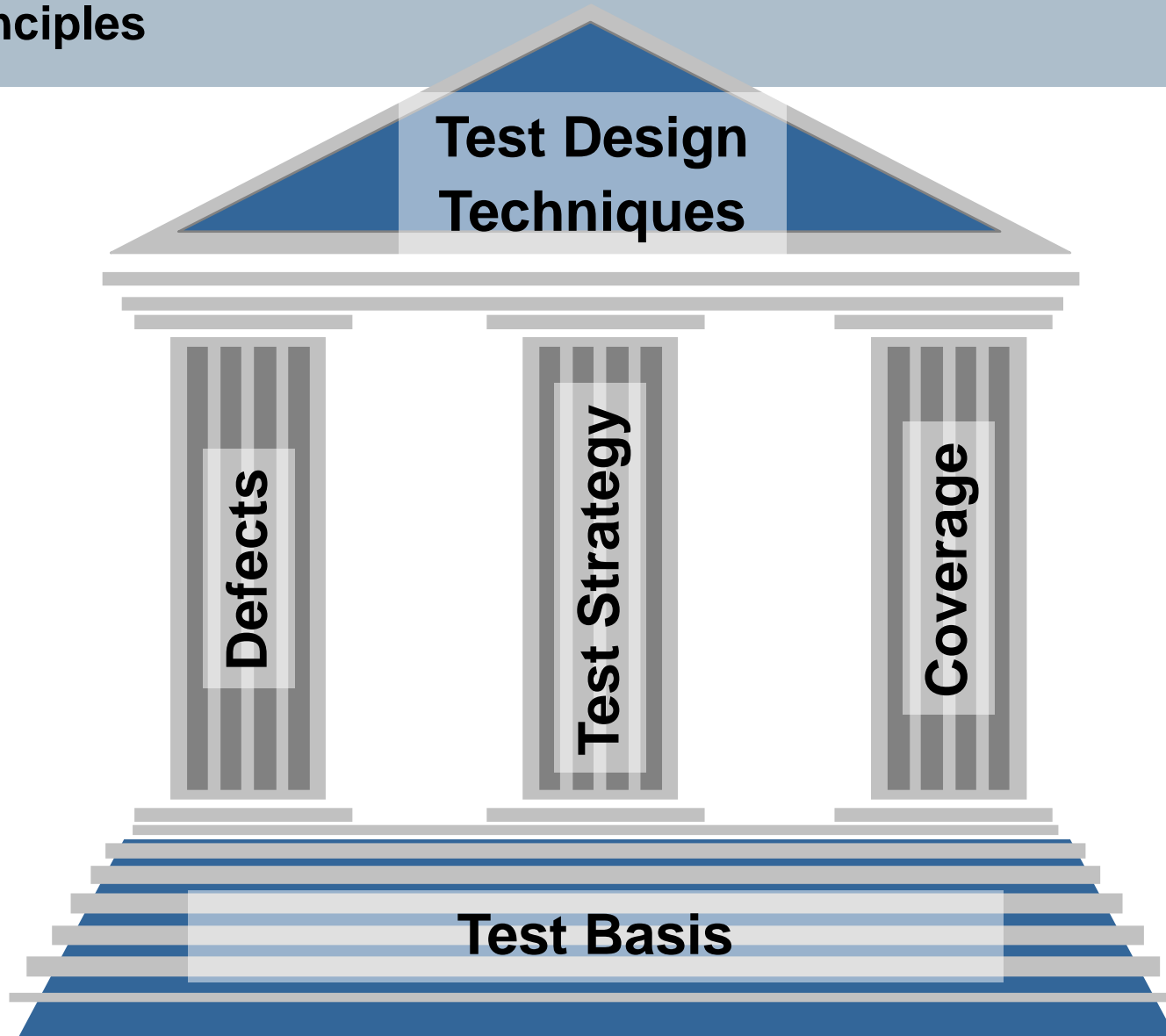
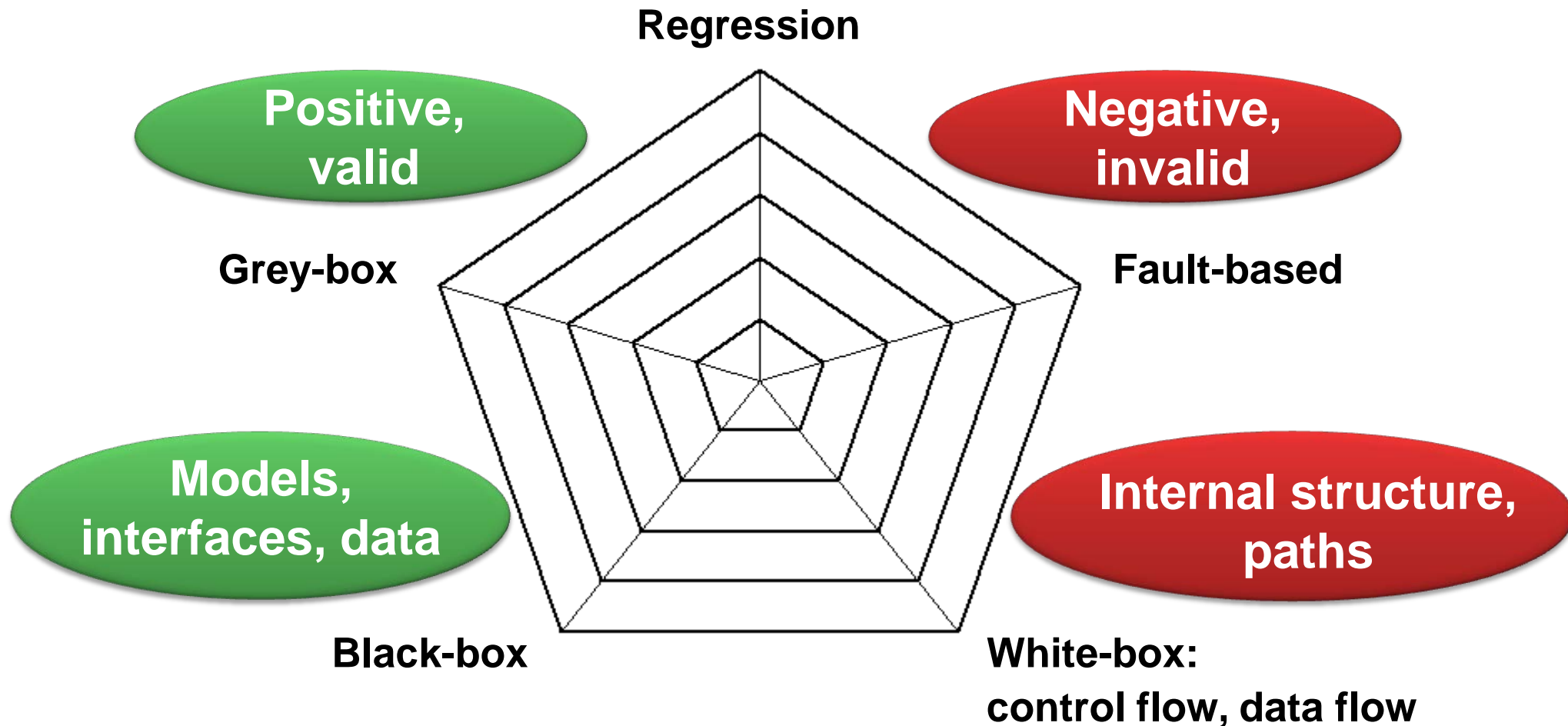


# Test design techniques

## Key principles



## Categories of test design techniques



# Summary: Building blocks of a Risk-based Test Strategy (RBT)

## Why do we test? Dimensions of testing

Prevent, protect, respond, control, influence, enable, and drive quality, support, drive, and speed up development

Coverage

Measure, assess, evaluate, predict

Goals

Demonstrate, check, confirm, verify, validate

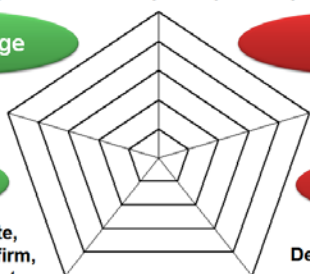
Risks

Mitigate, reduce risks, investigate, explore

Defects

Isolation & Diagnosis

Detect, search



Testing Strategy

Risk	Sub-system	P	D	RE	TE	TPN	Test Level	Test Technique	Measurements	Defects	Effort
Req. NFR		1..5	1..5	1..5	1..5	1..125					
Quality Criteria											
Claim 1											
Use Case 1											
Feature 1											
Arch. Decision											
Design Decision											
Technology Decision											
and partly completed											
Bug Category											
Risk 1											

Test Exit Criteria

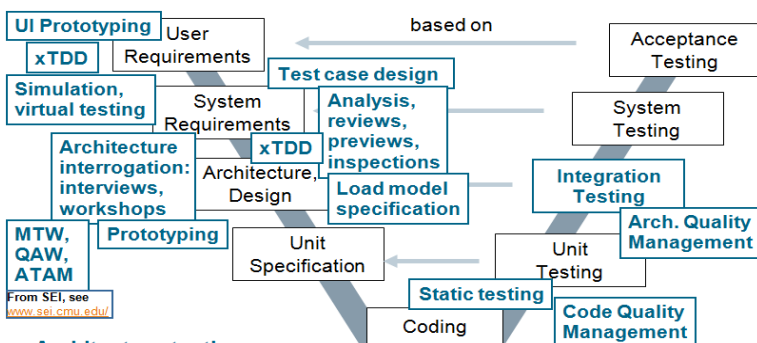
Defects

Test Progress

Coverage

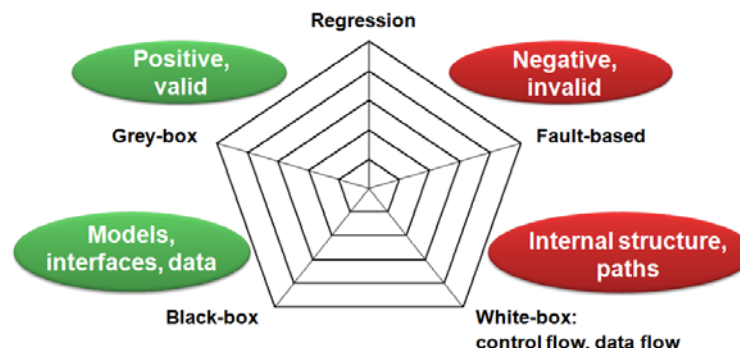
Running System  
Artifact Ready for Testing

## Test levels – Example V model with architecture testing



Architecture testing  
is any testing of architecture and architectural artifacts  
→ mapping of architectural risks to test levels

## Categories of test design techniques

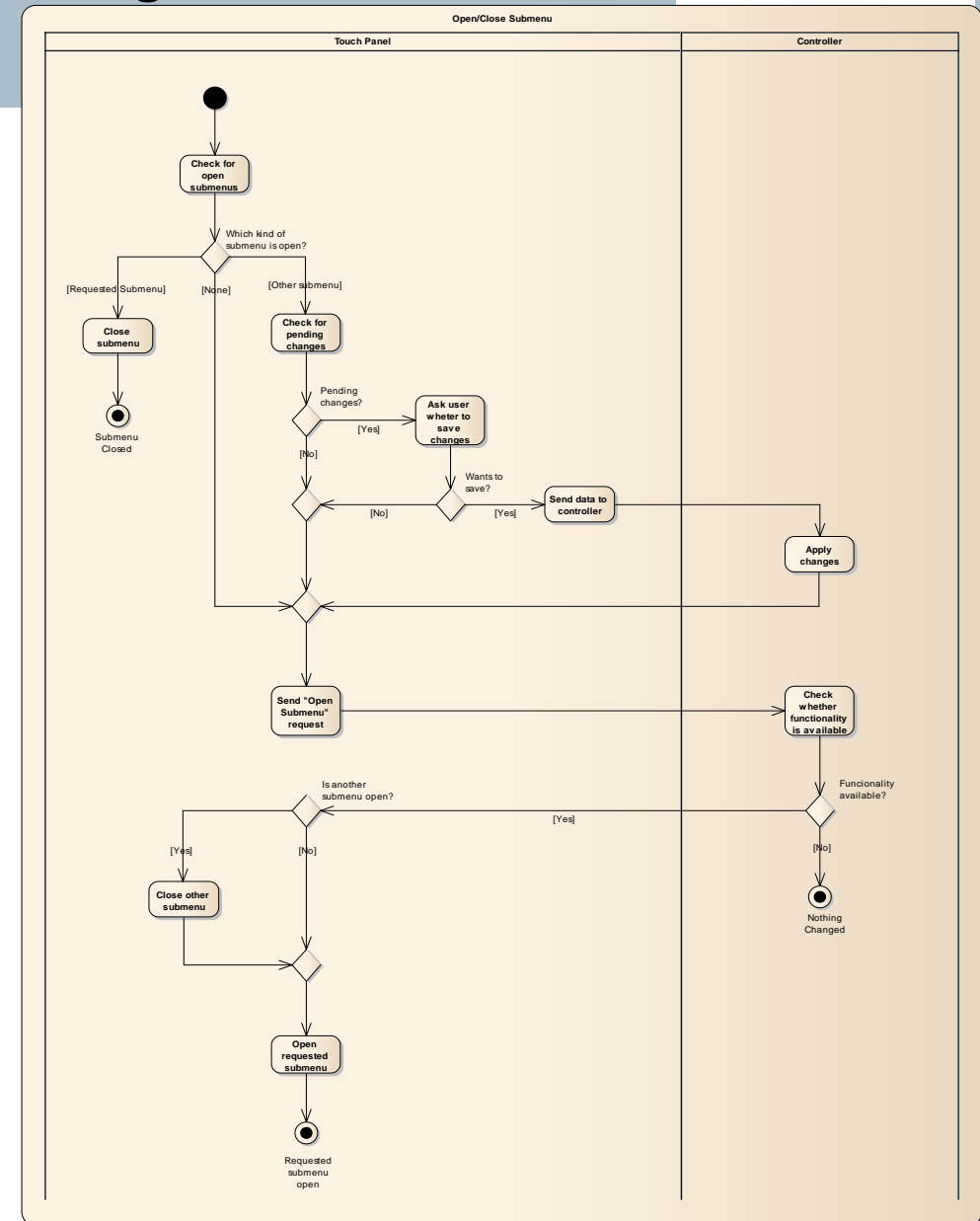


Test Design Techniques on One Page

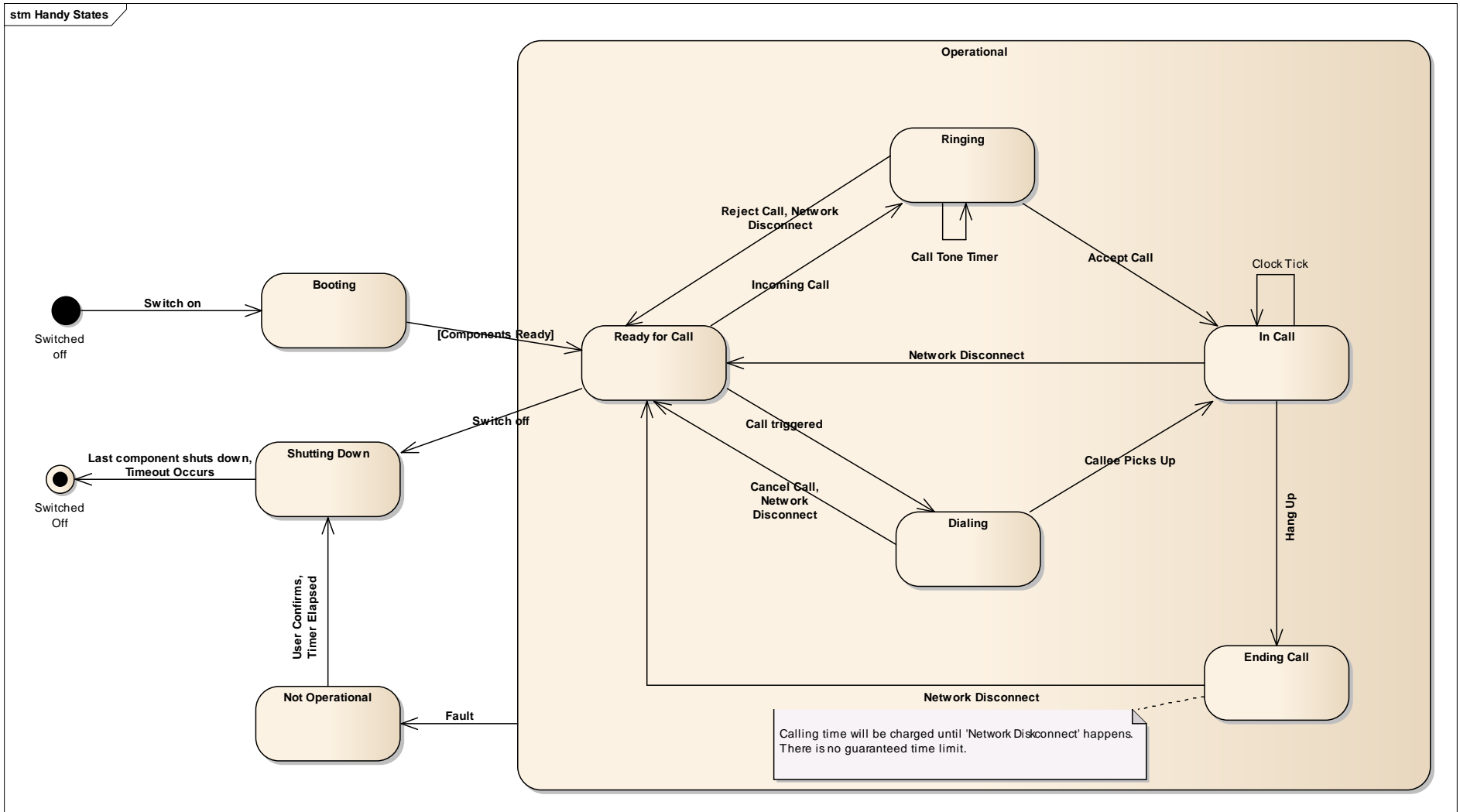
Category	Technique	Standard
Black-box (models, interfaces, data)	Standards (e.g. ISO/IEC 9126/2000, IEC 61508, norms, formal specifications, claims)	1
	Requirements-based with traceability matrix (requirements & test cases)	2
	Use case-based testing (sequence diagrams, activity diagrams)	3
	FSM (Finite State Machine) based testing (state transition diagrams, state machines)	4
	Flow testing, scenario testing, state space testing	5
	User / Operational profiles, frequency and priority / criticality (Software Reliability Engineering)	6
	Component testing, machine control	7
	Random priority testing	8
	Failures, functions, apps, user stories, processes, services, interfaces	9
	Design to contract (built-in self test)	10
Grey-box	Guarantee class partitioning	1
	Domain partitioning, category-partition method	2
	Classification tree method	3
	Boundary value analysis	4
	Test catalog / matrix for input values, input fields	5
	State-based testing (finite state machines)	6
	State effect graphing	7
	Decision tables, decision trees	8
	Combinatorial testing (generalized testing)	9
	Combinatorial testing (orthogonal / covering arrays, pair-wise, n-wise)	10
White-box (internal structure, paths)	Test catalog / matrix for input values, input fields	1
	Control flow-based	2
	Specification-based, model-based, code-based	3
	Static analysis	4
	Data flow-based	5
	Statements (CFG, nodes)	6
	Branches (CFG, transitions, links, paths)	7
	Conditions, decisions (CFG, links)	8
	Combinatorial partitioning (MC/DC)	9
	Interfaces (UML, SCL)	10
Positive, valid cases	Normal, expected behavior	1
	Unusual, unexpected behavior	2
	Invalid, unexpected behavior	3
	Invalid, unexpected behavior	4
	Invalid, unexpected behavior	5
	Invalid, unexpected behavior	6
	Invalid, unexpected behavior	7
	Invalid, unexpected behavior	8
	Invalid, unexpected behavior	9
	Invalid, unexpected behavior	10
Fault-based	Risk-based	1
	Systematic failure analysis (Failure Mode and Effect Analysis, Fault Tree Analysis)	2
	Check patterns (e.g. by James A. Whittaker, Jon Knight)	3
	Error catalogs, bug taxonomy (e.g. by Boris Beizer, Cem Kibar)	4
	Test patterns, standard, well-known bug patterns or produced by a root cause analysis	5
	Bug reports	6
	Test model dependent on used technology and nature of system under test	7
	Test patterns (e.g. by Robert Binder, Questioning patterns (Q-patterns) by Ugo Krieger)	8
	Ad hoc, creative, based on experience, check lists	9
	Error guessing	10
Regression (selective retesting)	Retest all	1
	Retest by risk, priority, severity, criticality	2
	Retest by profile, frequency of usage, parts which are often used	3
	Retest changed parts	4
	Retest parts that are influenced by the changes (impact analysis, dependency analysis)	5
	Retest parts that are influenced by the changes (impact analysis, dependency analysis)	6
	Retest parts that are influenced by the changes (impact analysis, dependency analysis)	7
	Retest parts that are influenced by the changes (impact analysis, dependency analysis)	8
	Retest parts that are influenced by the changes (impact analysis, dependency analysis)	9
	Retest parts that are influenced by the changes (impact analysis, dependency analysis)	10

# Use case-based testing – Scenario testing

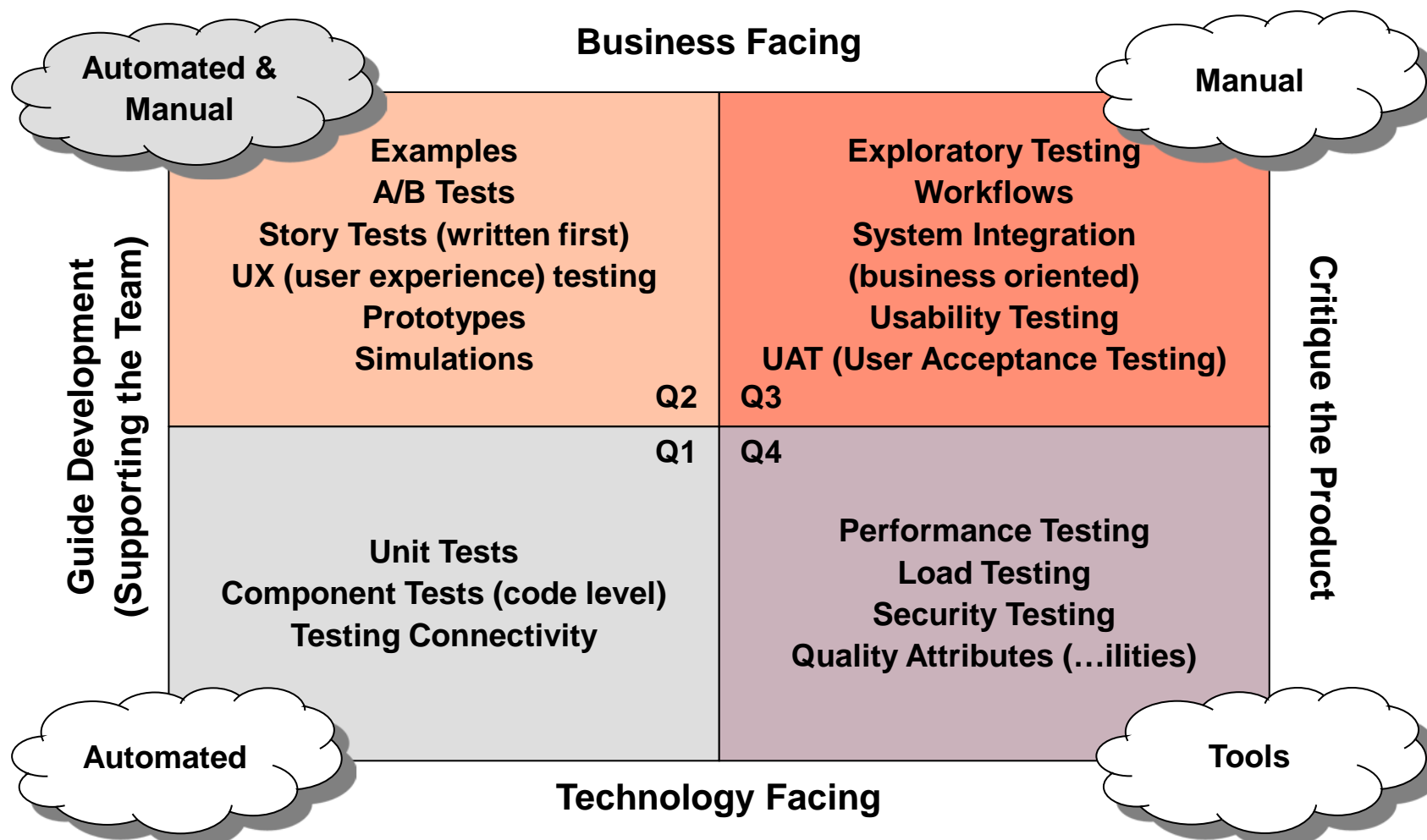
## Example: Activity diagram



# Example: Phone system

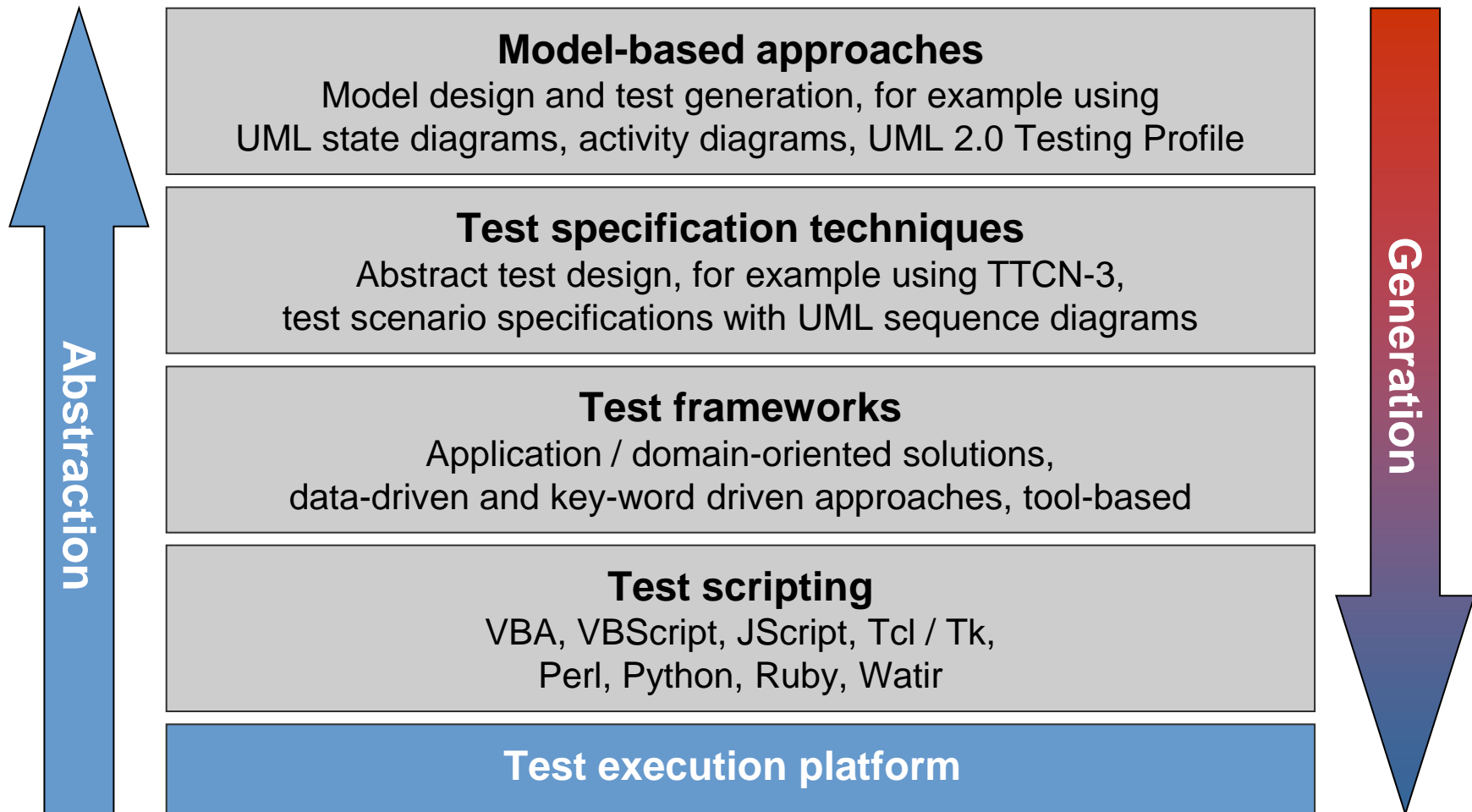


# Agile testing quadrants

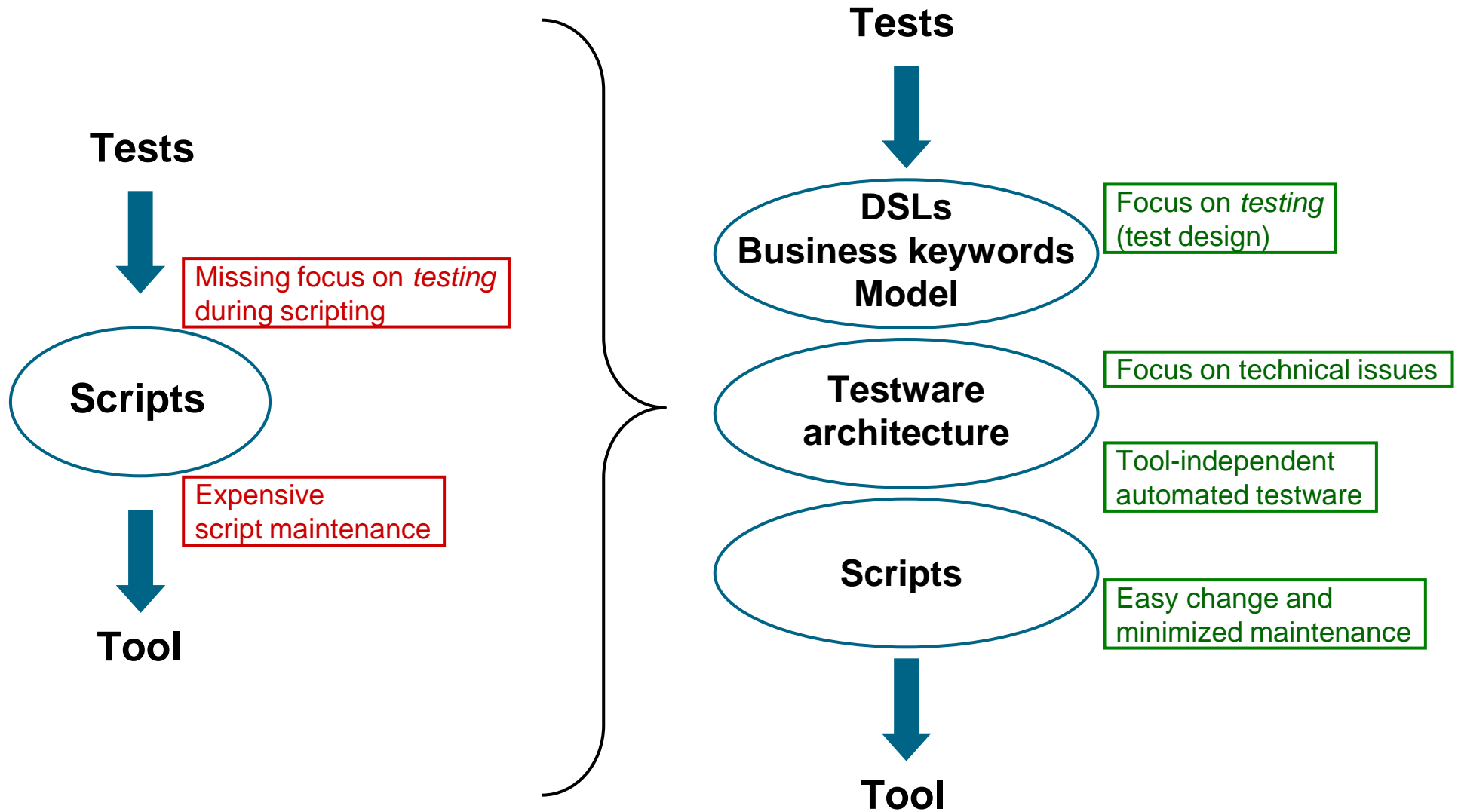


Reference: Brian Marick, Lisa Crispin, Janet Gregory

# Test automation is software development

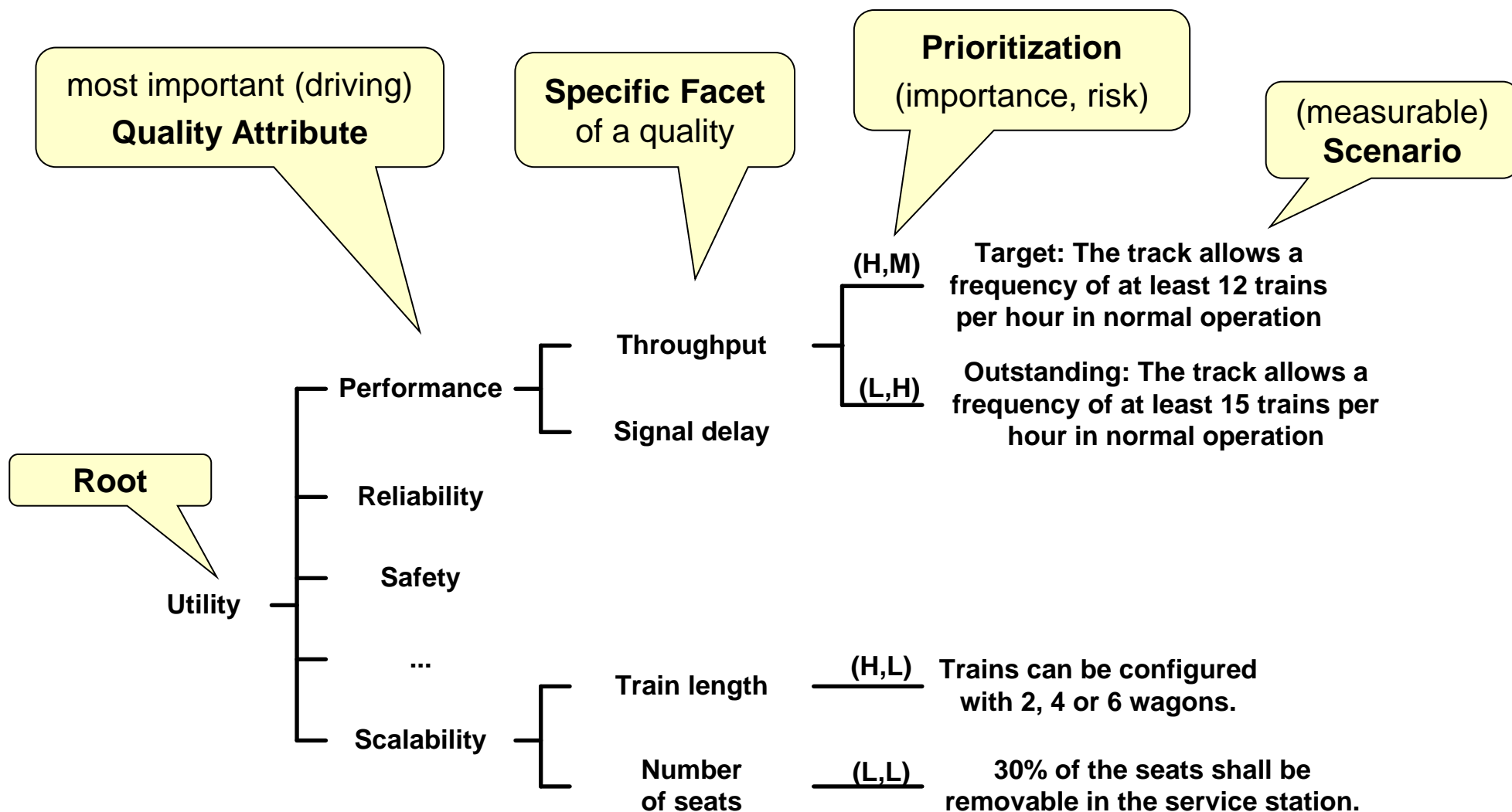


# Levels of abstraction

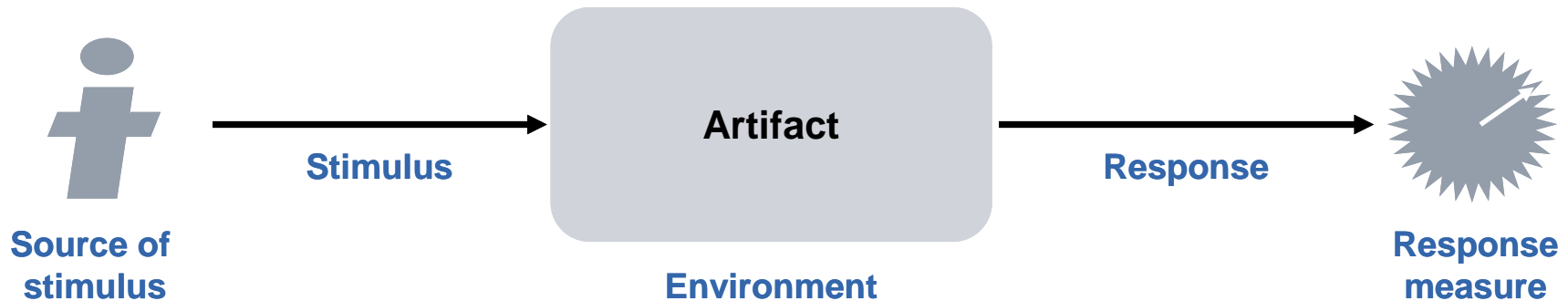




# Attributes of the "Utility Tree"



# Scenario description Template



- Source of stimulus** Who/what initiates the scenario.
- Stimulus** Which periodic, stochastic or sporadic event initiates the scenario.
- Artifact** What is the relevant unit; e.g. a (part of a) system or a feature.
- Environment** What is the environmental condition for this scenario;  
e.g. normal, startup / shut down, maintenance, emergency, overload, etc.
- Response** How does the artifact react to the event in the given environment.  
This may cause an environment change (e.g. from normal to shutdown mode).
- Response measure** How can the response be measured, using indicators like:
- the time it takes to process the event (latency or deadline); or the variation in time (jitter)
  - the amount of data, material or energy that can be processed in a particular time interval (throughput)
  - or a characterization of the events that cannot be processed (e.g. miss rate, data / energy / material loss)

## Exercise / Discussion

### “Performance of Image Processing – Let’s Test”

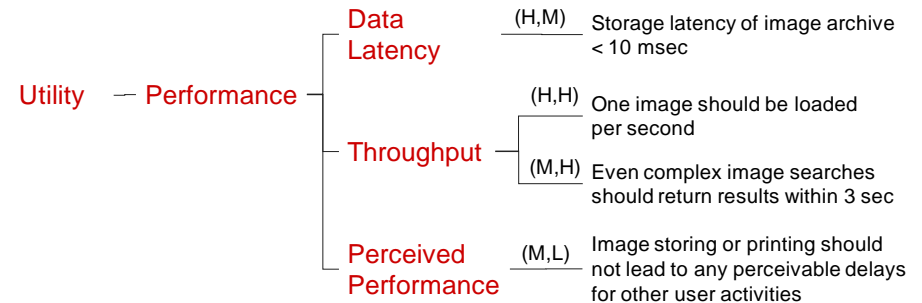
#### We built our quality attribute scenario...

Whenever a user searches for images in the User Interface (using keywords or advanced search criteria), finding and displaying the results should never take longer than 3 seconds.

#### Now let’s test this!

- What **test types** are needed?
- Which **test levels** should be used?
- What potential **obstacles** for writing test cases do you see?
- Which additional **input** is necessary? Which **stakeholders** can provide it?
- How can you – as Test Architects – **support** quality attribute testing activities?

#### ... derived our utility tree ...



#### ... and selected our design strategies and tactics

##### Performance of Image Processing

##### Resource demand

- Use advanced image processing algorithms (if possible GPU based)
- Introduce caching strategies
- Either allow upper bound of clients or scale-out mechanisms depending on resources

##### Resource management

- Introduce a pool of worker threads for background loading, storing, printing
- Don't copy mass data but use meta files and refs for image processing/copying
- Use thumbnail images for browsing
- Only use full resolution when images are selected
- Apply eager loading

##### Resource arbitration

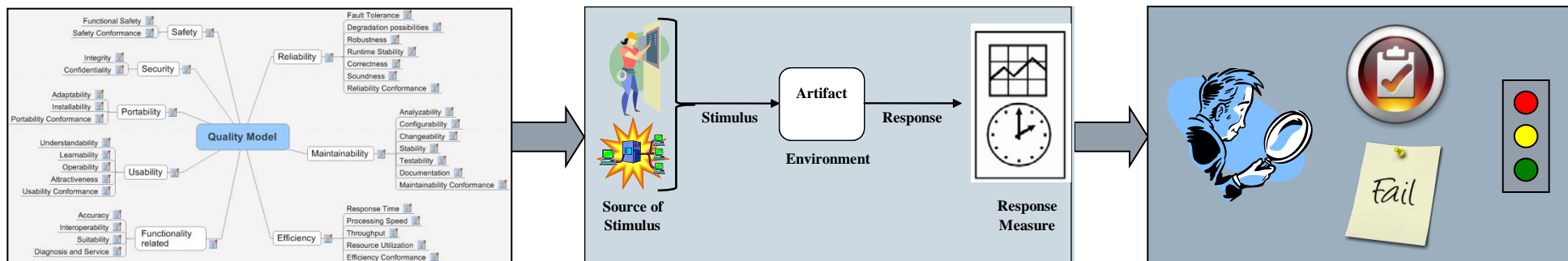
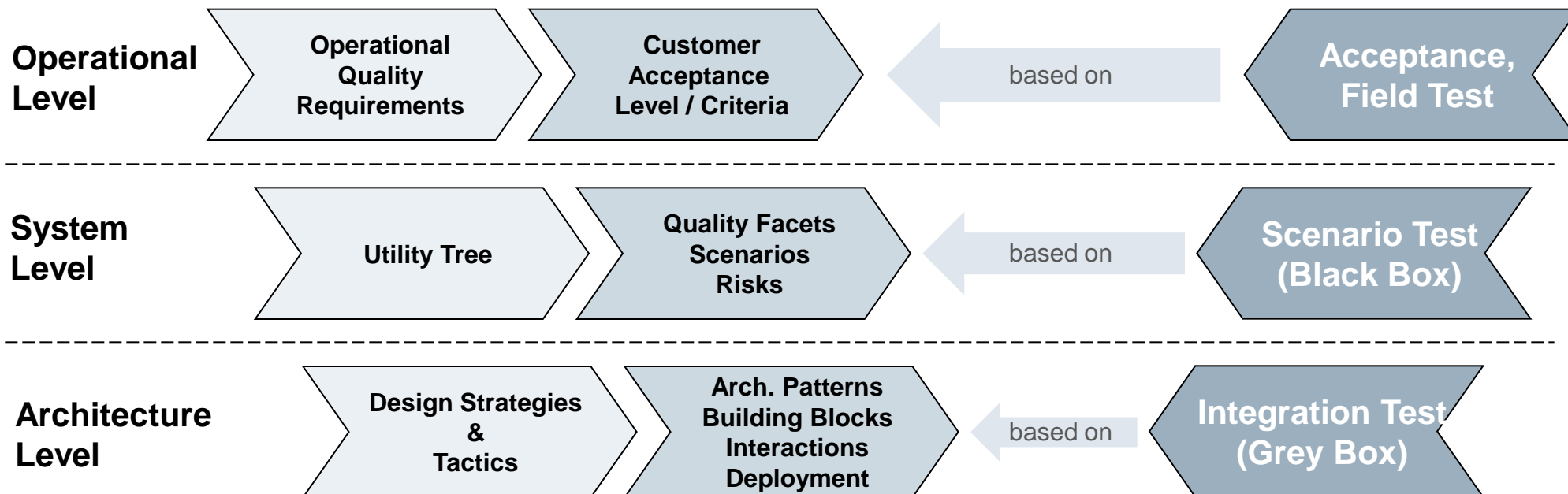
- Schedule resources preferably for processing visible images

There are various patterns for implementing some of the tactics such as

- Caching
- Lazy Evaluation
- Coordinator
- Eager Loading
- Evictor & Activator
- Half Sync / Half Async
- Command Processor



# Testing quality attributes Approach



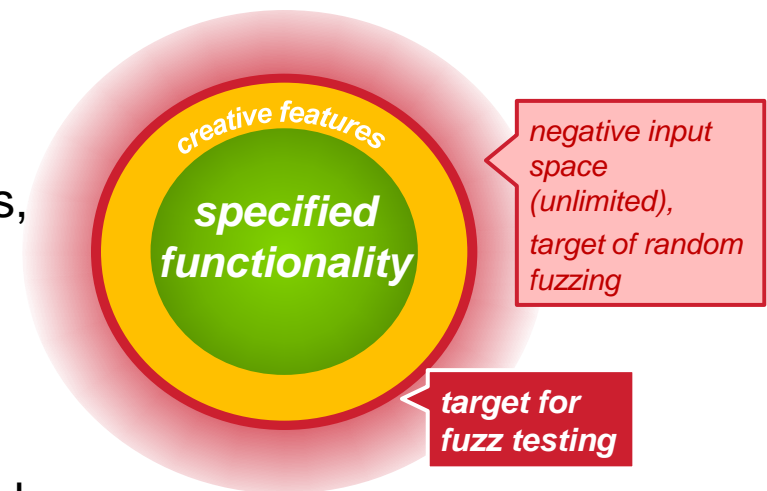
# Security Testing

## Fuzz Testing Approach

- Fuzzing originally describe the generation of randomly generated test vectors (Miller et. al. in the early 1990s)
- **Random fuzzing:** has close to zero awareness of the tested interface.
- **Mutation-based fuzzing:** mutate existing data samples to create test data, breaks the syntax of the tested interface into blocks of data, which it semi-randomly mutates.

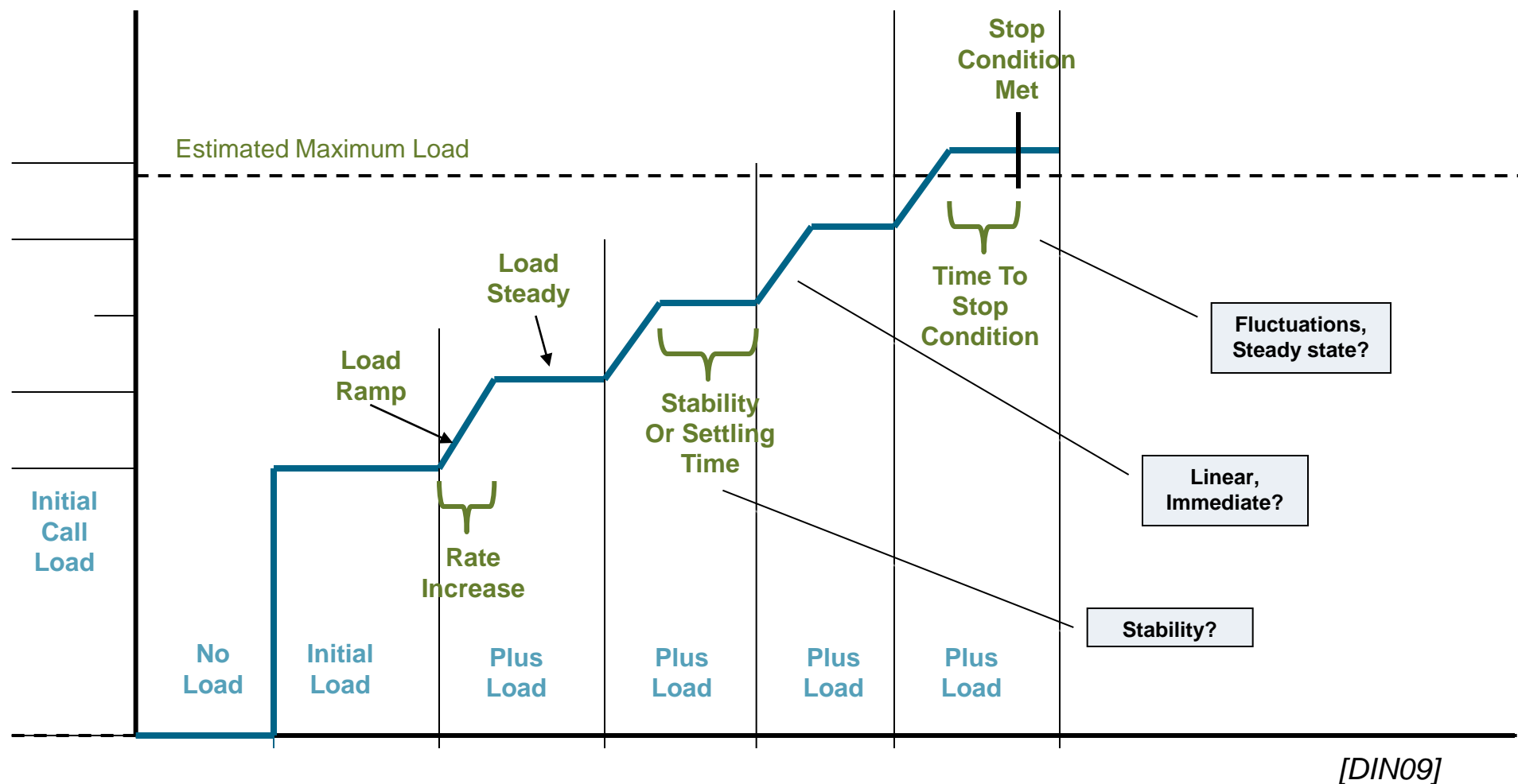
### Model-based fuzzing

- uses models of the input domain (protocol models, e.g. context free grammars), for generating systematic non-random test cases
- in security testing purposes, the models are augmented with intelligent and optimized anomalies that will trigger the vulnerabilities in code
- finds defects which human testers would fail to find



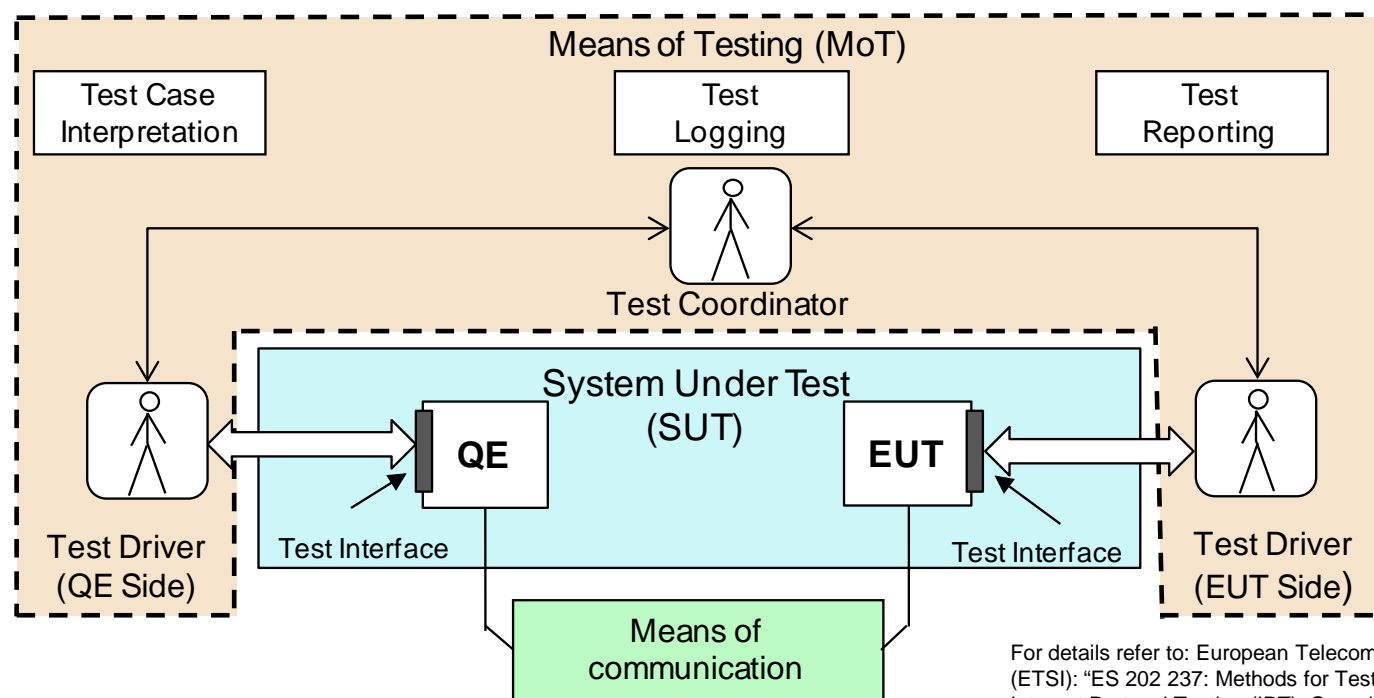
See also: [TAK08]

# Performance Testing Benchmarking Procedure



## Interoperability testing with GAIT

- Generic Approach to Interoperability Testing (GAIT) v1.2.1 (2010)
- Defined for software interoperability testing, without focusing on a specific software domain.
- QE – Qualified Equipment



For details refer to: European Telecommunications Standards Institute (ETSI): "ES 202 237: Methods for Testing and Specification (MTS); Internet Protocol Testing (IPT); Generic Approach to Interoperability Testing", version 1.2.1, 2010.

# Interoperability testing

## Classification of IOP checking levels

### Interaction Scenario

- **what**: the required sequence of messages is validated
- **how**: timeout events indicate that some messages are not sent according to the specified sequence

### Message Type

- **what**: check that IOP *message structure profile* constraints are fulfilled
- **how**: type checking upon receipt of a message from SUT according to IHE IOP integration profiles

### Fields Conditionality

- **what**: the conditionally constraints across the fields within the same message is validated (e.g., if field1 is present, then field2 must be also present )
- **how**: special checking functions are used

### Message Content

- **what**: the content of messages is inspected against expected values, code sets (tables), values imposed by standards, etc.
- **how**: message content checking functions

### Semantic Correlations

- **what**: correlation of pieces of information *across different messages* within the flow has to be verified (e.g., for updating a patient, a Patient ID used in a previous step has to be used)
- **how**: using message tuning functions with semantic parameters or using semantic checking functions



How can you lead without formal power?

*An attempt at definition:*

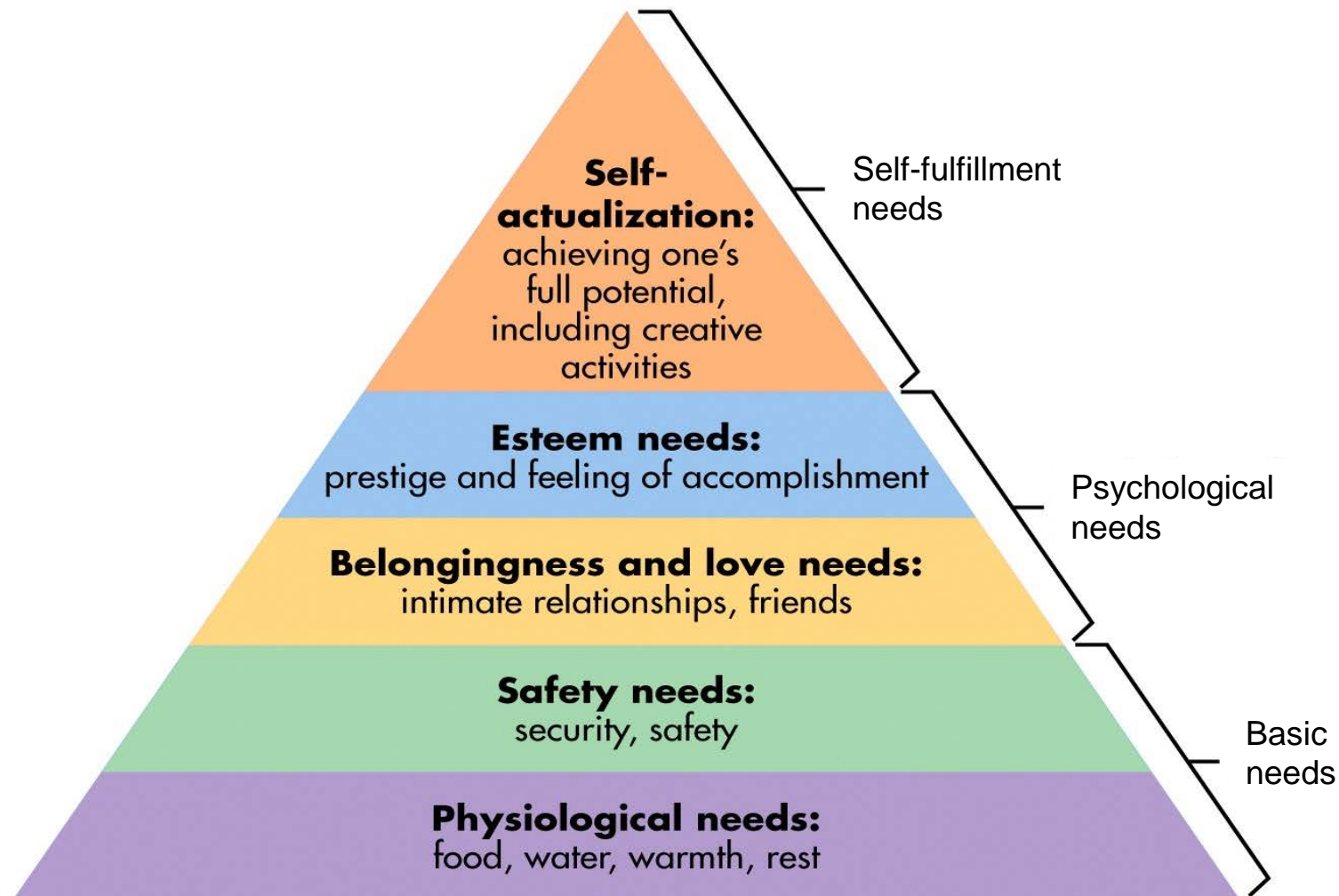
# Leadership

means **motivating** others

to **achieve** a common **goal**.

→ Set direction, align resources, inspire action, be responsible for results

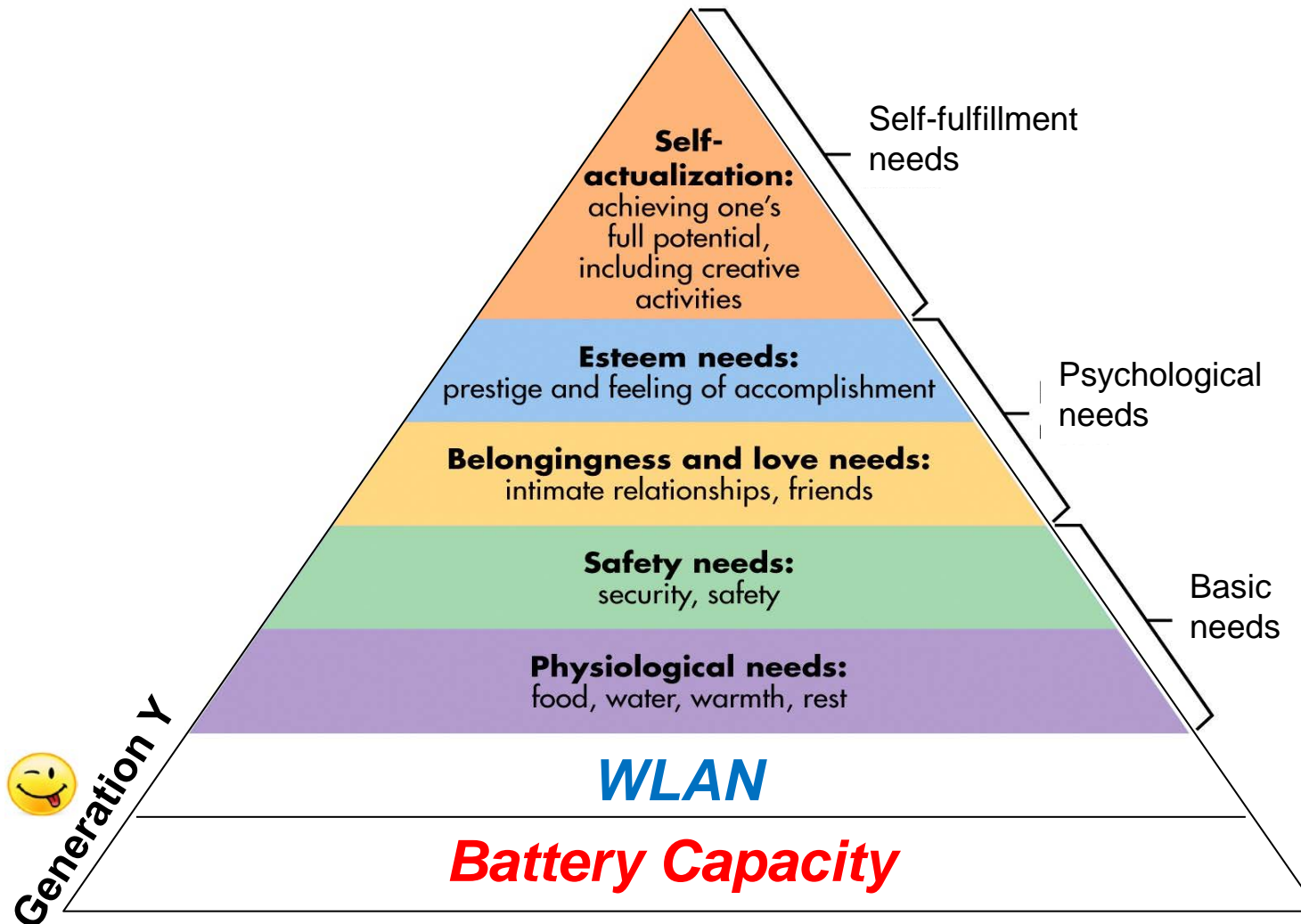
If you want to motivate – you should know the motives



Abraham Maslow  
(1908 –1970)

Maslow's Pyramid of Motivation

If you want to motivate – you should know the motives



Abraham Maslow  
(1908 – 1970)

Maslow's Pyramid of Motivation

# Natural and vested authority two different sources of power



## Authority



### Borrowed power vested by someone else

A role  
with specified powers,  
duties and responsibilities

e.g. minister of transport,  
project leader,  
department chief.

→ What is given  
can be taken away.

Helps to fill  
a role

### Natural power earned by yourself

A competence  
developed through practice,  
founded on personal attributes

e.g. ability to communicate,  
solve conflicts,  
convince others.

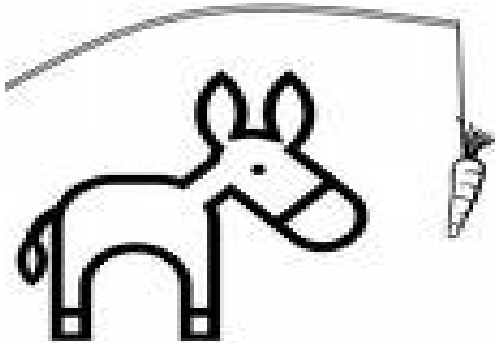
→ You take your skills with you  
wherever you go.

## How *NOT* to motivate – Carrot and KITA



**You need a maximum of controlling effort if you use force instead of motivation. Manipulation instead of motivation will lead to frustration and a loss of trust.**

## How to motivate: The carrot only helps...



...if donkey is hungry...



...if donkey can reach it and...



...if donkey likes it,...



...if the task fits donkey's ability.

## To use force and manipulation fosters resistance

**You need a maximum of controlling effort if you use force instead of motivation.**



**If you use manipulation it will eventually lead to frustration and a loss of trust.**

# What do you do if someone is not delivering as expected?

## Knowing

- Does he know what you expect him to do?
- Does he know the goal and why it is important?

## Able

- Does he know how to do it?
- Does he have the necessary skills and experience?

## Willing

- Is he motivated to do it?
- What would motivate him?

## Allowed

- Do the circumstances allow him to fulfill the task?
- What barriers have to be removed?



# The Situational Leadership Model can provide orientation

**Supportive**  
leadership  
style



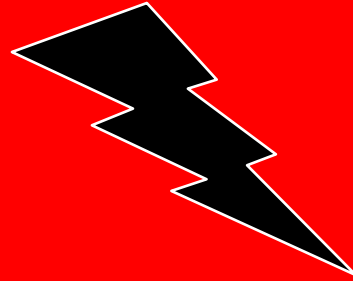
**Cooperative**  
leadership  
style

**Delegative**  
leadership  
style

**Directive**  
leadership  
style

# “Shadow Styles“

Better to avoid the dark side



**Charity**  
~~leadership~~ style

**Chitchat**  
~~leadership~~ style

**Laisser-faire**  
~~leadership~~ style

**Authoritarian**  
~~leadership~~ style



# There are ways to become more effective

