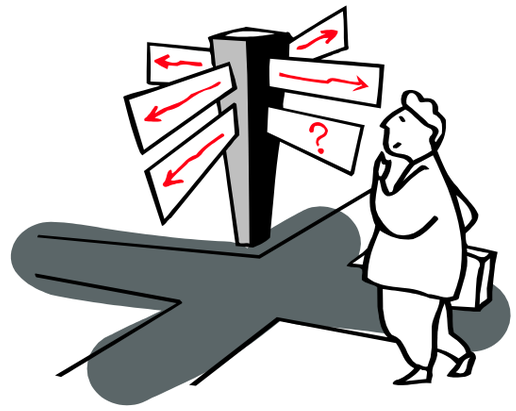
Software Testing

1. Introduction

1. Introduction



(Loosely based on “Chapter 1: Introduction” of Practical Test Design
+“Chapter 1: A Perspective on Testing” of Software Testing)

- Challenge
- What is Testing?
  + V-model
  + Agile Development / DevOps / …
  + Test Adequacy (vs. Test Inadequacy)
  + Testing = Risk Reduction
- Terminology
  + ISTQB
- Requirements
  + Failure Mode and Effects Analysis (FMEA)
  + Misuse cases
  + Safety stories

Challenge

Devise a test plan (i.e. a set of test cases) for a program that …

… reads three integer values from a card(*). The three integer values are interpreted as representing the lengths of the side of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral.

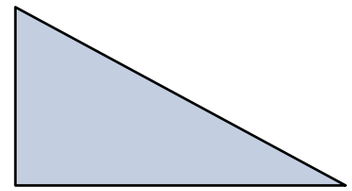
From “The Art of Software Testing” (Myers, 1978)

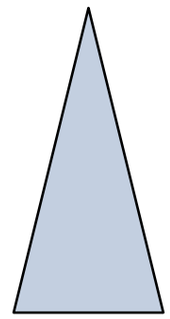
(*) Cards where the common input medium in 1978, you may interpret this as a file.

Challenge (help)

- A valid triangle must meet two conditions
  + No sides may have a length of zero
  + each side must be shorter than the sum of all sides divided by 2

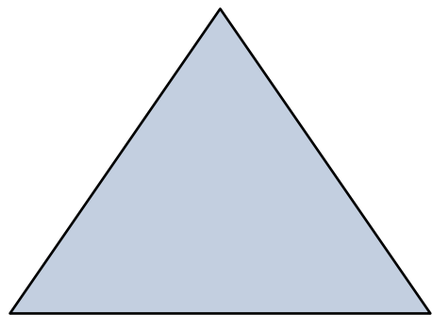
- A triangle is
  + scalene: no sides are equal in length





  + isosceles: there exist two sides which are equal in length

  + equilateral: all sides are equal in length

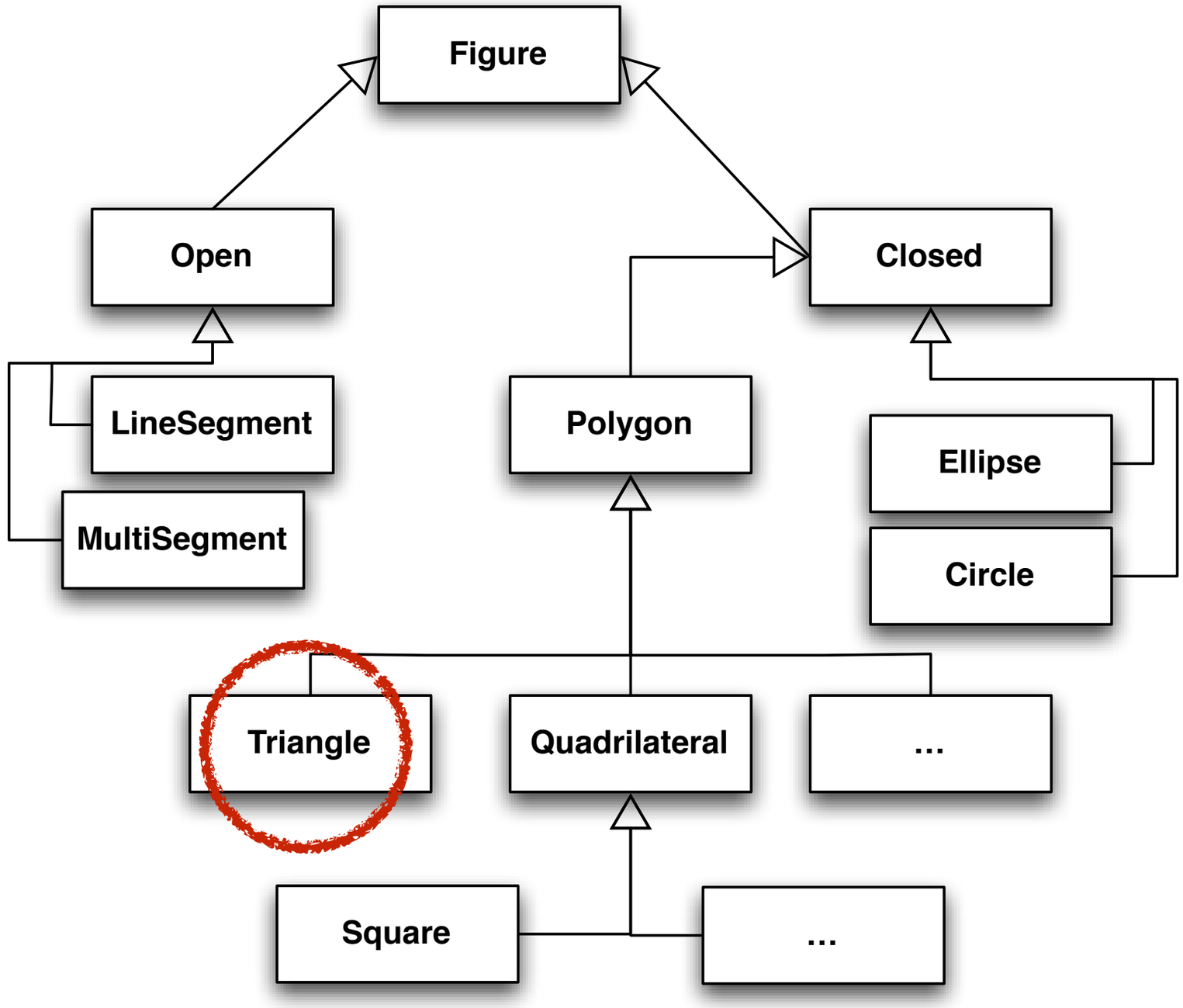


Challenge (solution)

- 3 one valid for each scalene, isosceles, or equilateral
- 3 permutations for equal sides (all isosceles)
- 1 one side a zero length
- 1 one side negative length
- 3 permutations for equal sides (all invalid)
- 6 permutations (one side smaller than sum of all sides divided by 2)
- 1 all sides zero
- 3 non-integer inputs
- 3 missing inputs
- 6 permutations (one side equals the sum of the other two)
- 3 three, two and one sides at maximum value (MAXINT)

  - 33 test cases are possible!
  - Highly experienced programmers score on the average 7.8/14

Challenge revisited (class diagram)



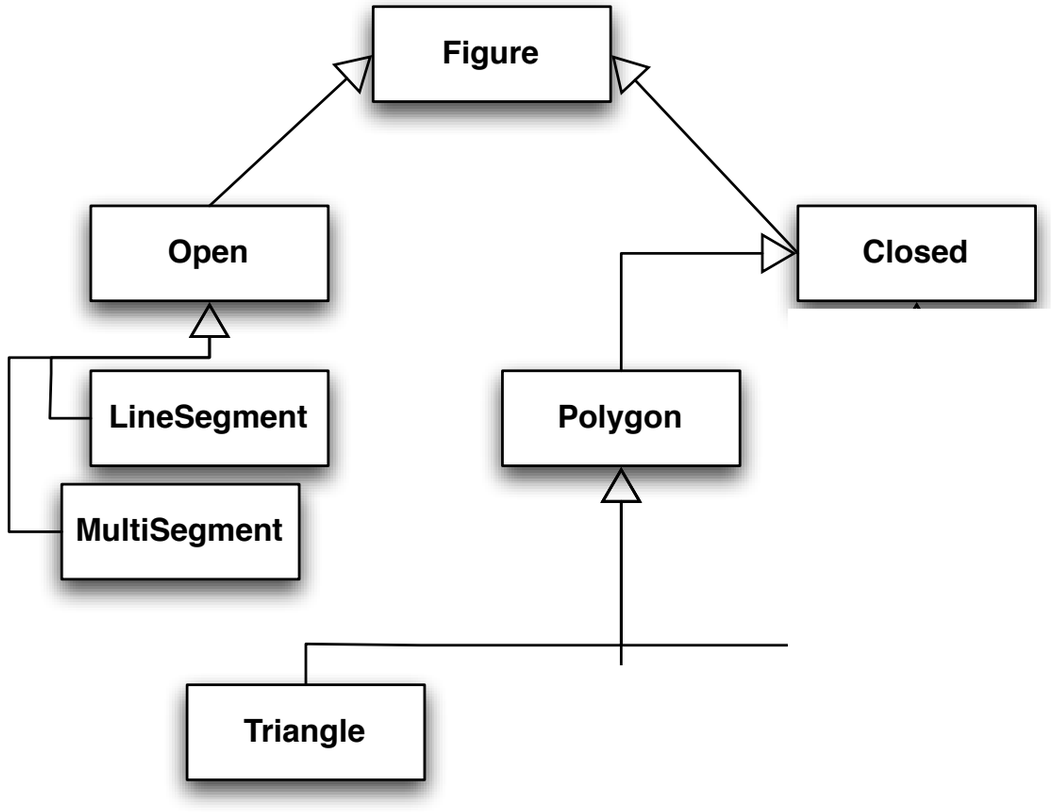
How to test?

- is_scalene()
- is_isosceles()
- is_equilateral()

Challenge (object-oriented - solution)

Original Myers Tests

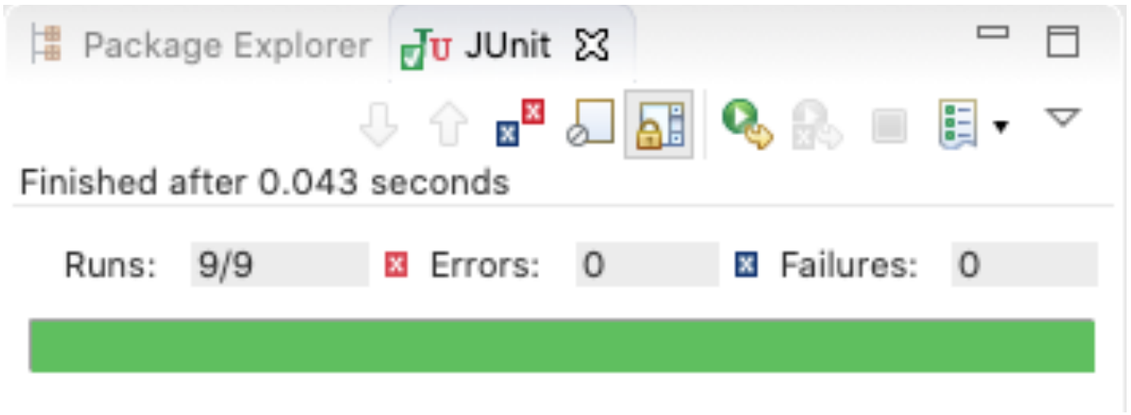
- 33 test cases
  + 6 not possible (non integer input & missing input)
- 62 + 27 = 89 test cases!

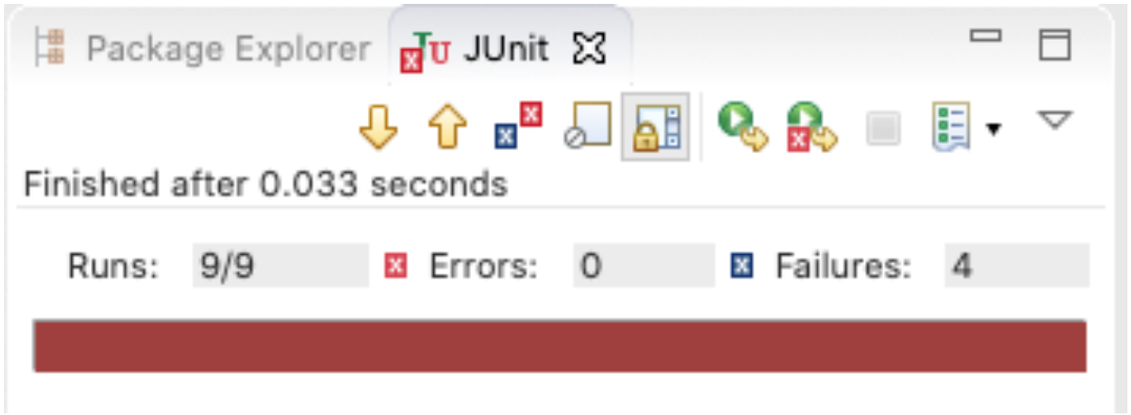


Inheritance & Polymorphism

- All methods defined in Figure inherited or overridden by Triangle provide a response that is consistent with the original definition
- above for *Closed*
- above for *Polygon*
  + 3 x other tests with a substituted triangle

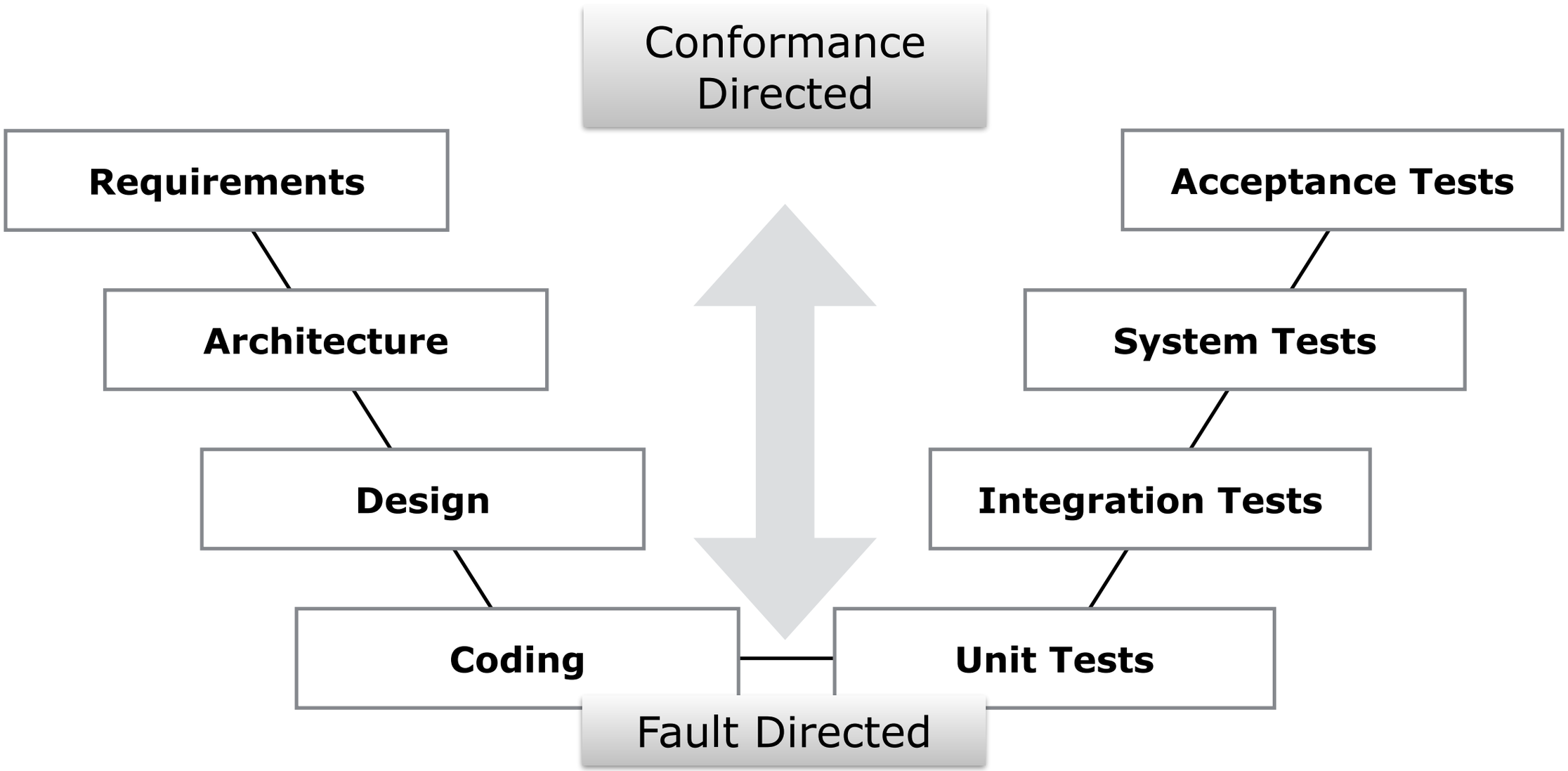
What is Testing?



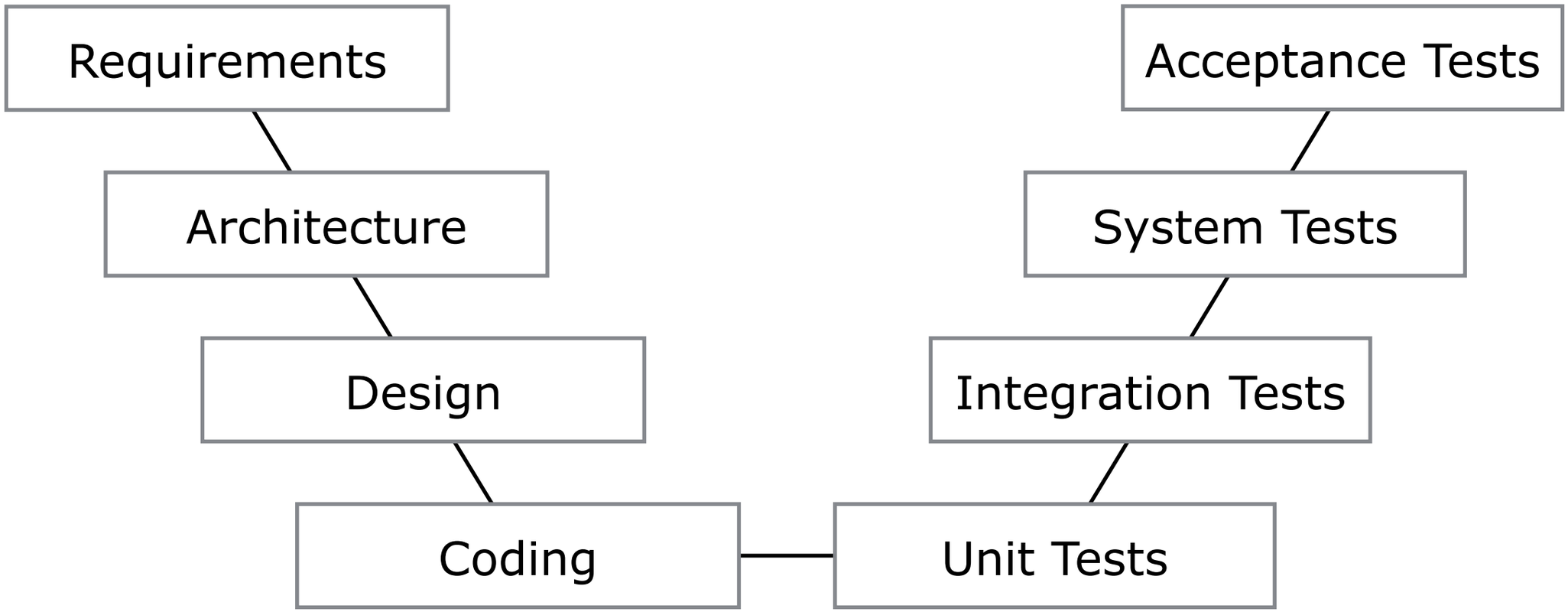


Software Testing is the process of executing a program or system with the intent of finding errors.
(Myers, Glenford J., The art of software testing. Wiley, 1979)

Test Strategy = The V-model

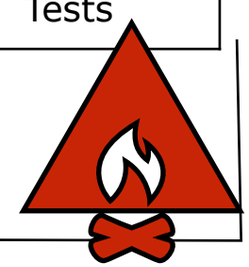


V-Model

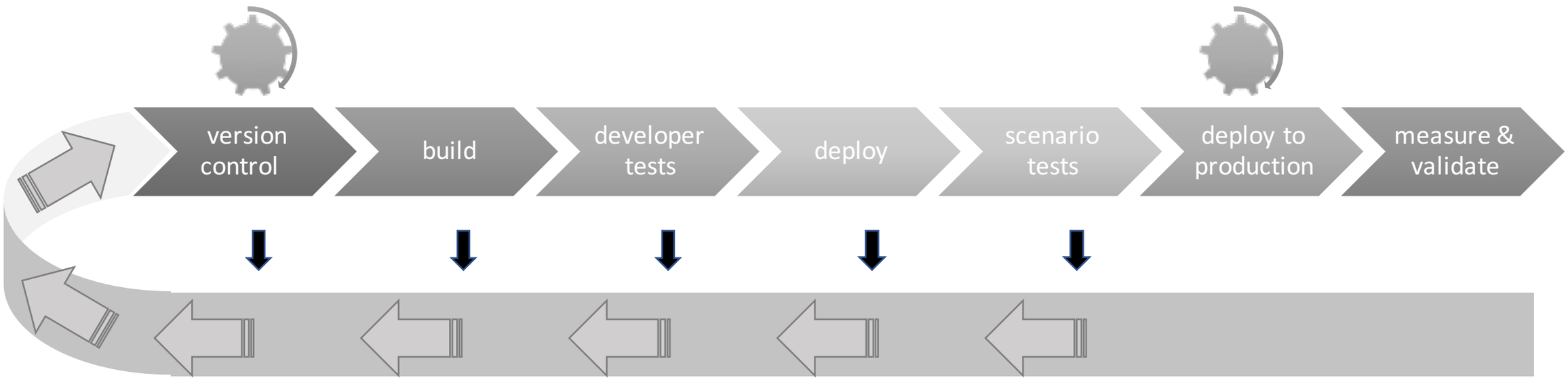


Requirements	Architecture	Design	Coding	Testing			
Test Design				Test Execution			
Acceptance Test Cases	System Test Cases	Integration Test Cases	Unit Test Cases	Unit Tests	Integration Tests	System Tests	Acceptance Tests

Integration hell?

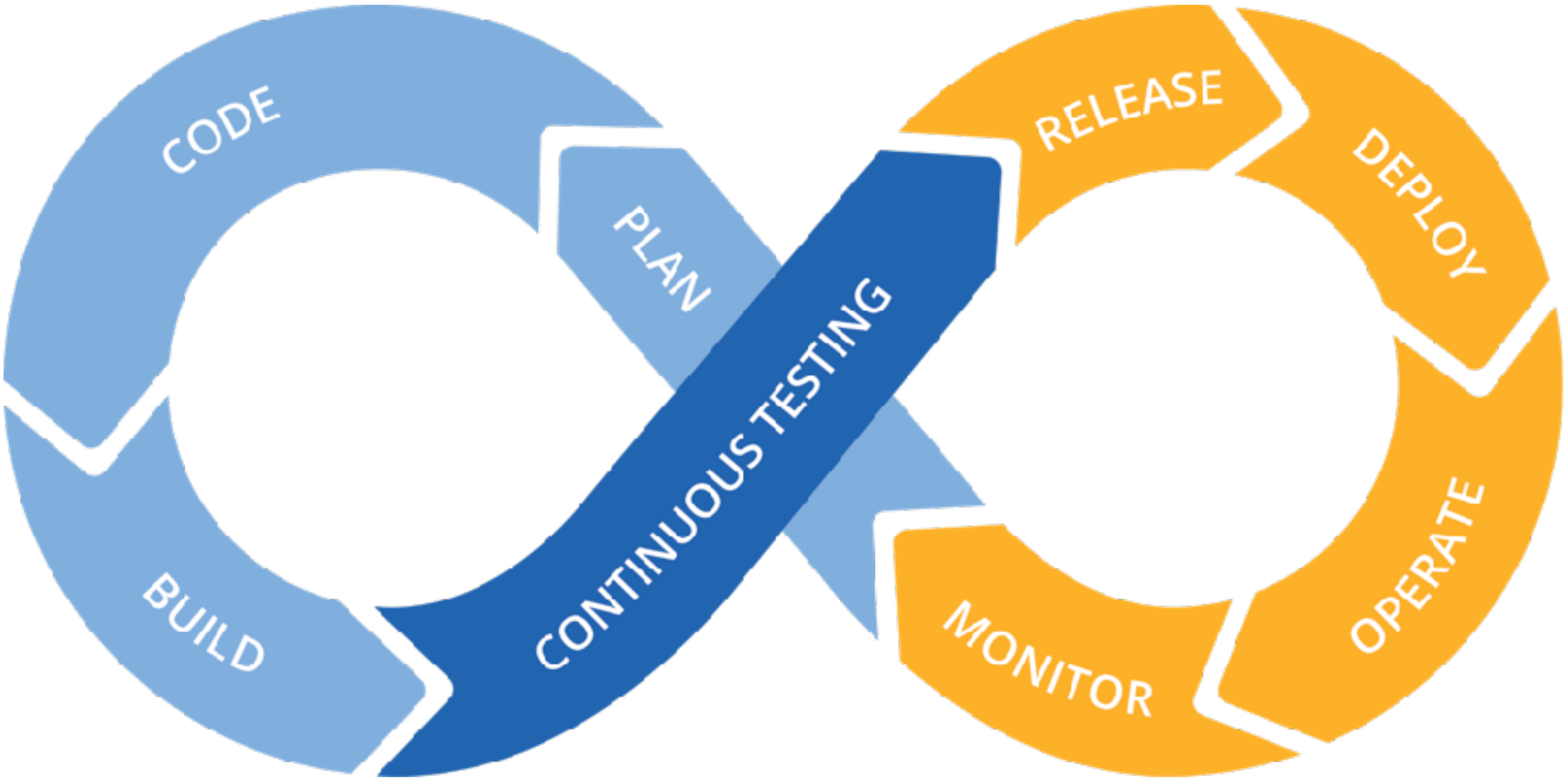


Continuous Integration Pipeline

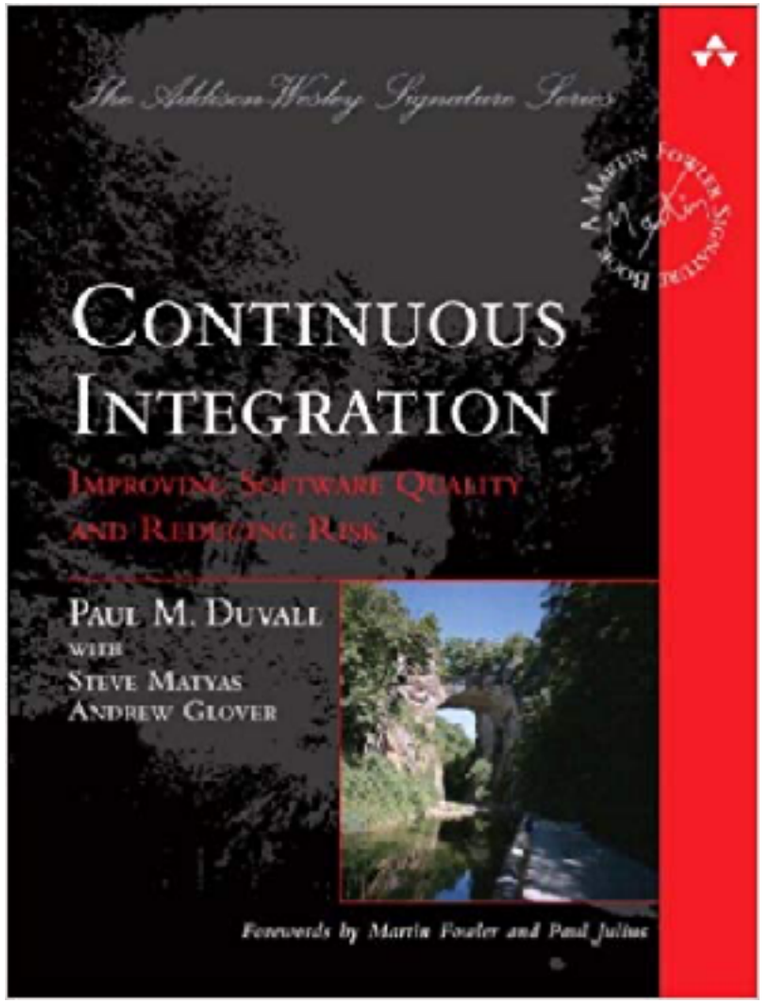


<<Breaking the Build>>

DevOps

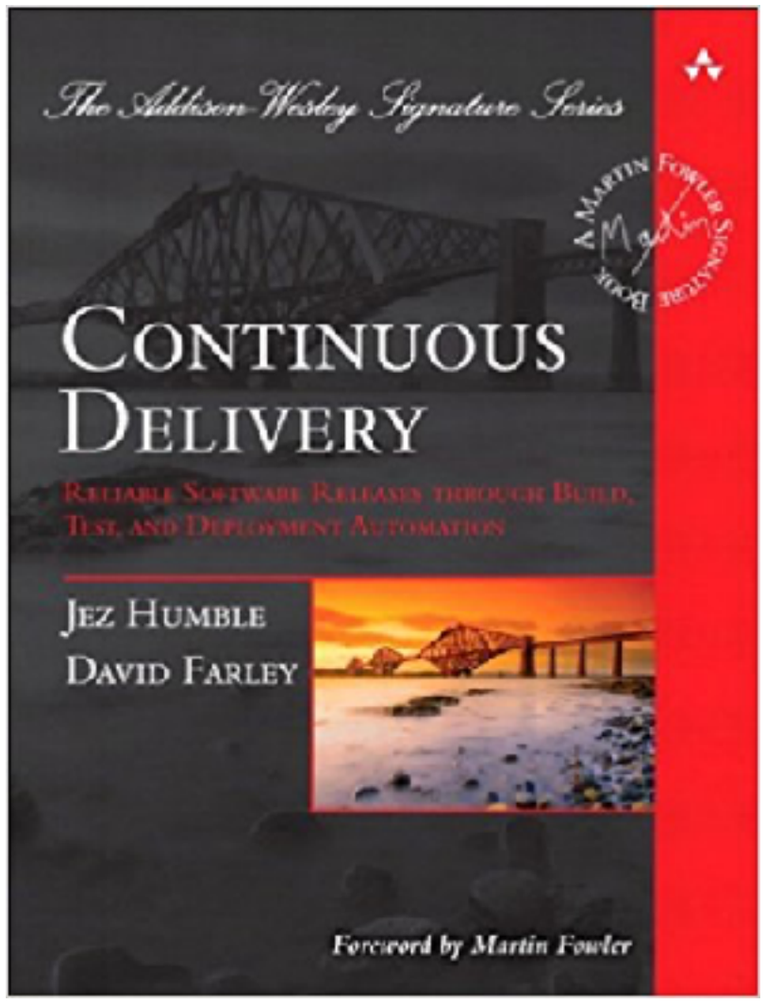

CODE
PLAN
CONTINUOUS TESTING
RELEASE
DEPLOY
BUILD
MONITOR
OPERATE

The DevOps Spectrum



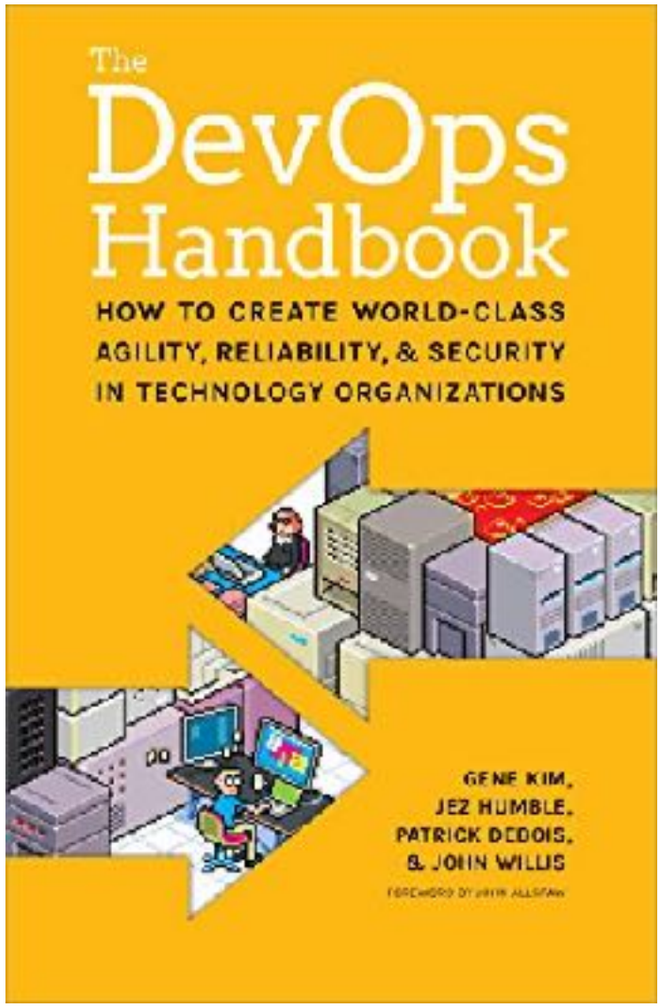
Continuous Integration

Tesla
"over-the-air" updates
± once every month



Continuous Delivery
Continuous Deployment

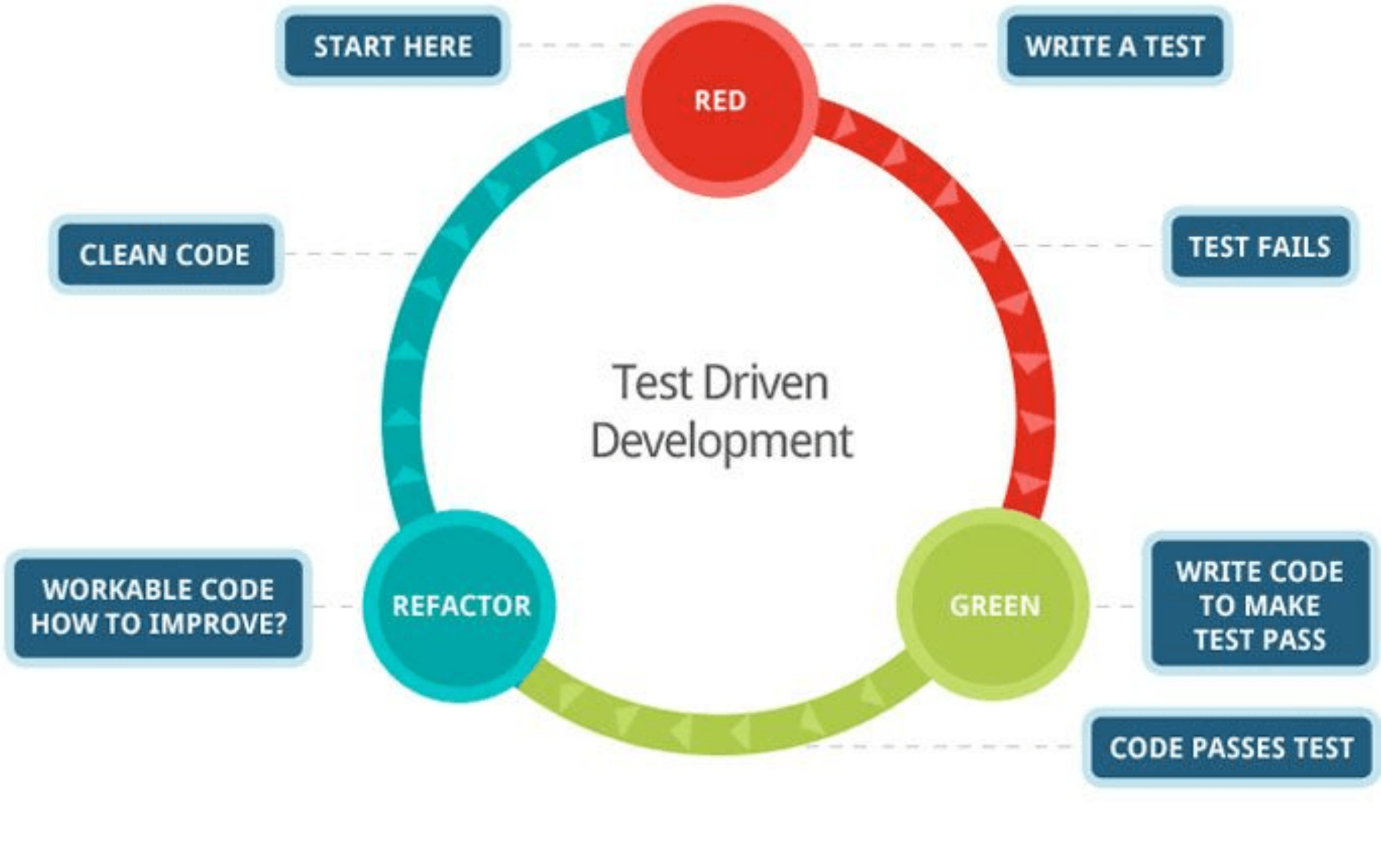
Amazon deploys to production
± every 11,6 seconds



DevOps

September 2015
Amazon Web Services suffered major disruption.
NetFlix recovers quickly!
(Chaos Monkey)

Test Driven Development



4 Quadrants

Automated & Manual

Business Facing

Manual

Supporting Team	Functional Tests Examples Story Tests Prototypes Simulation	Exploratory Testing Scenarios Usability Testing Acceptance Testing Alpha / Beta
	Unit Tests Integration Tests	Performance Testing Load Testing Security Testing "ility" Testing

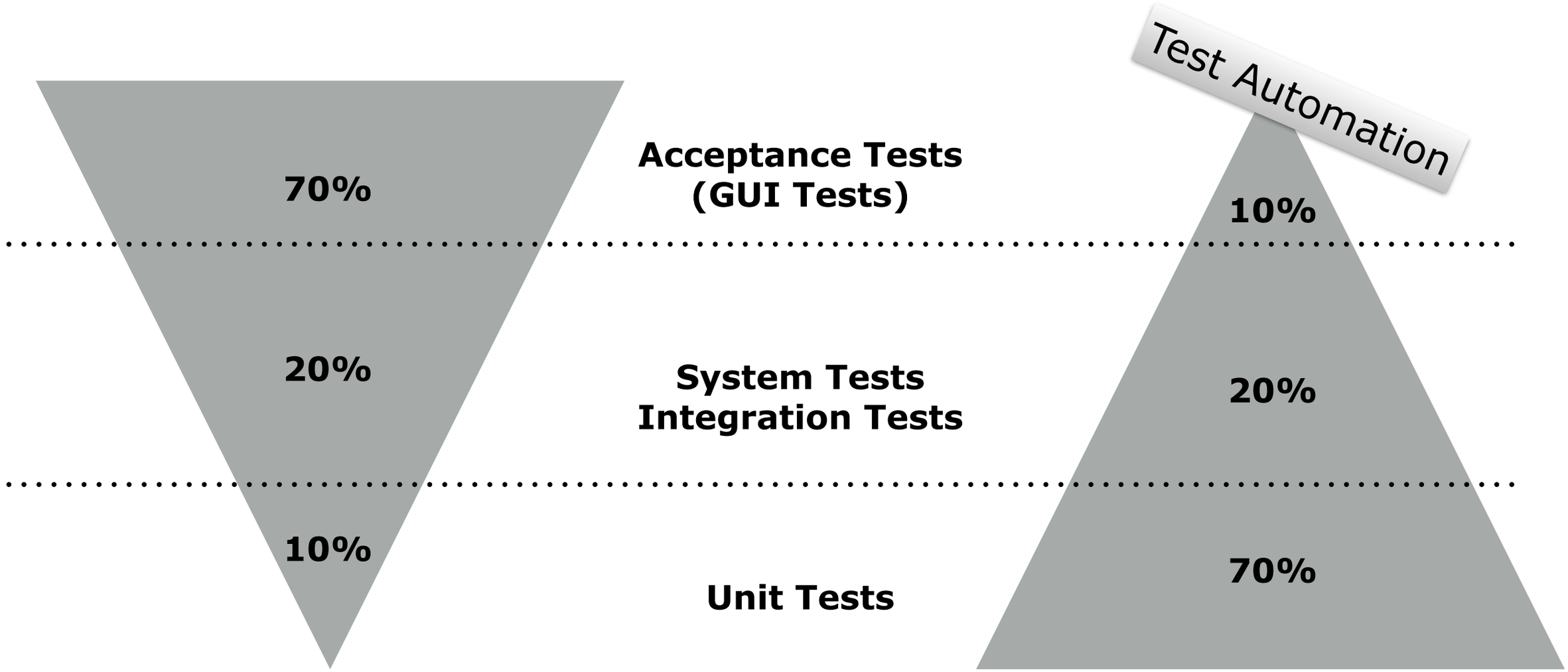
Critique Product

Automated

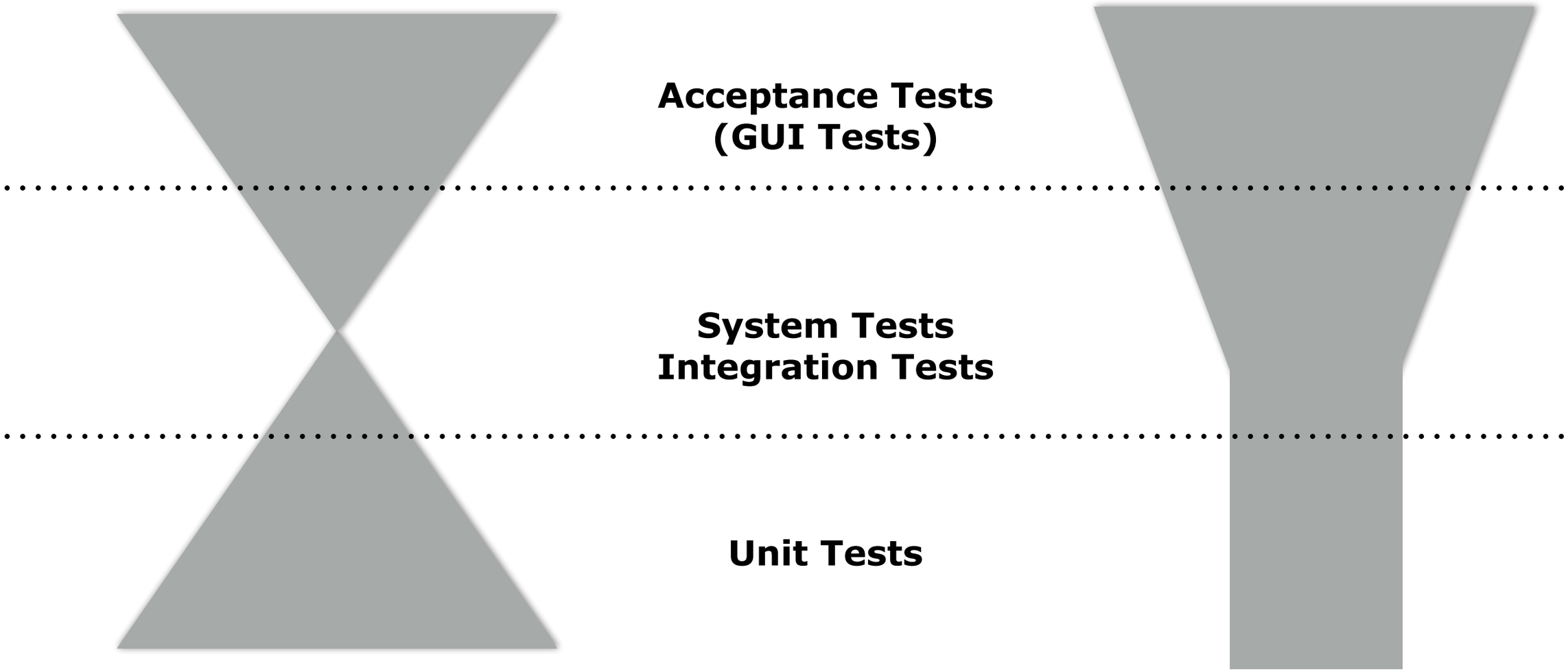
Tools

Technology Facing

Flipping the V



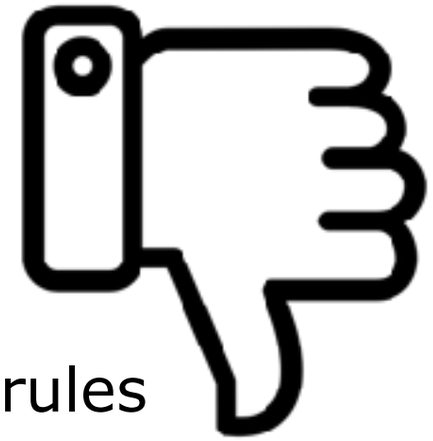
Flipping the V in Practice



Test Adequacy vs. Test Inadequacy

The “Test Adequacy Utopia”

- If a system passes an ADEQUATE suite of test cases, …
  then it must be correct
  + impossible: provable undecidable



Weaker proxies for adequacy

- Design rules to highlight INADEQUACY of test suites
- If a given suite of test cases does not satisfy the design rules
  … reconsider carefully
  + typically expressed via some form of *coverage*
- compare: “due diligence”

Testing is risk assessment!
(Can we release?)

Risk Projection (2 dimensions)

		impact		
		Low	Medium	High
likelihood	High	low	medium	high
	Medium	low	medium	medium
	Low	low	low	low

Risk = impact * likelihood

		impact				
		insignificant	minor	moderate	major	catastrophic
likelihood	almost certain	moderate	high	high	critical	critical
	likely	moderate	moderate	high	high	critical
	possible	low	moderate	high	high	critical
	unlikely	low	moderate	moderate	high	high
	rare	low	low	moderate	moderate	high

Risk Projection (3 dimensions)

Sometimes a 3rd item is added to the equation

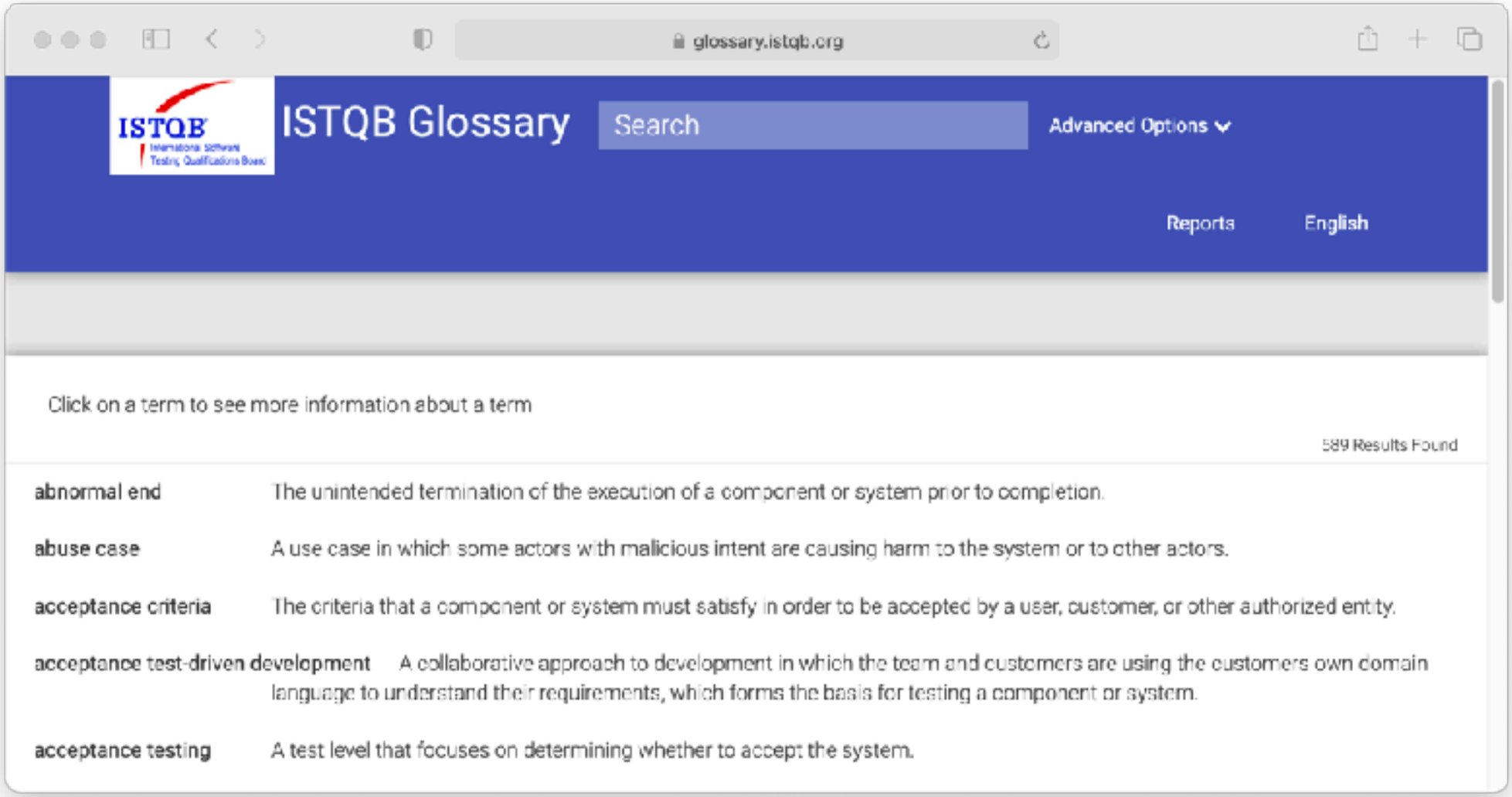
- urgency
  = the time left before measures or responses would need to be implemented
- less time available ⇒ risk becomes more critical

Risk = impact * likelihood * urgency

Good testing …

- reduces likelihood (provided sufficient test coverage)
- should reduce urgency (provided test strategy is in place)
  + When should which tests be executed?
  + What actions should be taken if tests fail?
- does not affect impact (⇒ fall-back plans, disaster scenarios)

V-Model



https://glossary.istqb.org/

Terminology (1/4)

- software testing = execution of code using combinations of input and state to reveal bugs
  + (not requirements validation! not code/design/… reviews!)
- component (under test) = any software aggregate that has visibility in the development environment (method, class, object, function, module, executable, task, subsystem, ...)
- scope of test = collection of components to be verified

implementation under test	= IUT
method under test	
object under test	= OUT
class under test	
component under test	= CUT
system under test	= SUT

Perspective of a forensic investigator dissecting suspicious samples

Terminology (2/4)

- **unit test** =
  + test scope is small executable (object of a class, method)
- **integration test** =
  + test scope is complete system or subsystem
    - (software *AND* hardware)
- **system test =**
  + test scope is a complete and integrated application

- fault-directed testing =
  + a.k.a: code-based testing, white-box testing
  + intent is to reveal faults through failures
- conformance-directed testing =
  + a.k.a: specification-based testing, black-box testing
  + intent is to demonstrate conformance to required capabilities

- confidence = assessment of the likelihood of unrevealed bugs

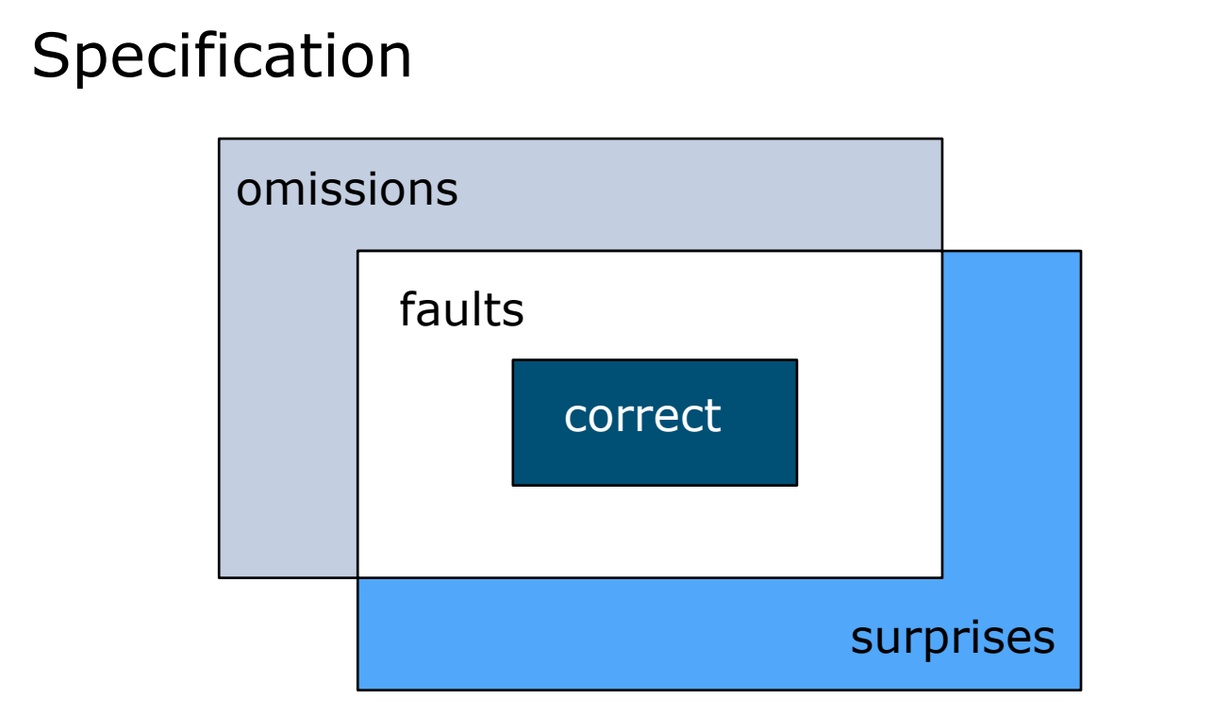
Terminology (3/4)

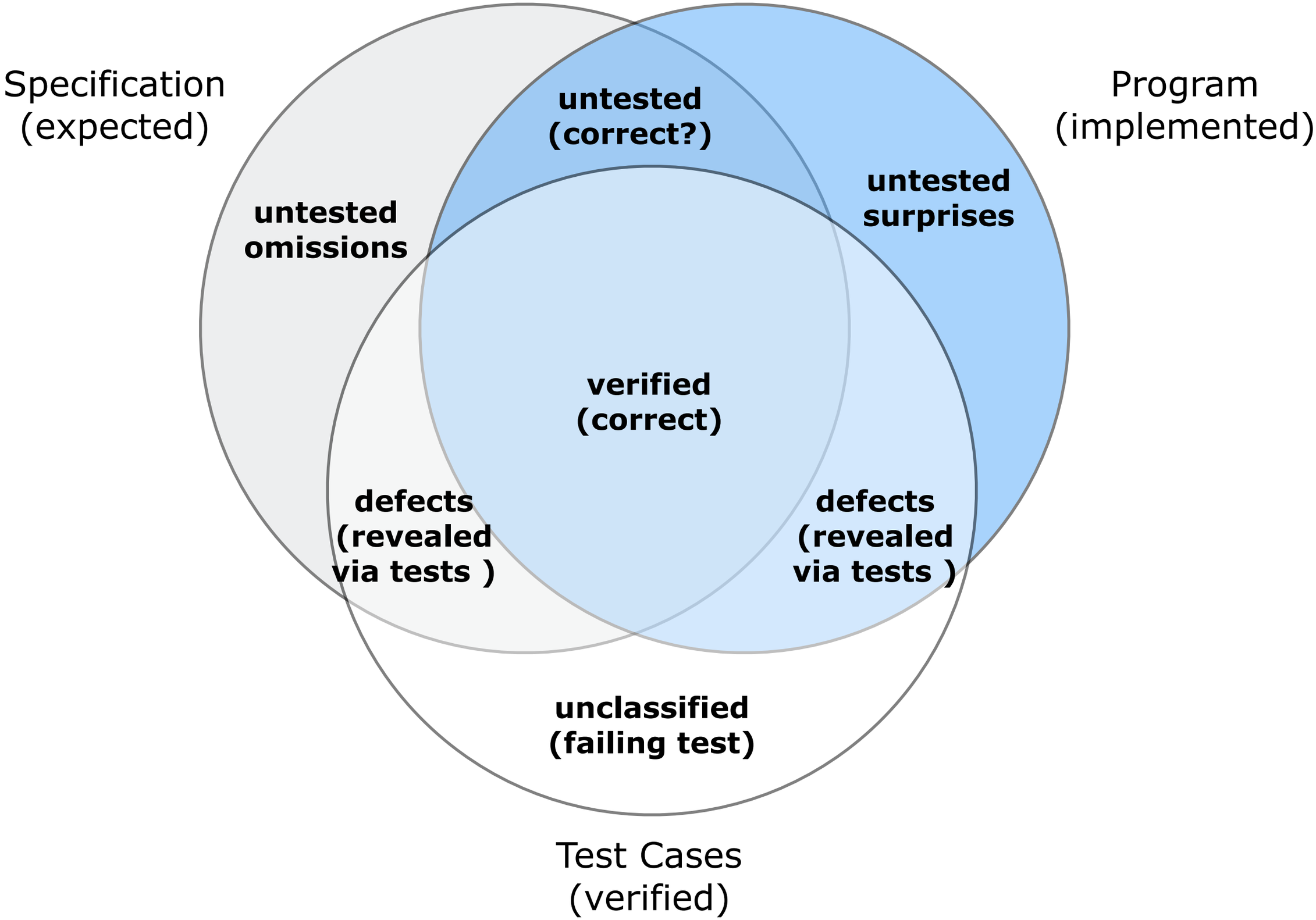
- **test case** =
  + pretest state of implementation under test
    + test inputs or conditions + expected results
- expected results =
  + generated messages + thrown exceptions
    + returned values + resultant state
- **oracle** =
  + means to produce expected result
- **test point** =
  + specific value for test case input and state variables

- domain =
  + a set of values that input or state variables of the implementation under test may take
- domain analysis:
  + places constraints on input/state/output to select test points
    > *equivalence classes (partition testing)*
    > *boundary value analysis, special values testing*

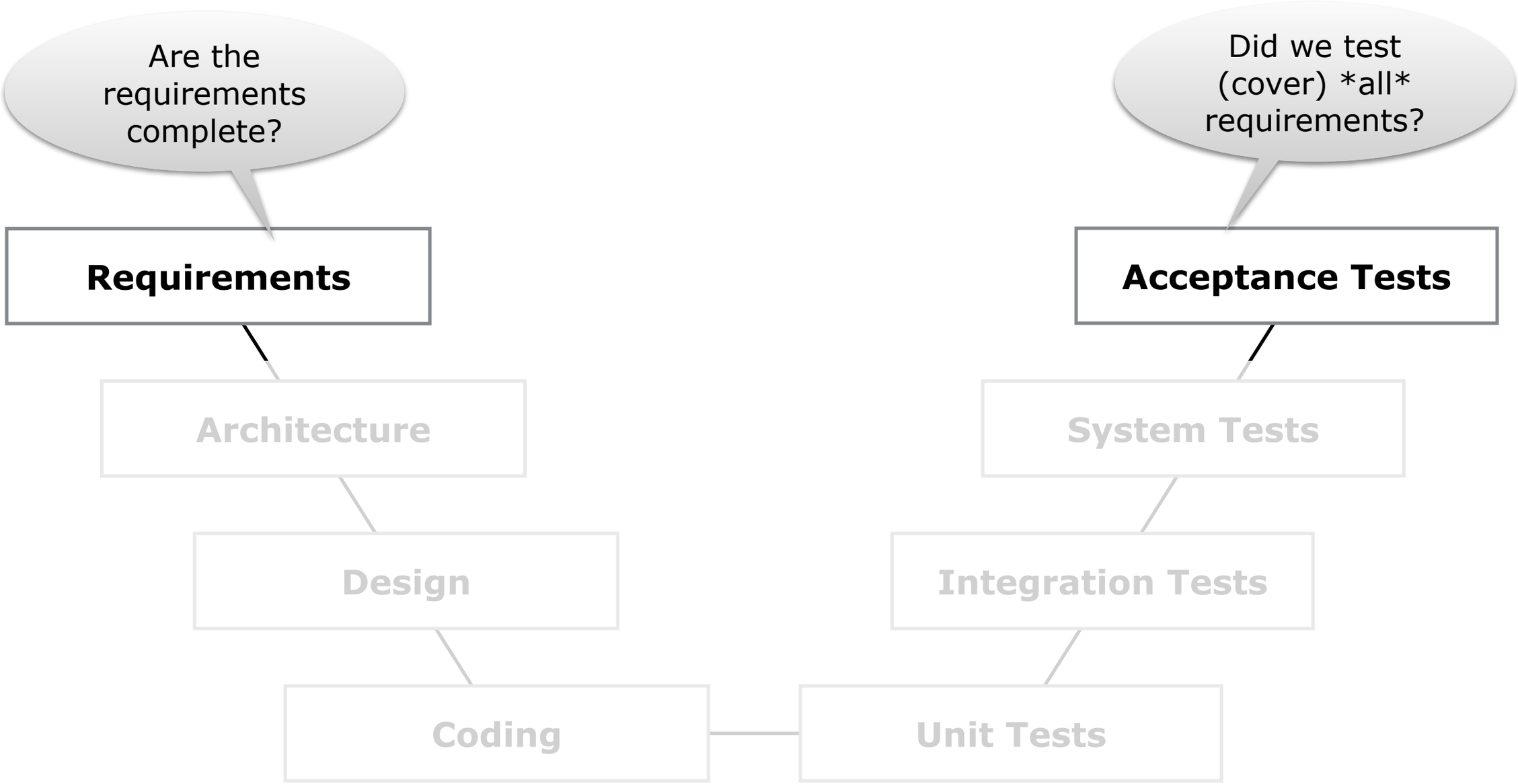
Terminology (4/4)

- defect = manifested inability of a system
- (software) fault = missing or incorrect code
  + error = human action that produces a fault
- omission = required capability that is not present
- surprise = code that does not support any required capability

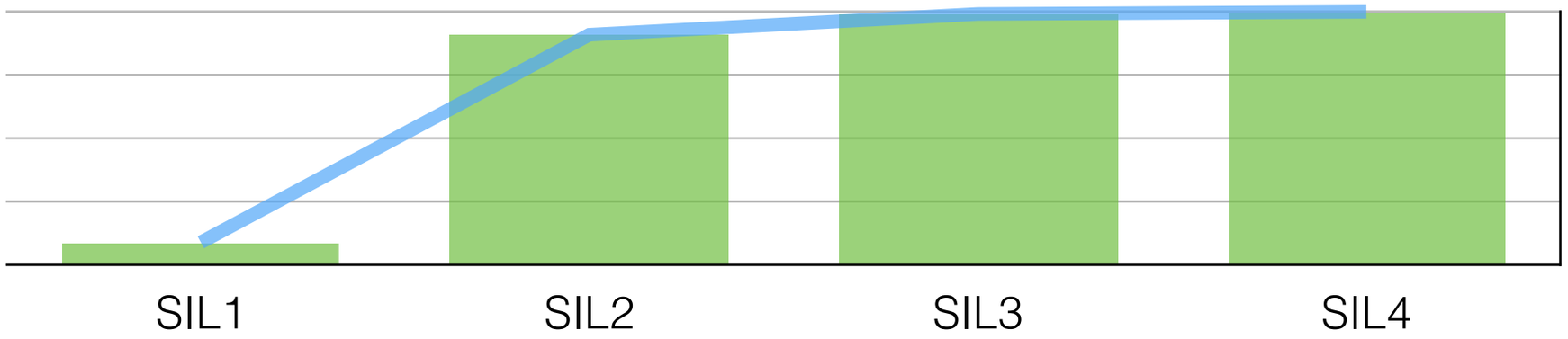



Specification
(expected)
Program
(implemented)
untested
(correct?)
untested
omissions
untested
surprises
verified
(correct)
defects
(revealed
via tests)
defects
(revealed
via tests)
unclassified
(failing test)
Test Cases
(verified)

Requirements Based Testing



Safety Integrity Levels (SIL)



SIL Level 4	The system must be able to perform its safe functions 99,99% of the time	Braking
SIL Level 3	The system must be able to perform its safe functions 99,9% of the time	Automatic train operation
SIL Level 2	The system must be able to perform its safe functions 99% of the time	Signalling system
SIL Level 1	The system must be able to perform its safe functions 90% of the time	Speed indicator

Failure Mode and Effects Analysis (FMEA)

- A step-by-step approach for identifying all possible failures in a design, a manufacturing or assembly process, or a product or service.

  + *"Failure modes"*
    - means the ways, or modes, in which something might fail. Failures are any errors or defects, especially ones that affect the customer, and can be potential or actual.

  + *"Effects analysis"*
    - refers to studying the consequences of those failures.

FMECA: Failure Mode, Effect and Criticality Analyses

+ "*Criticality Analysis*"
  - used to chart the probability of failure modes against the severity of their consequences
  - mainly when systems are already in operation

Failure Mode and Effects Analysis (Example)

	Potential Failure Mode	Potential Effects of Failures	Severity	Potential Causes of Failures	Current Process Control	Occurrence (± Likelihood)	Detection (± Urgency)	Critical (± Impact)	Risk Priority Number	Recommended Actions
Function: Dispense Fuel										
	Does not dispense fuel	- Customer Dissatisfied - Discrepancy in bookkeeping	8	- Out of fuel - Machine jams - Power failure	- Out of fuel alert - Machine jam alert - none					
	Dispense too much fuel	- Company loses money - Discrepancy in bookkeeping	8	- Sensor defect - Leakage	- none - pressure sensor					
	Takes too long to dispense fuel	- Customer annoyed	3	- Power outage - Pump disrupted	- none - none					

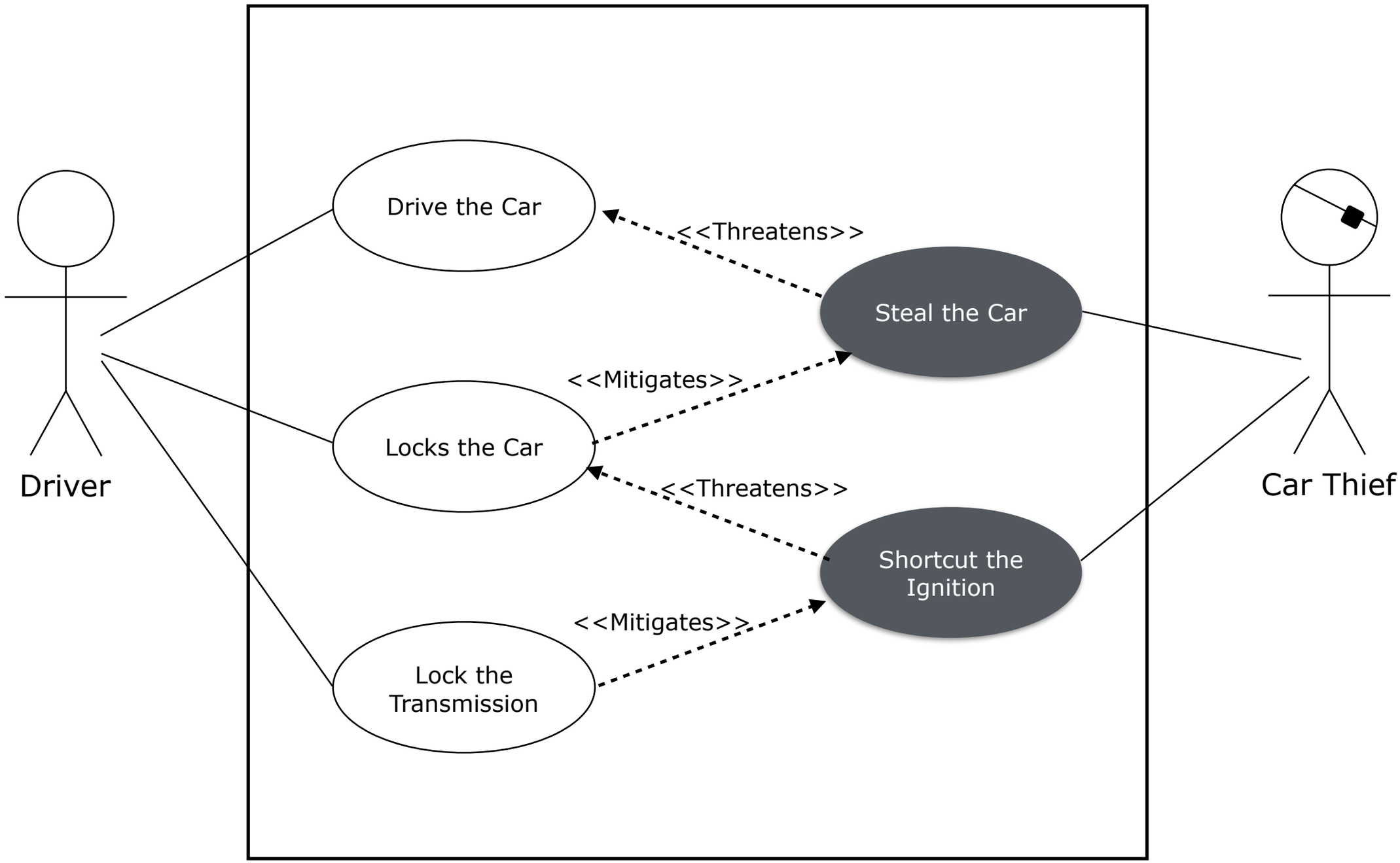
We must be able to express *"can never happen"* scenarios

Misuse Cases

- = a use case from the point of view security of an actor hostile to the system under design.
  + Results from a Failure Mode and Effects Analysis (FMEA)

Adds extra items to a use case diagram
  + Misuse case (coloured black)
  + Negative actor (marked somehow)
  + “Threatens” relationship
    - Between misuse case and ordinary use case
    - ≈ Potential causes of failures (FMEA analysis)
  + “Mitigates” relationship
    - Between misuse case and ordinary use case
    - ≈ Current Process Control (FMEA analysis)

Misuse Case (Example)



© Adapted from Ian Alexander, "Misuse Cases: Use Cases with Hostile Intent"

Safety Stories

- = if satisfied, will prevent a hazard from occurring or reduce the impact of its occurrence

Extended template for Safety Stories (Easy Requirements Syntax — EARS)

- Ubiquitous: The <component name> shall <response>
- Event Driven: When <trigger> the <system name>
- State Driven: While <in a specific state>
  the <system name> shall <system response>
- State Option: Where <feature is included> the <system name> shall <system response>
- Unwanted Behavior: If <optional preconditions> <trigger>, then the <system name> shall <system response>

Safety Stories (Example)

Several variants of stories

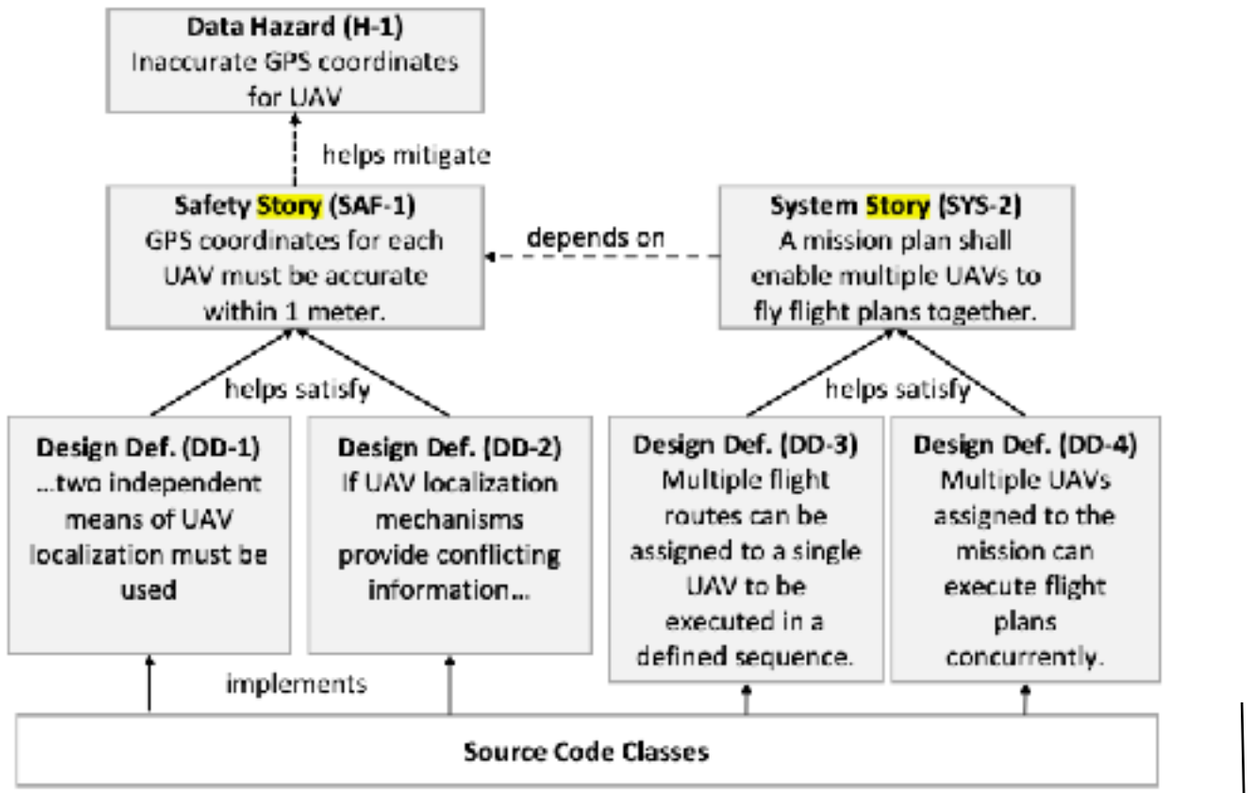
- System Story (SYS-1): A UAV shall maintain a minimum separation distance from other UAVs at all times.

- Data Hazard (H-1): Inaccurate GPS (Global Positioning System) coordinates for UAV.
  + Failure Mode: GPS provides inaccurate readings.
  + Effect: Violation of minimum separation distance between two UAVs goes undetected, and UAVs collide in midair and then crash onto bystanders.
  + Level: Critical

- Safety Story (SAF-1): The GPS coordinates of each UAV must be accurate within one meter at all times.

- Design Definition (DD-1): When the Dronology system is deployed in an urban environment at least two independent means of UAV localization must be used.

Establish traceability links!



© Adapted from Jane Cleland-Huang et. al, "Discovering, Analyzing, and Managing Safety Stories in Agile Projects,"