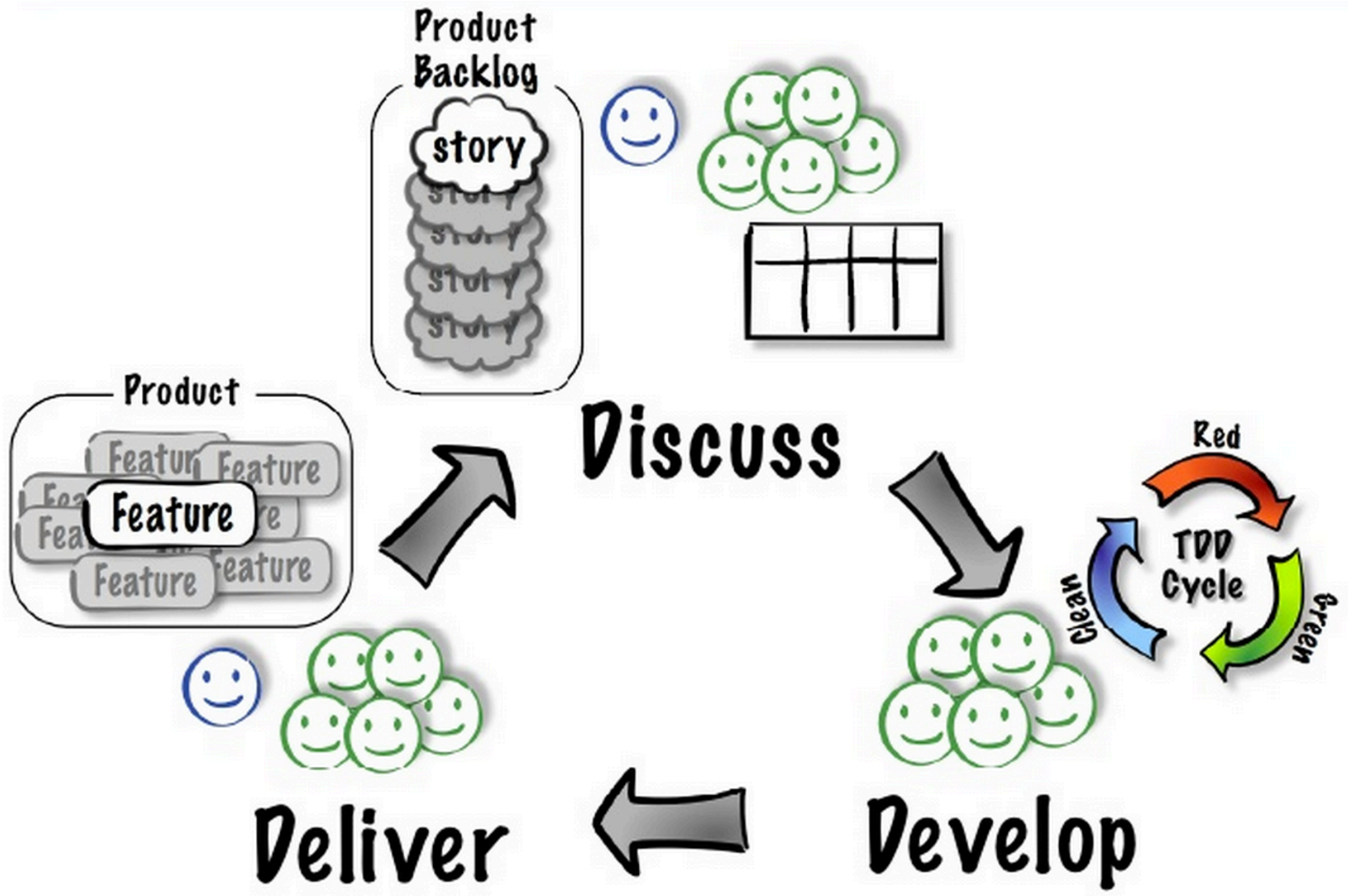
Examples (data)

Acceptance **T**est
Driven **D**evelopment

ATDD

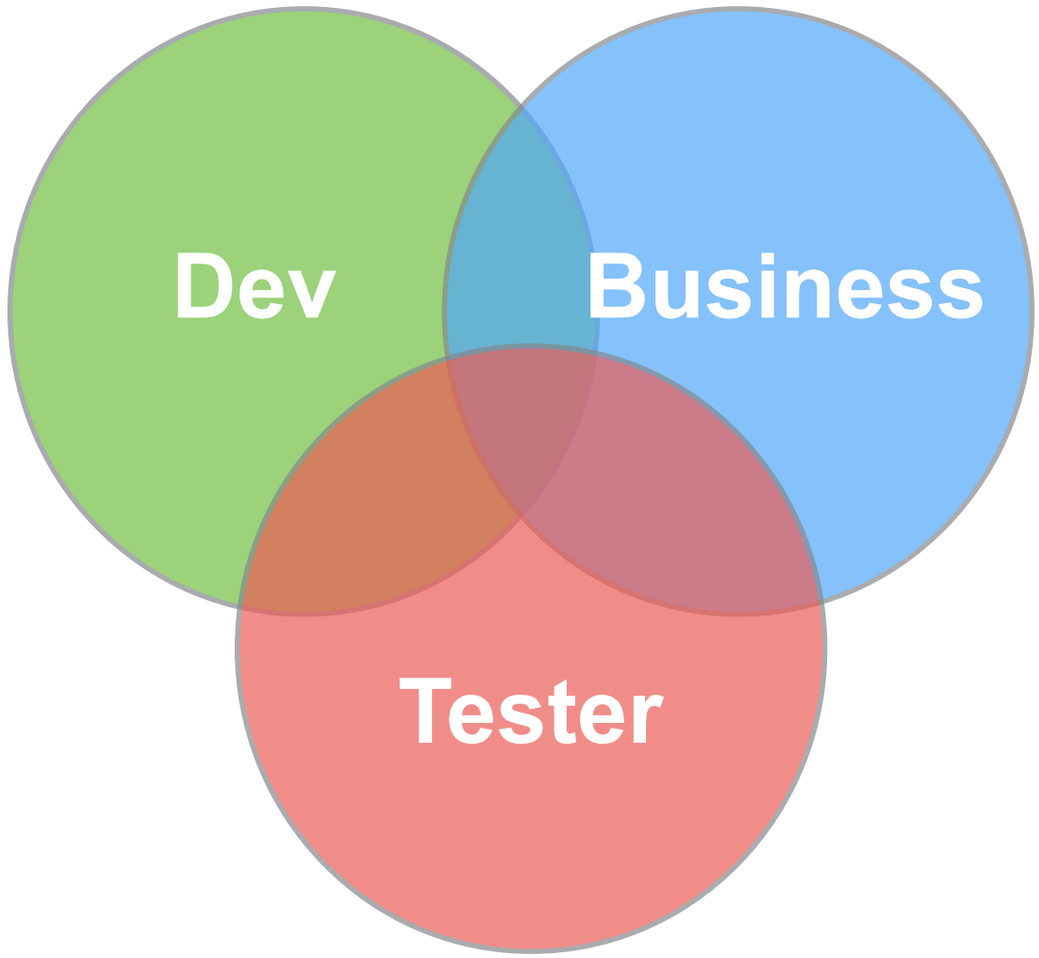
Common language
Common and share understanding
Executable requirements or examples
Living document

ATDD

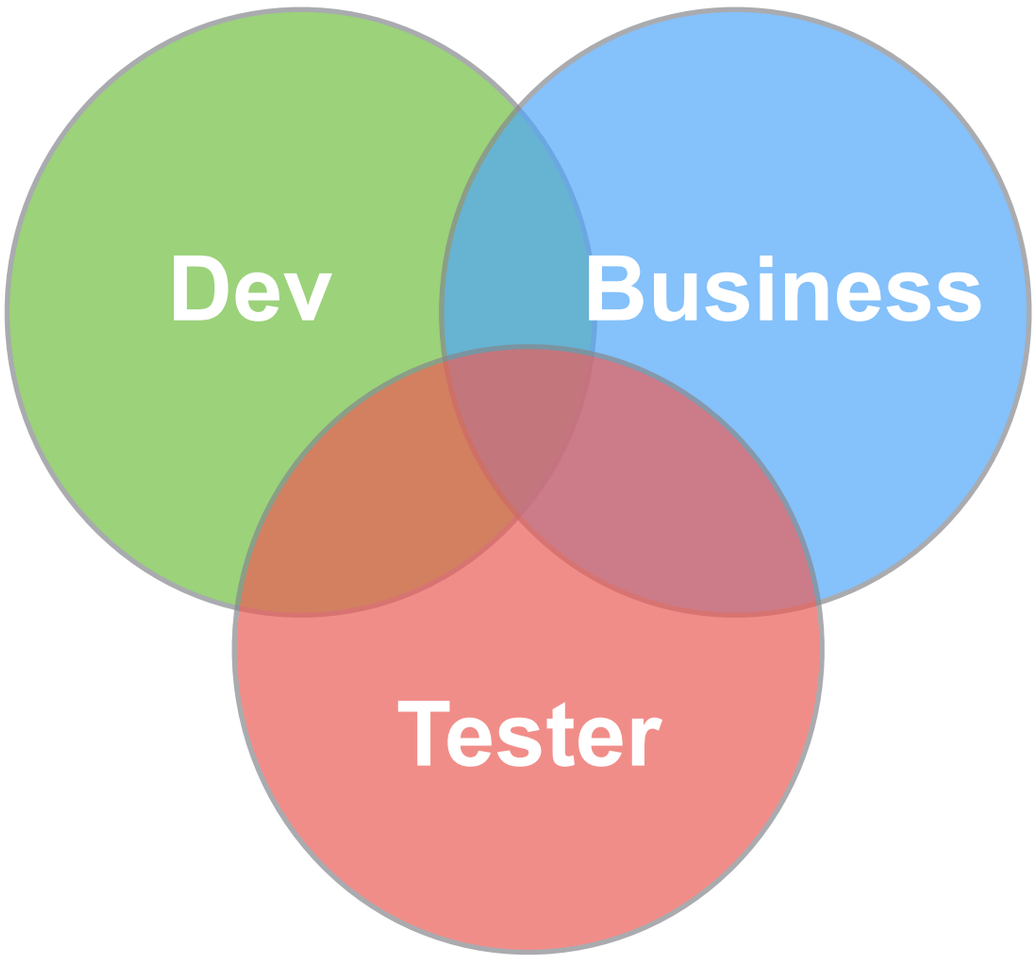


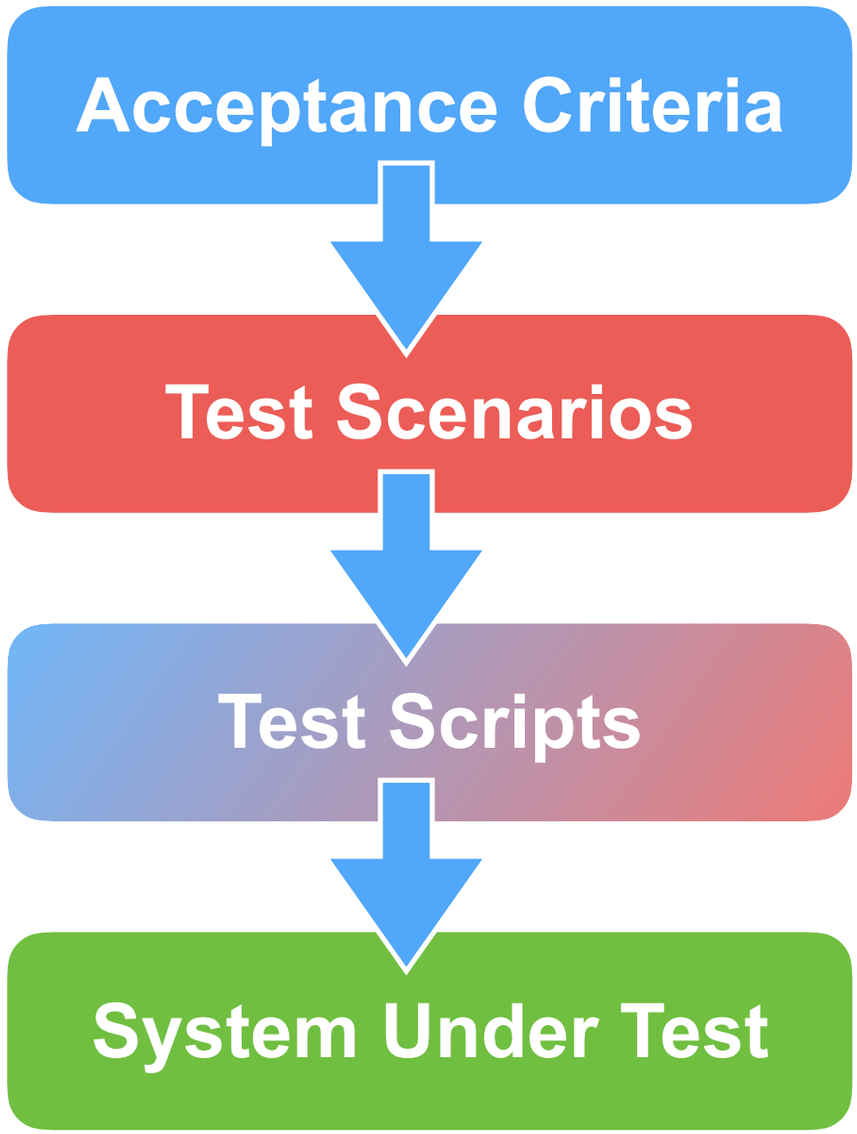
(Model developed with Pekka Klärck, Bas Vodde, and Craig Larman.)

Acceptance testing



Acceptance testing





Test design

Economy of Test Design

Easy to understand
Easy to maintain
Readable by the business
One purpose per test
Re-runnable

Poor test practices reduce the benefits

Respect your tests

Don’t ignore it if it fails

Fix the code or fix the test

100% of regression tests must pass all the time

Always refactor or improve

Test data !!

Avoid database/external system access

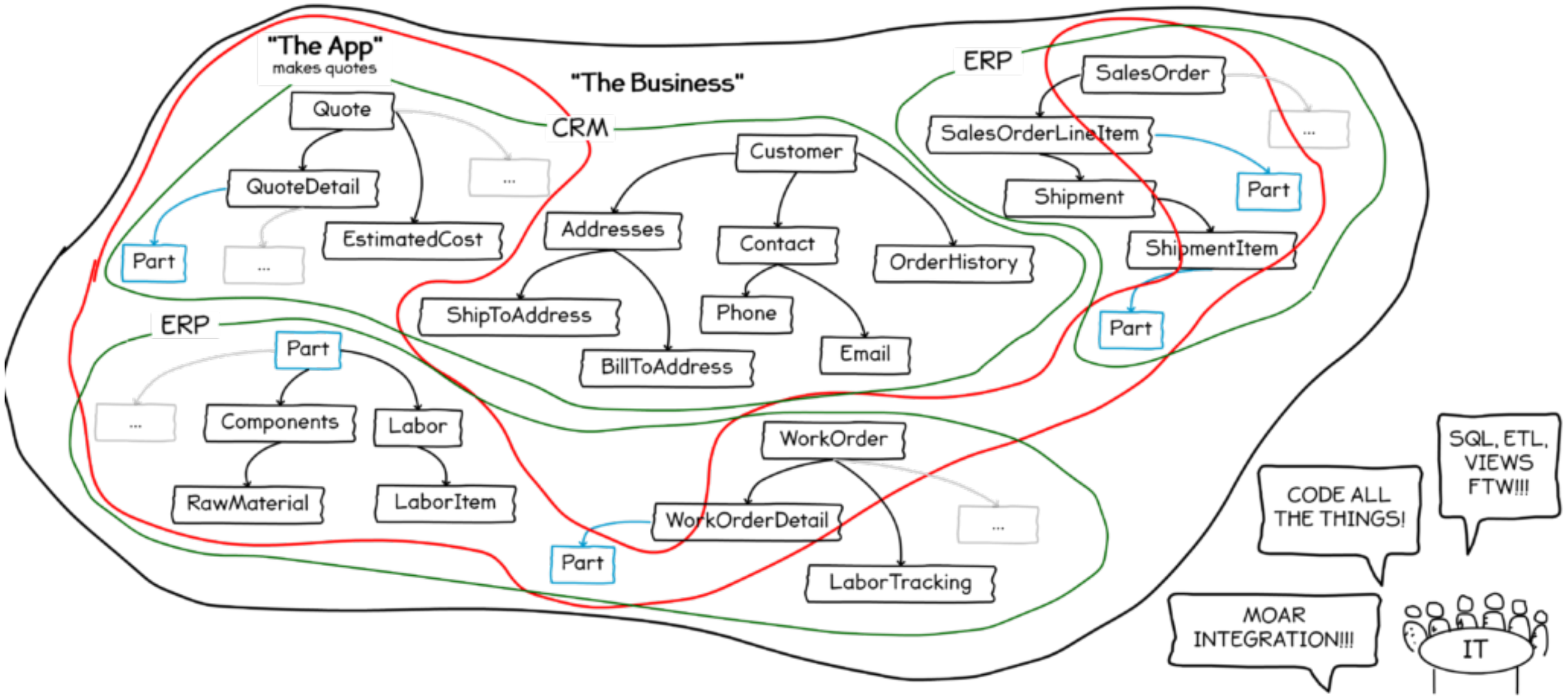
Setup/ tear down test data

Use production-like data

Need to control your data test

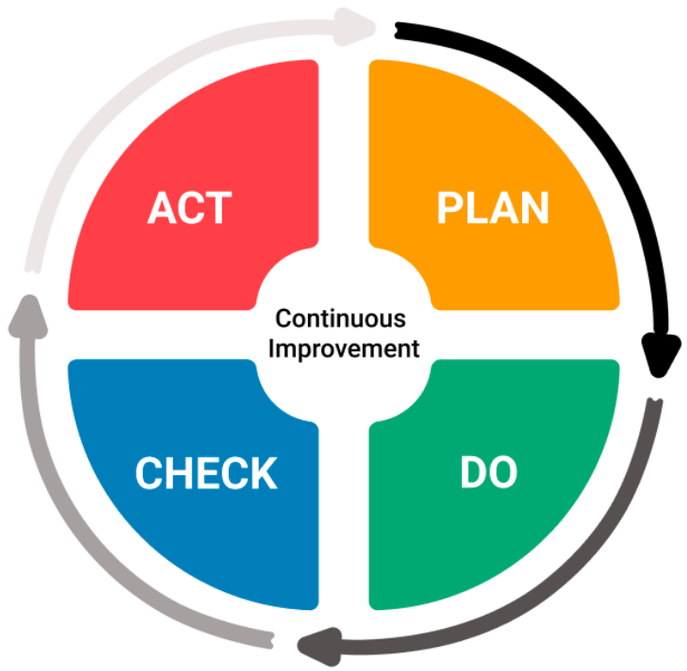
Systems impacts

Know your systems



Automation feedback !!

Easier over time ?
Time spent on maintenance ?
Test find regression bugs ?



Choosing your tools

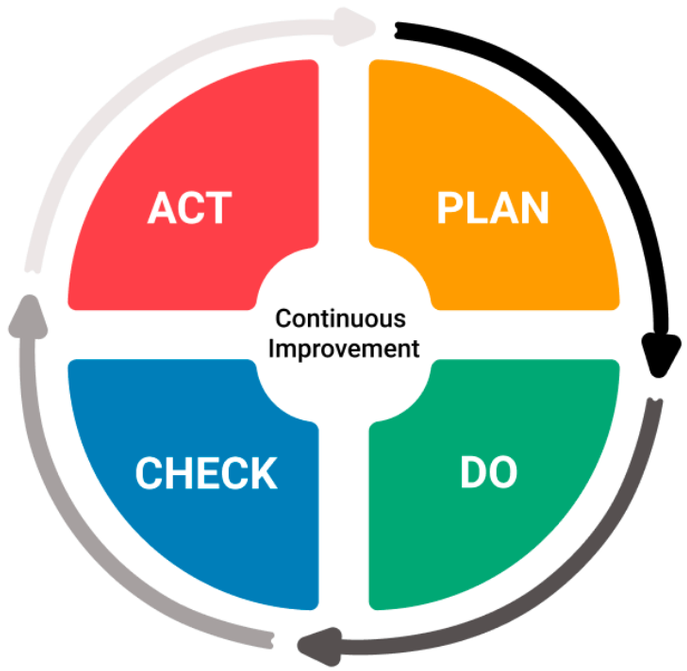
No one size fit all

Find time for evaluating

Right tool for the job

Determine your need and purpose

What ?
Constraints ?
Who ?



What fits your situations ?

Existing skills on the team
Language/technology off application under test
Collaboration needs
Workflow/process

Collaborative tools ?

Enable **tester** and **business** to define tests
Test code can be in programming language
Programmers can run tests as they code
Testers can ask **programmers** for help

Manage automated tests

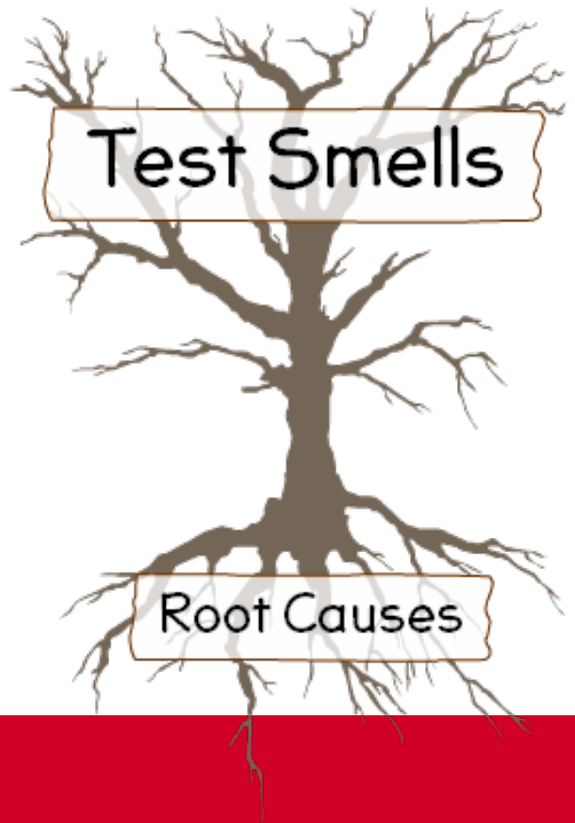
Use source control

Continuous integration to run tests on every change

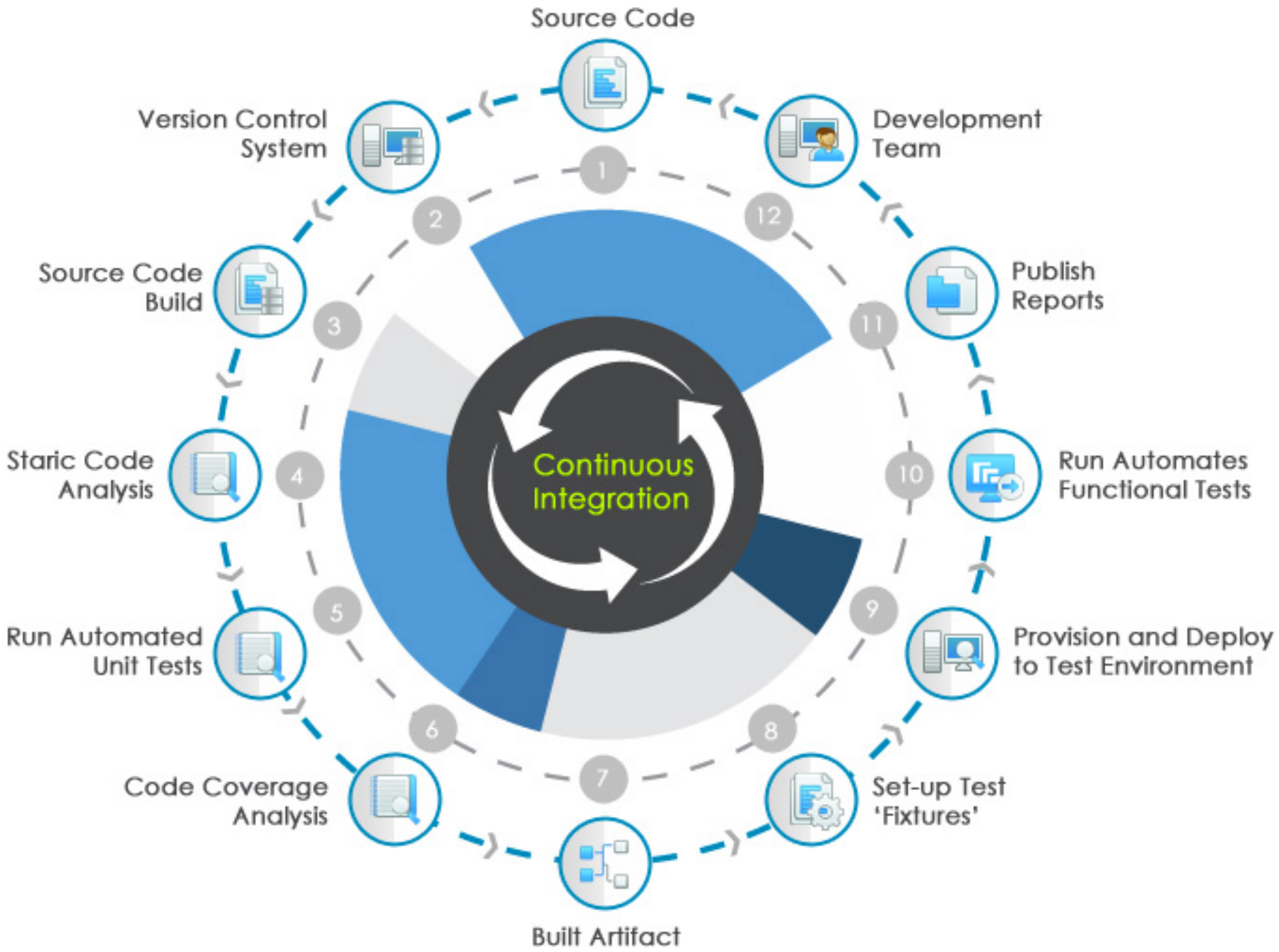
Keep all tests passed

Analyse failures tests

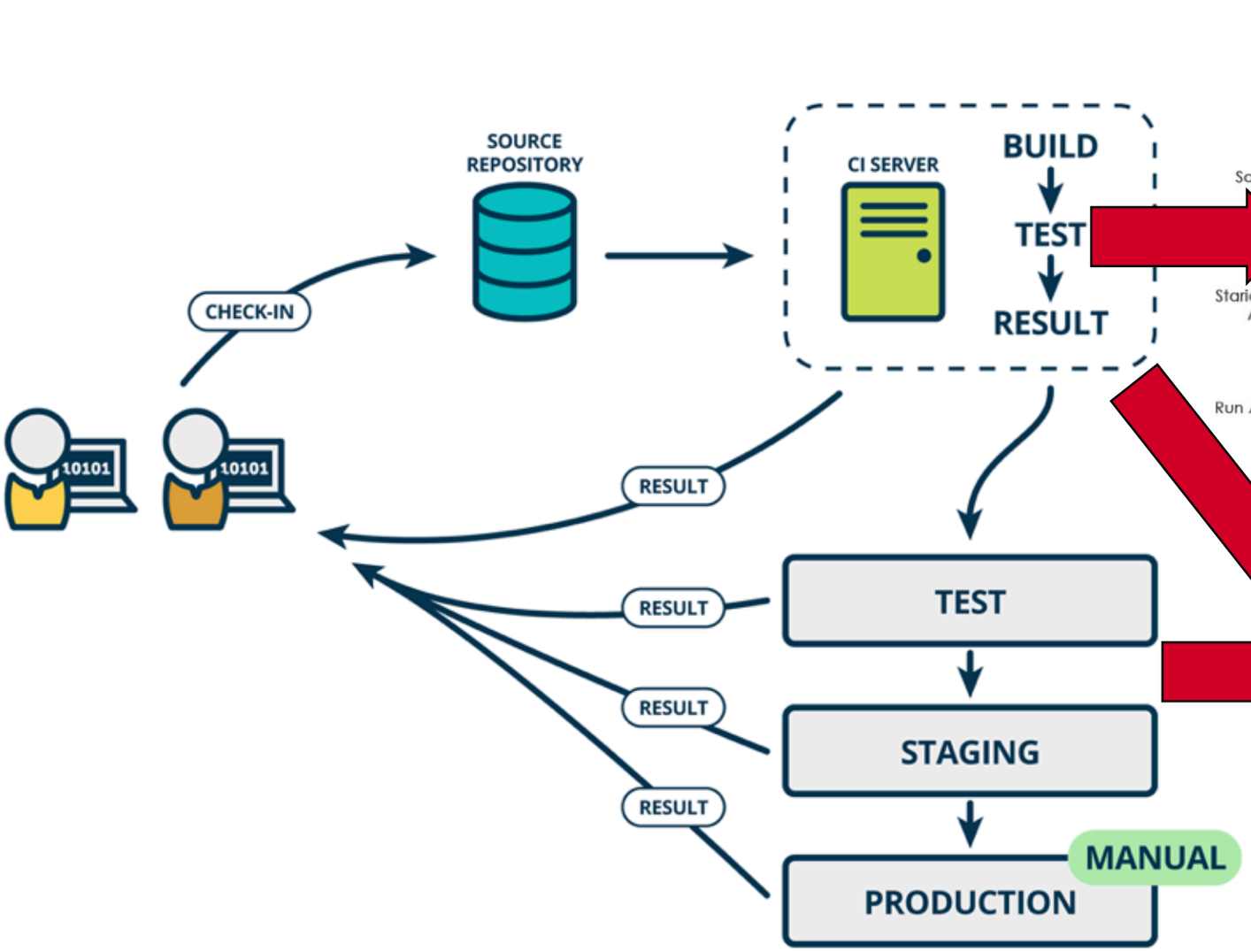
Create test standard

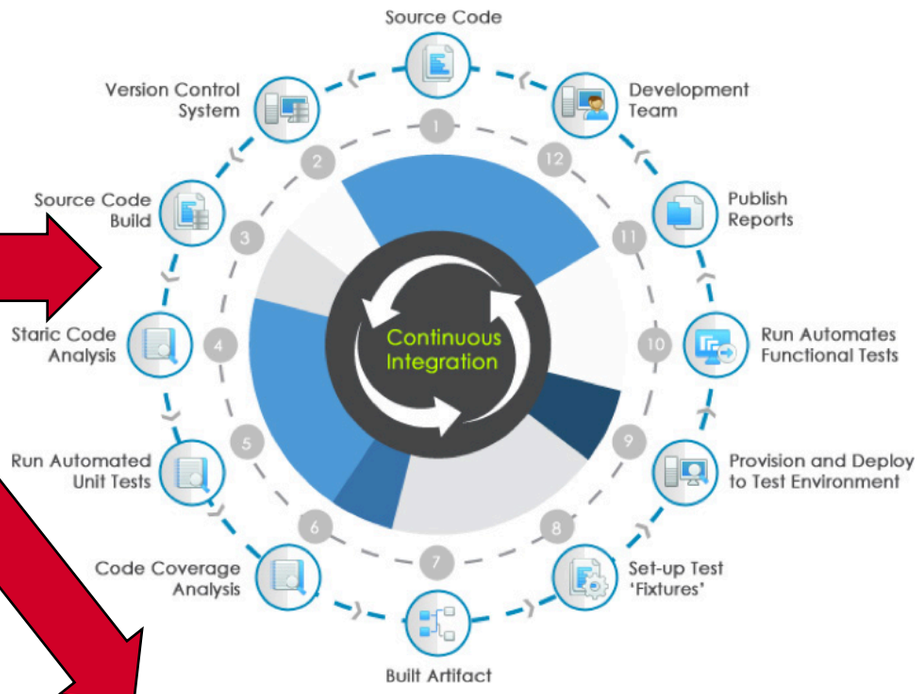


Continuous integration



Continuous integration





Build & Test Results Dashboard

Start with simple

Use feedback to improve

Welcome to new world !!

Discussion

Defect tracking (story of bugs)

How does your teams deal with defects ?

Defect tracking

Focus on prevention, not tracking
Defects are queues of rework (waste)
Zero (find, test-first, fix and forgot)

Alternative of Tracking tool !!

When to fix bugs ?

When to fix bugs ?

Fix now

Estimate, prioritise and fix later

Never fix

Manage and tracking defects ?

Not necessary if you have zero
Necessary for distributed teams
Necessary for legacy system
Look for trends