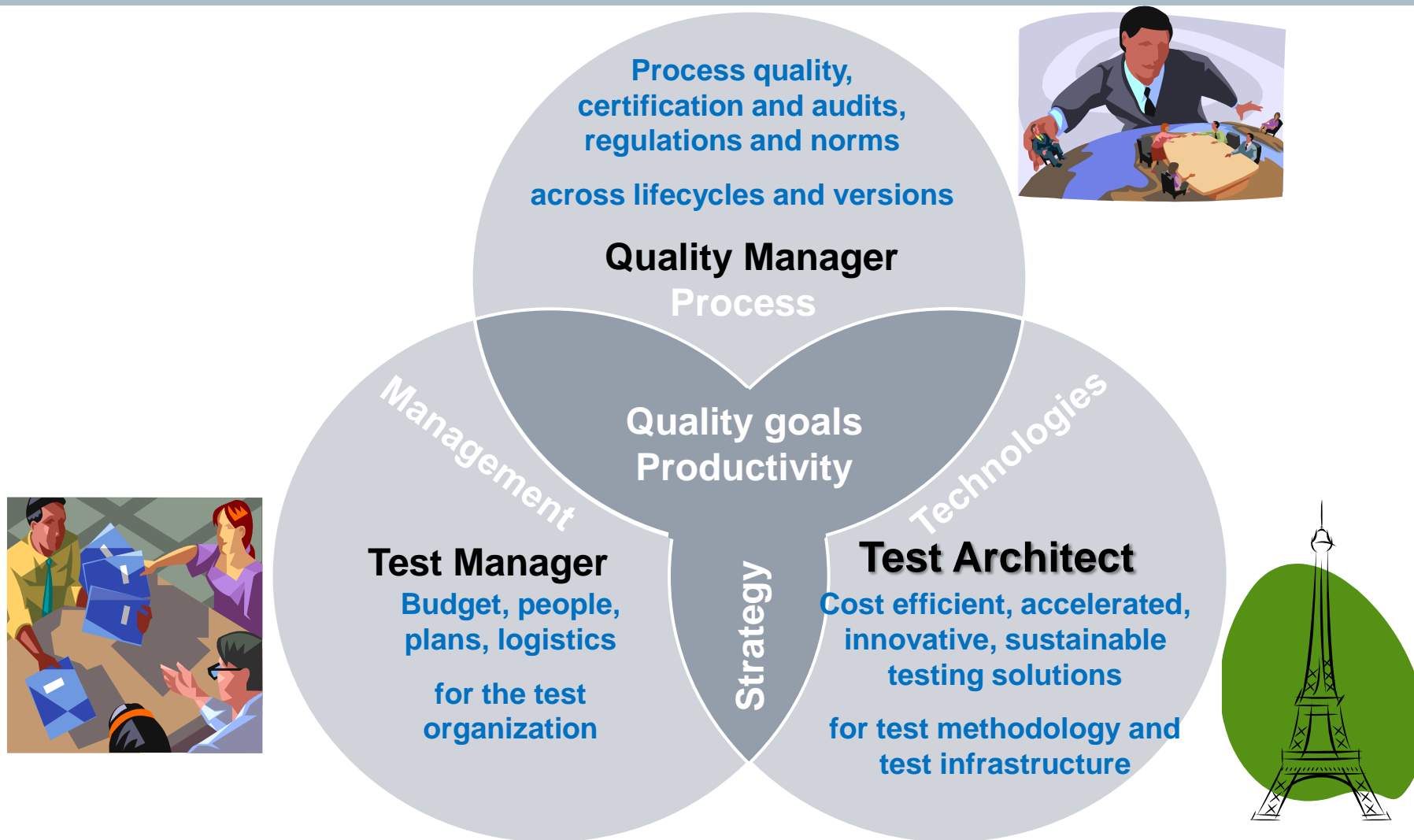


# Driving triumvirate for testing & quality engineering



# As a **Test Architect** You have **one Role** – but you are wearing **two Hats**



## **Test Expert**

for the system under test (SUT)

- Design the test approach
- Apply innovative test technologies
- Drive the quality of the SUT



## **Software / System Architect**

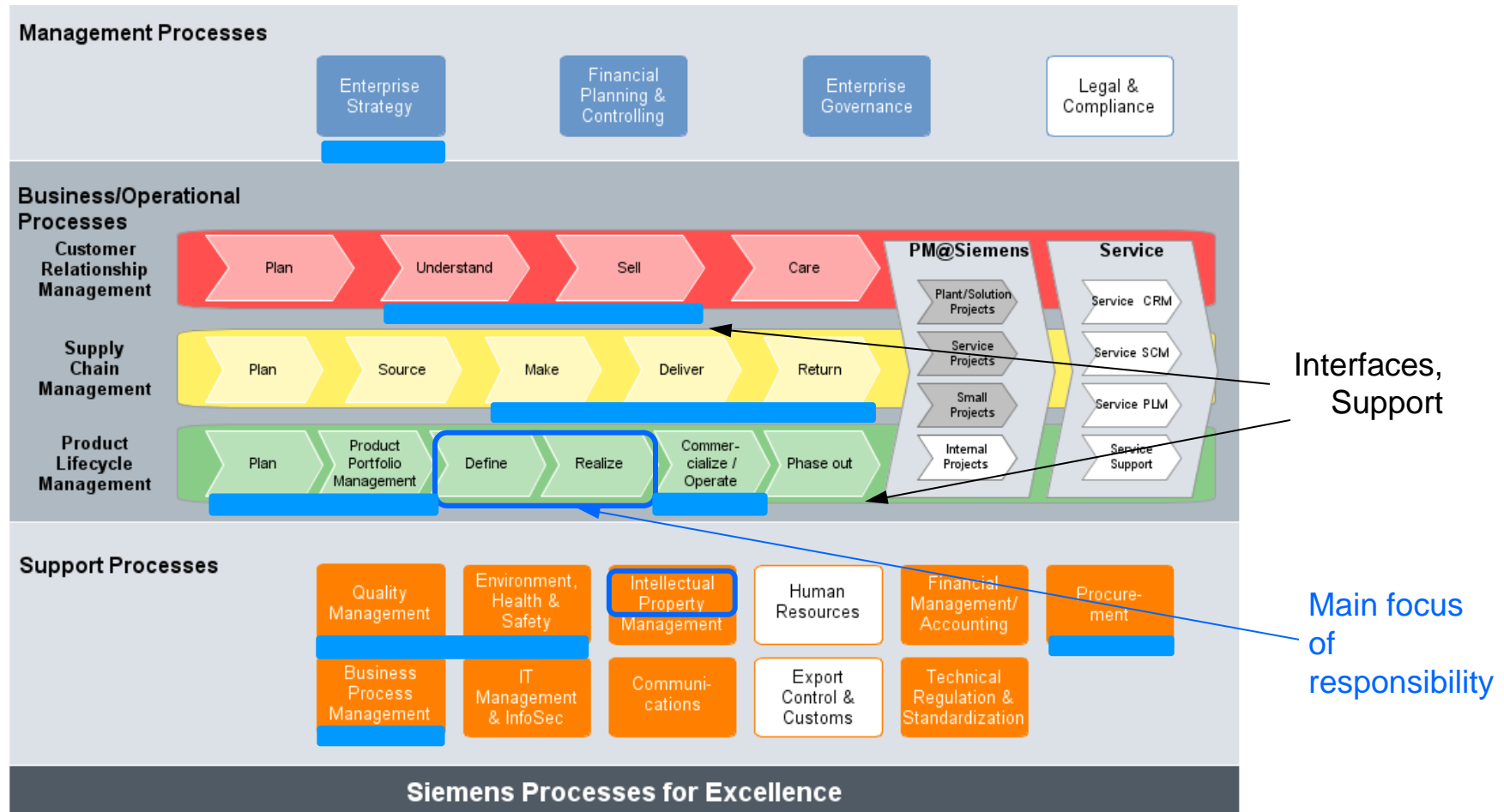
for the test system

- Design and realize the test architecture
- Apply innovative software technologies
- Drive the quality of the test system

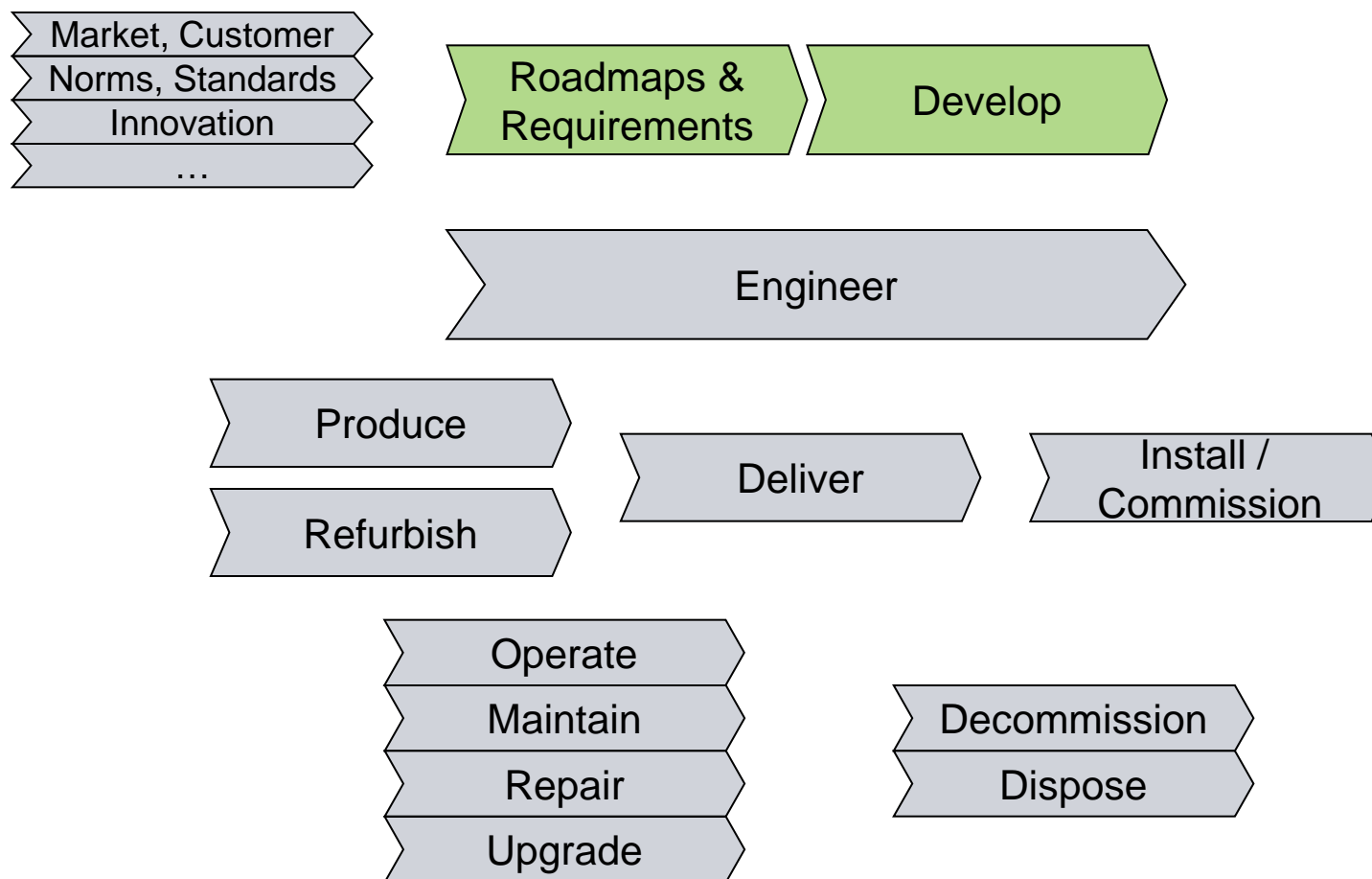
This is the  
architect's job!



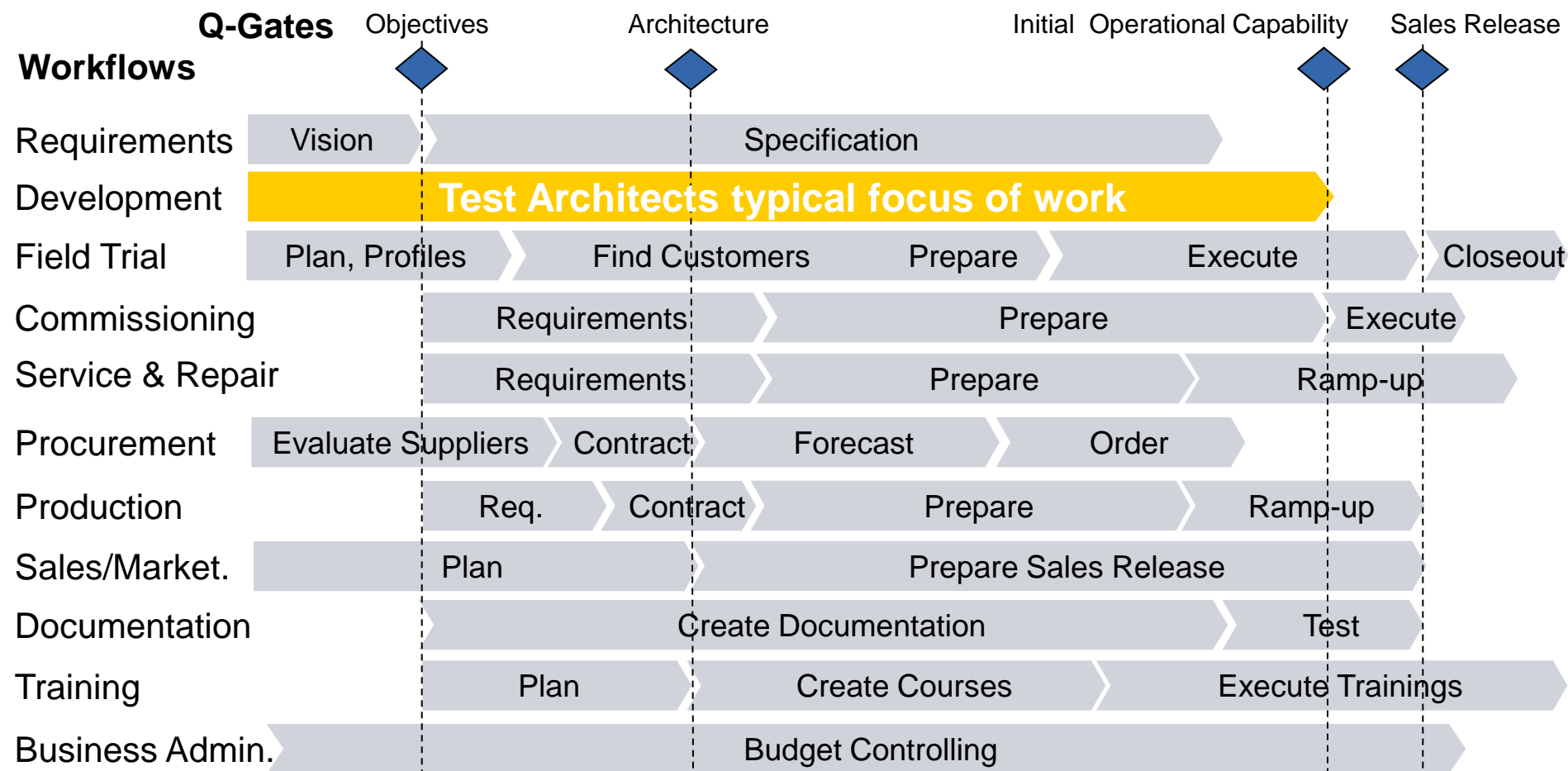
# Test Architect's scope in SIPEX



# Major system lifecycle elements that influence a system architecture

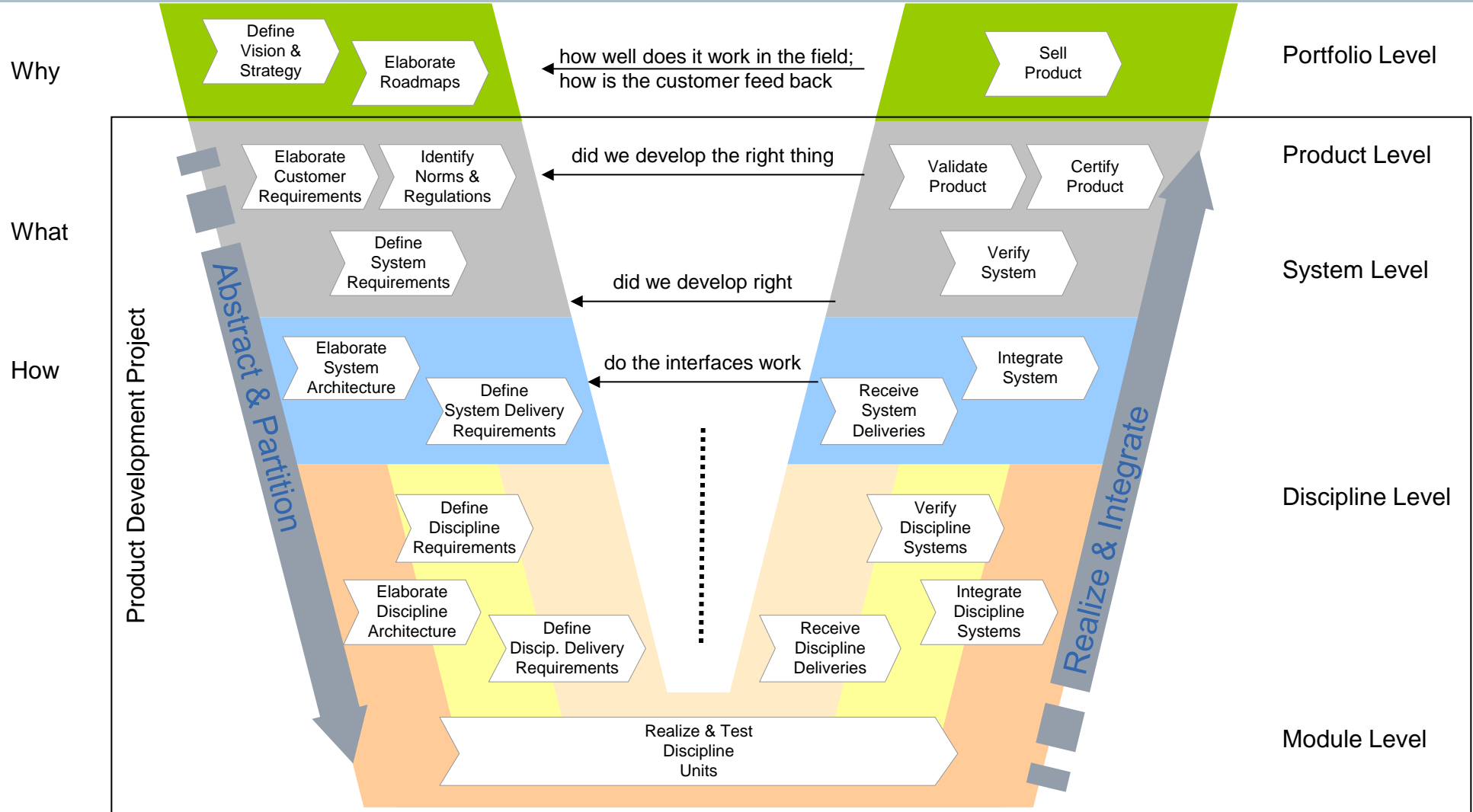


# Parallelism of Processes

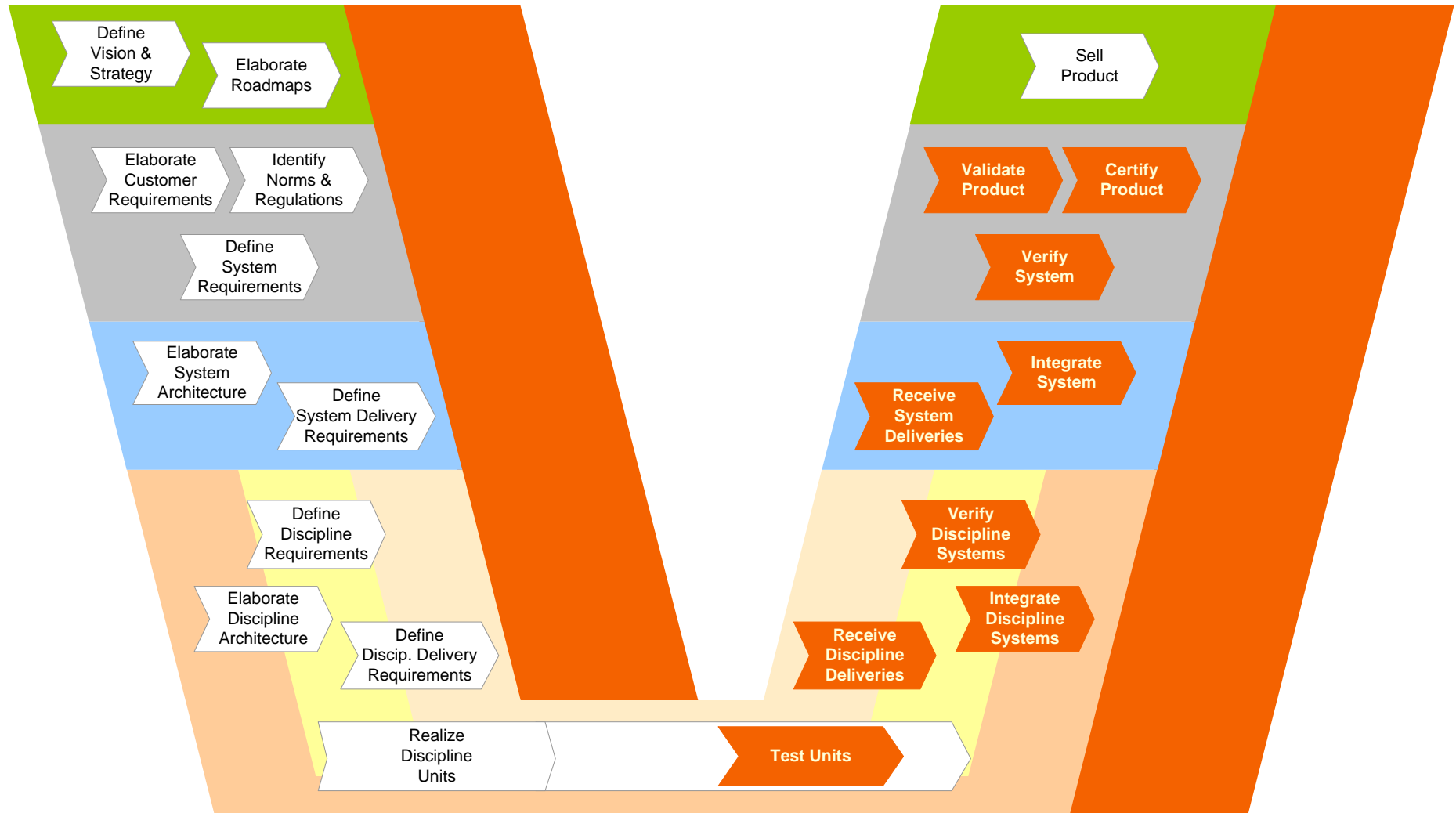


# System Development Process Model

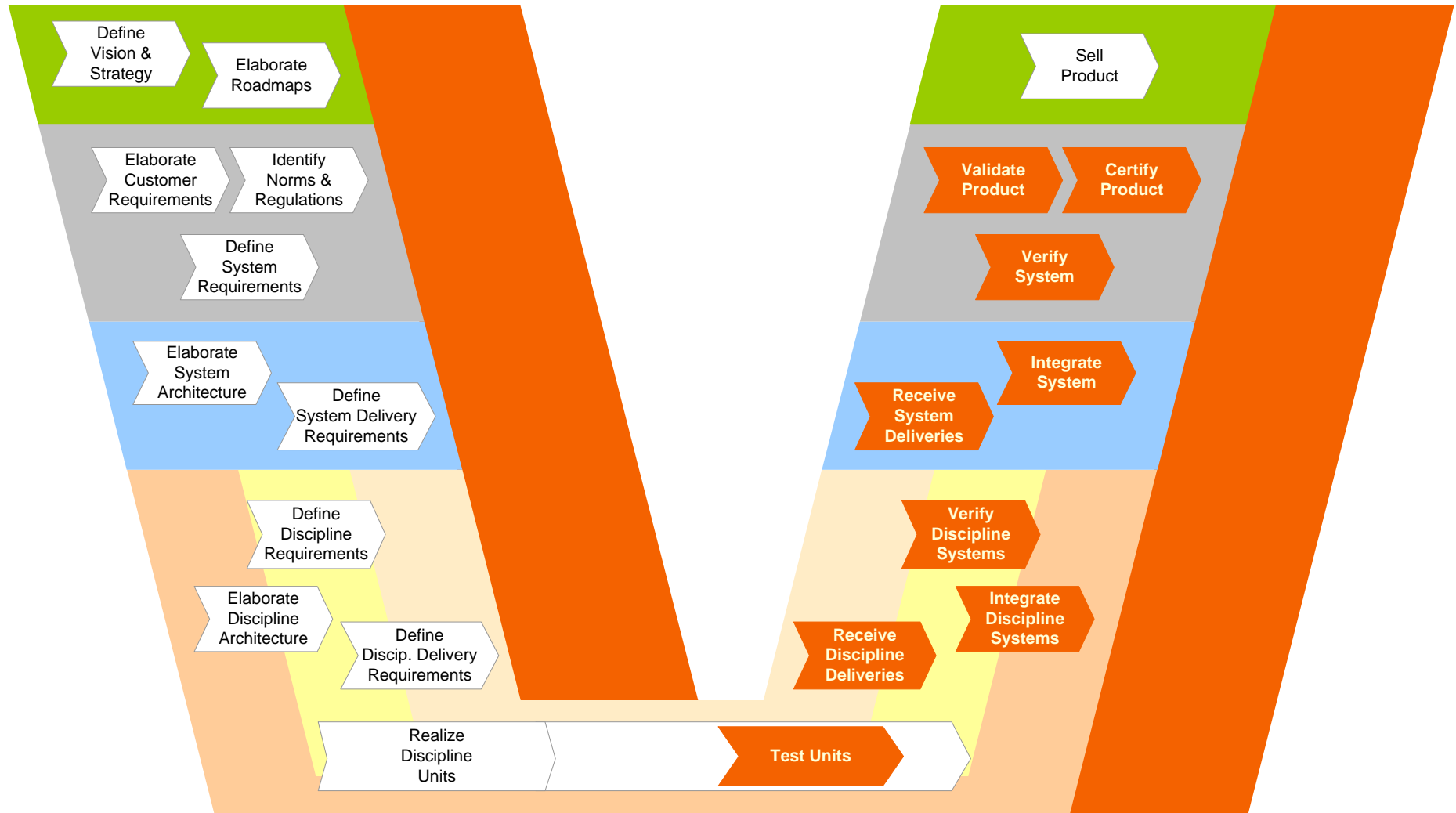
## basic V-Model



# Based on the V-Model develop a Test-W-Model

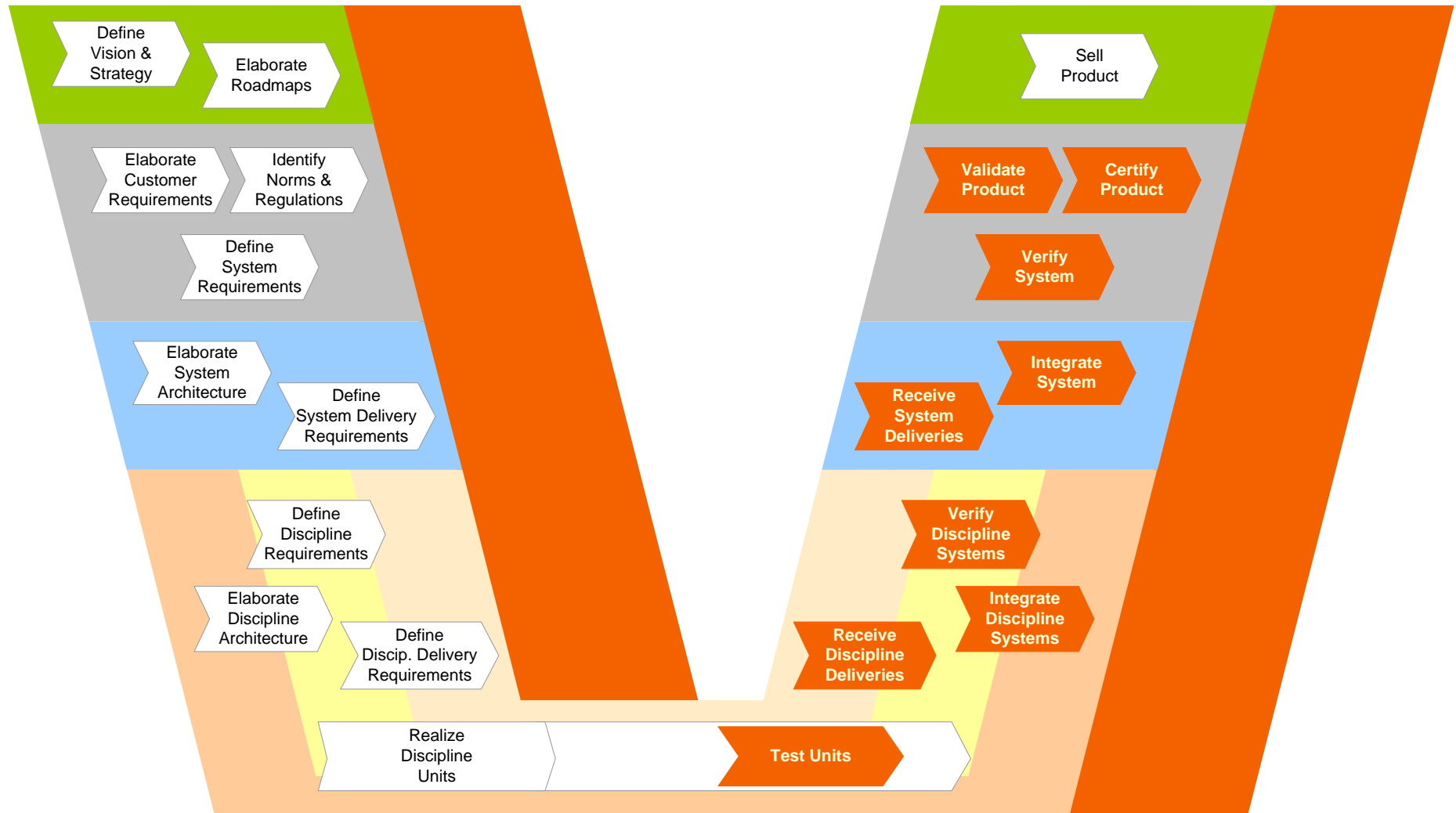


# Based on the V-Model develop a Test-W-Model

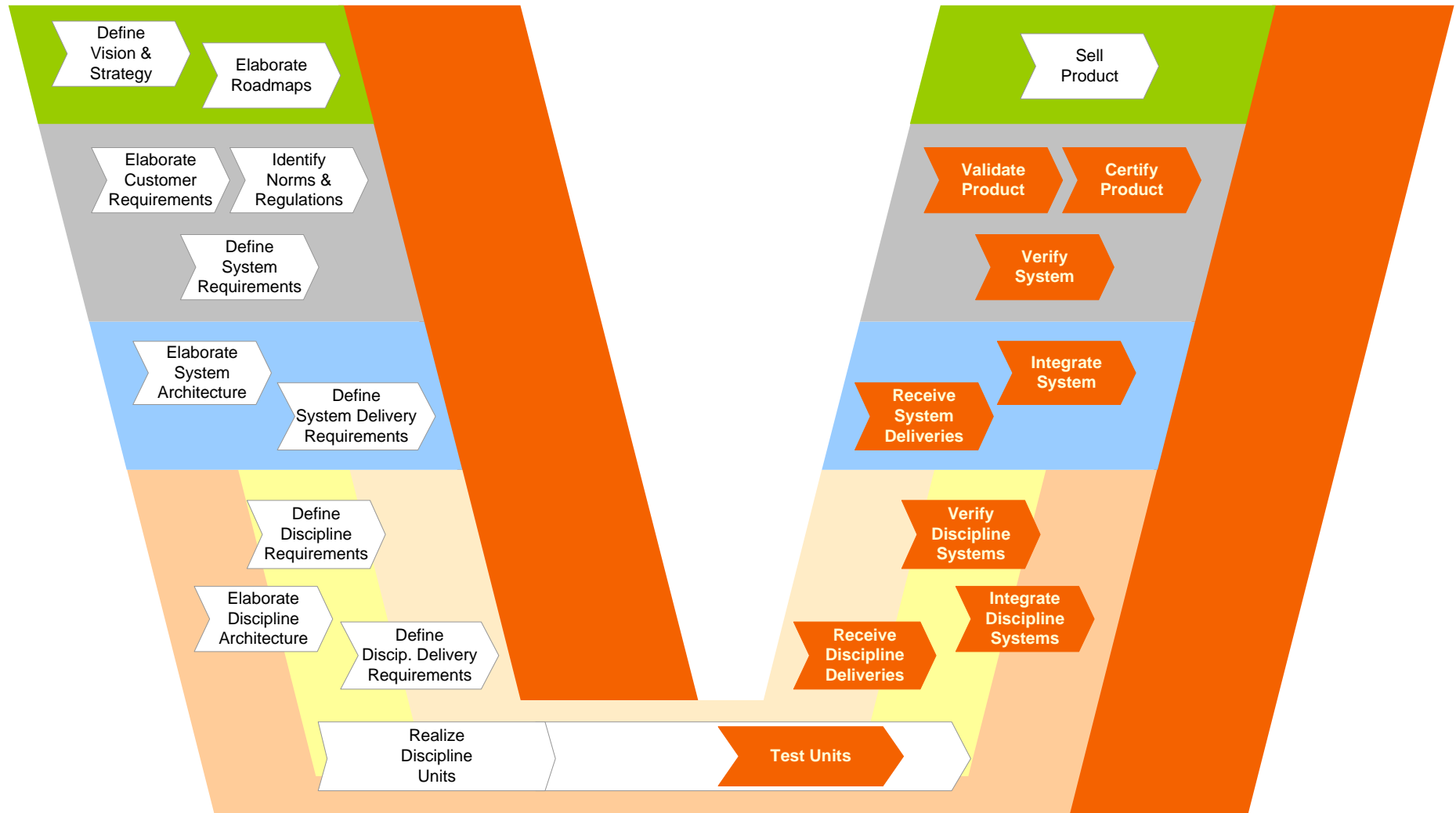




# Based on the V-Model develop a Test-W-Model



# Based on the V-Model develop a Test-W-Model



# Uncertainty Sources

resource  
(capacity) shift

offshoring  
issues

conflicting or  
over-determined  
information

patent  
litigation

regulatory  
changes

internal  
resistance

distributed  
development

missing acceptance  
(of innovative products  
and solutions)

not invented here  
syndrome

disruptive / destructive  
innovations

incomplete / old  
documentation  
(of used parts or  
components)

supplies  
in development

changing  
market conditions

false or contradicting  
requirements

unstable  
interfaces

unclear  
scope

outsourcing  
issues

financial /  
political crisis

changing  
environment

errata in  
supplier parts

communication  
issues

changing / evolving  
standards

missing / unknown  
information

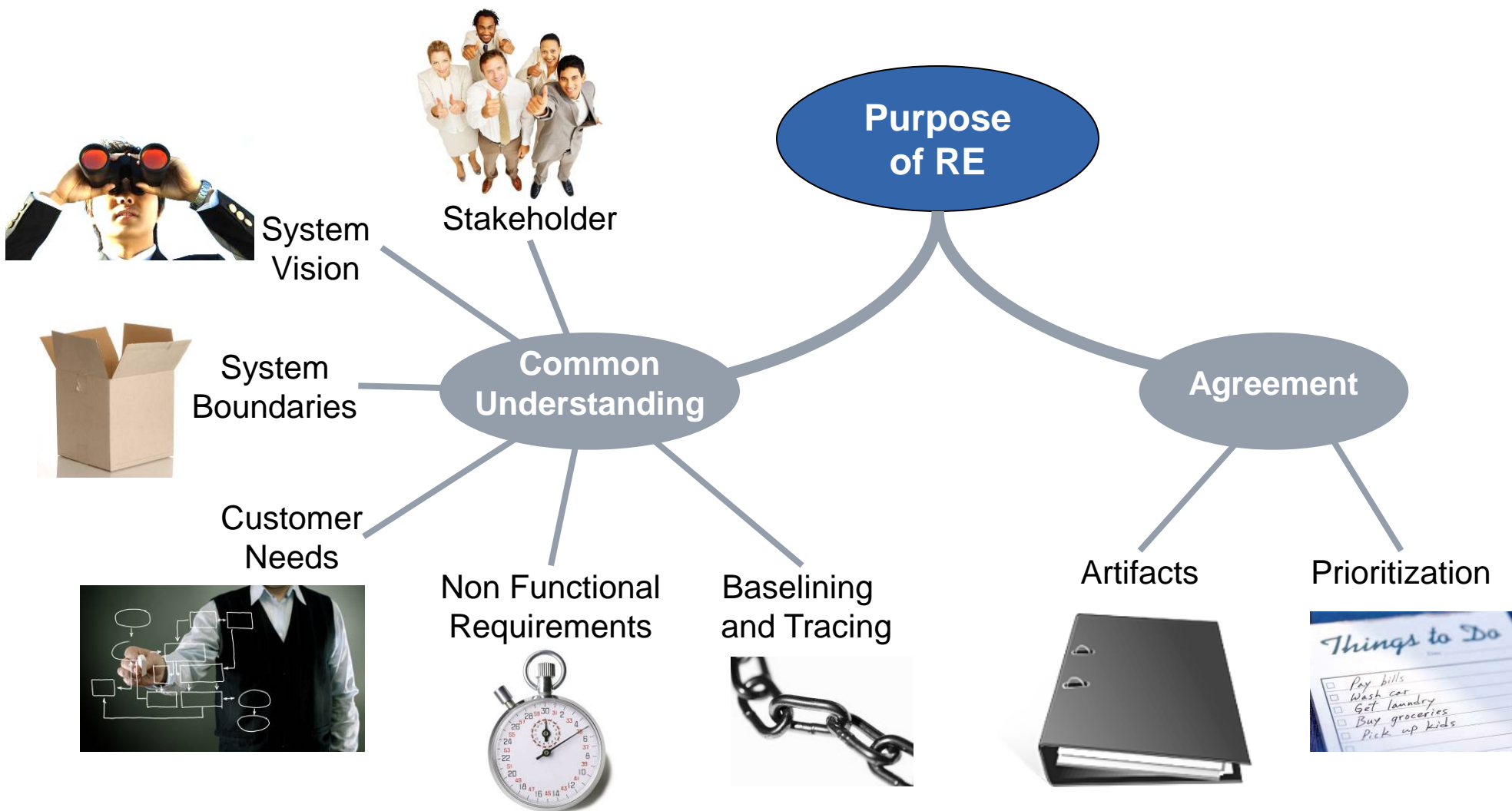
unknown or  
not involved  
Éminence grise

changing  
requirements or  
knowledge

missing / non-identified  
requirements

incorrect  
assumptions

# What do you need to know to build a system?

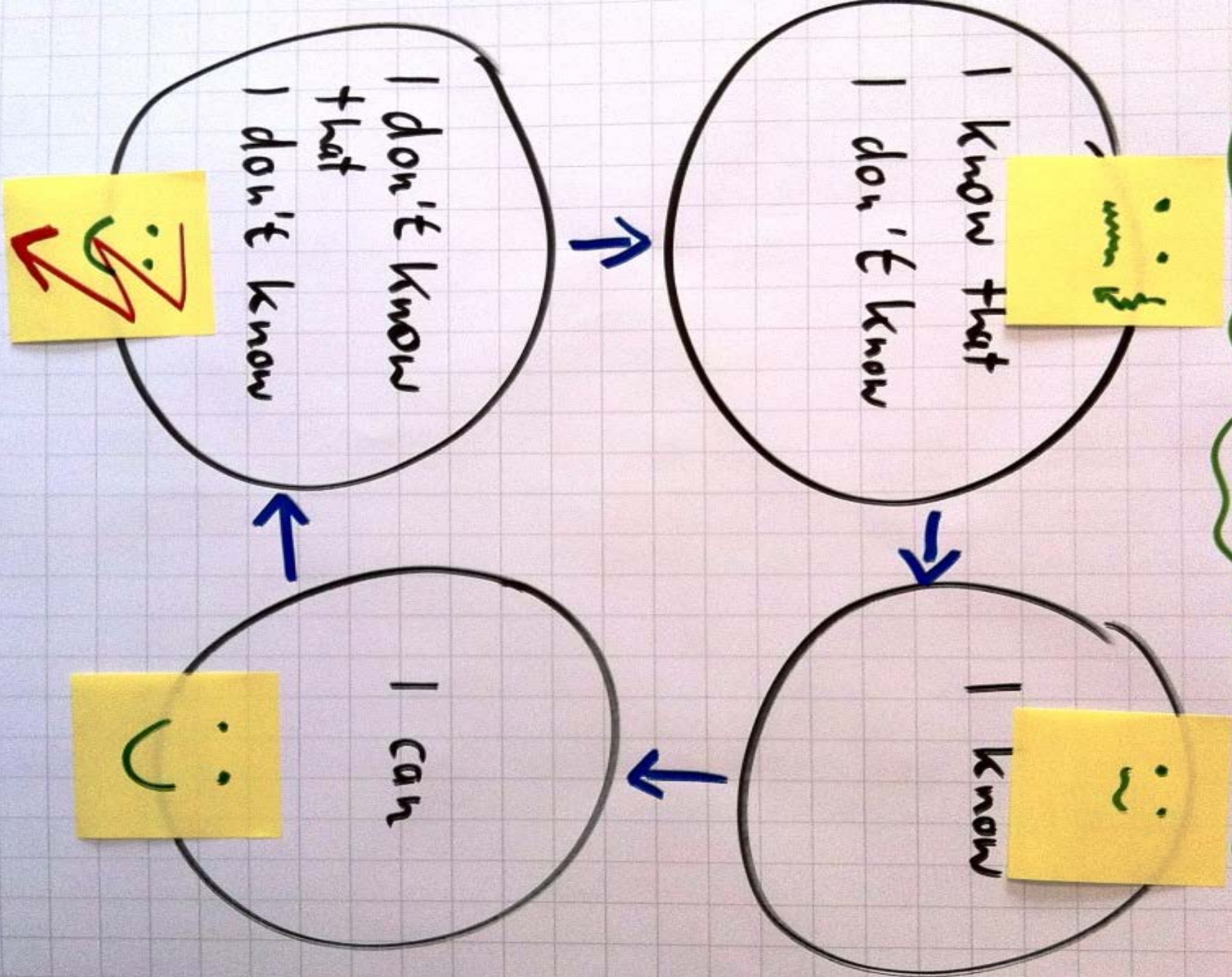


# Mapping of IEEE 29148 characteristics to Requirement Pyramid

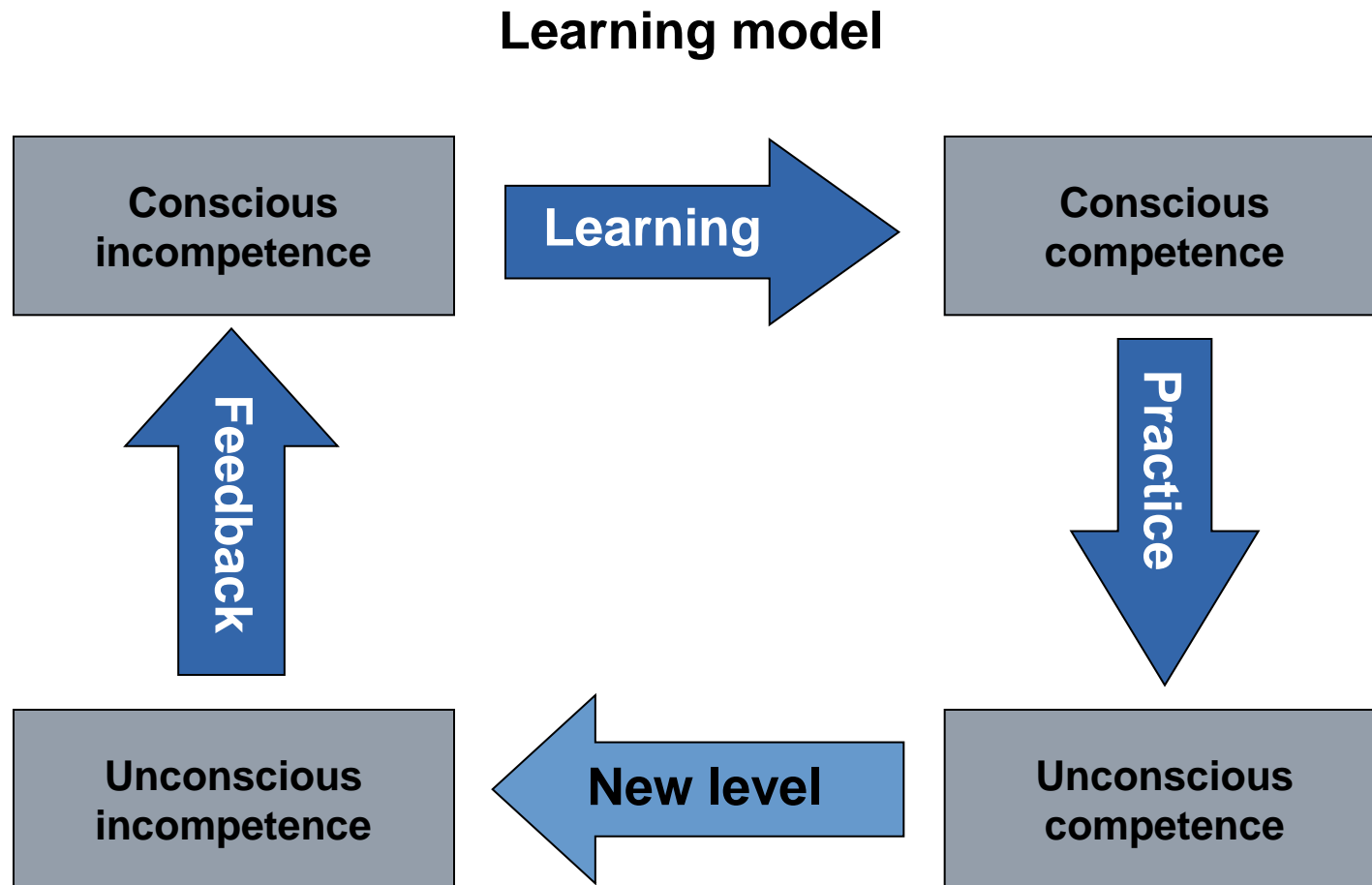
Quality Criteria	Business Goals	Customer Requirements	System Requirements
Individual Requirements			
Necessary	Mandatory	Mandatory	Mandatory
Implementation free	Mandatory	Mandatory	Mandatory
Unambiguous	Optional	Mandatory	Mandatory
Consistent	Mandatory	Mandatory	Mandatory
Complete	Optional	Optional	Mandatory
Singular	Mandatory	Optional	Mandatory
Feasible	Optional	Optional	Mandatory
Verifiable	Optional	Mandatory	Mandatory
Traceable	Mandatory	Mandatory	Mandatory
Set of Requirements			
Complete	Optional	Optional	Mandatory
Consistent	Mandatory	Mandatory	Mandatory
Affordable	Optional	Optional	Mandatory
Bounded	Optional	Optional	Mandatory



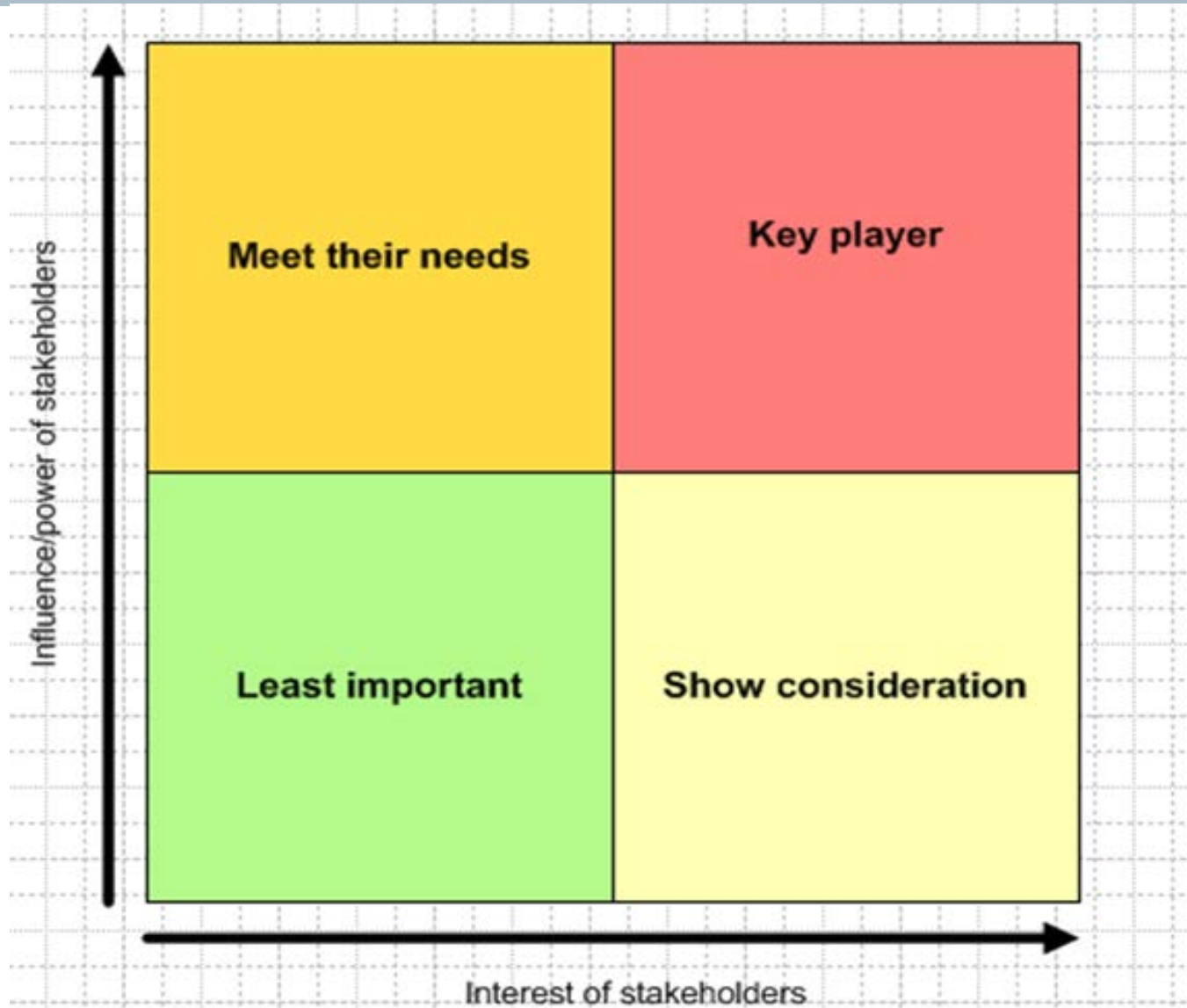
# Learning Model



# From good to great - through reflection and practice

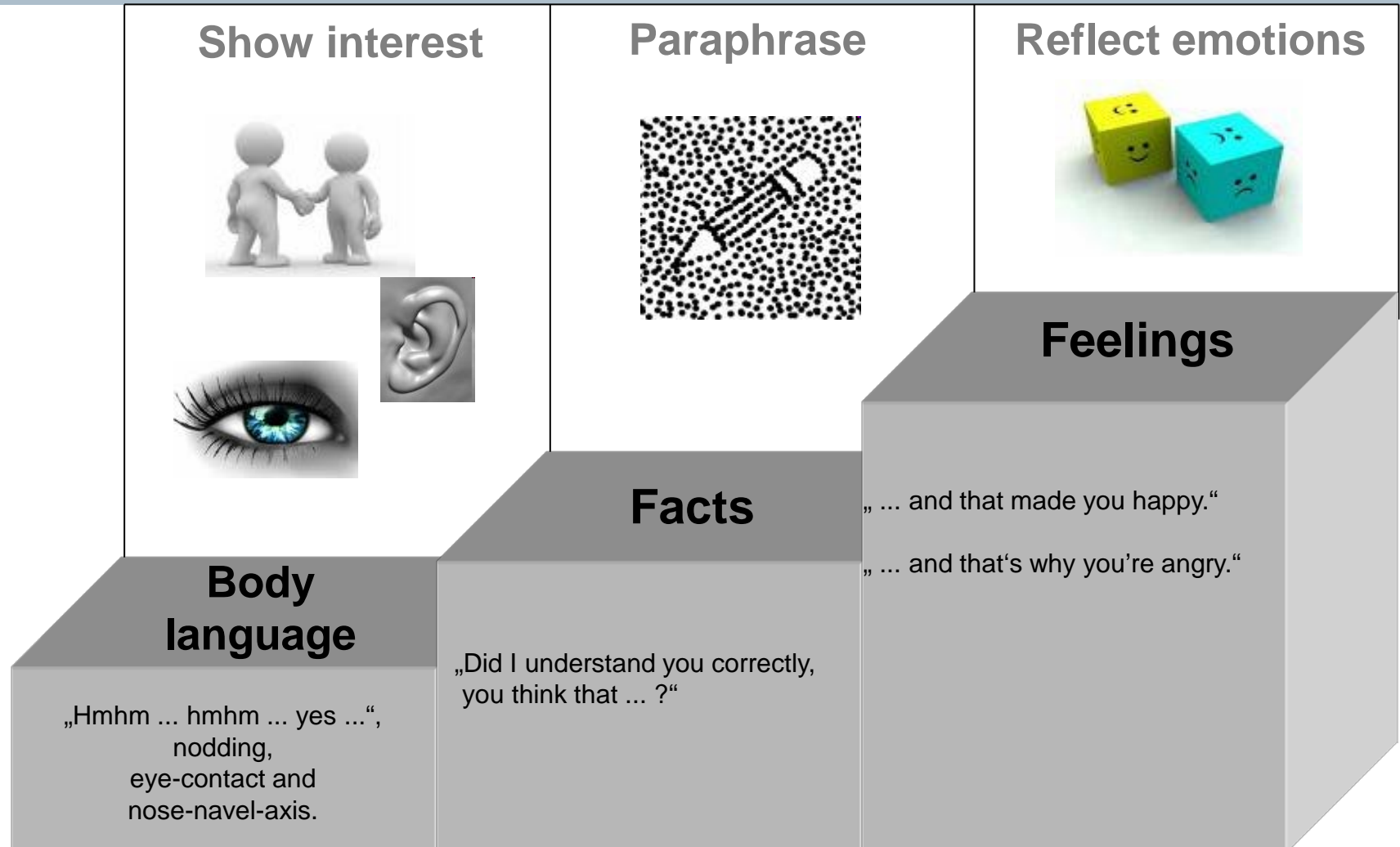


**If you want success – know your stakeholders!**





# Active Listening – first understand than be understood

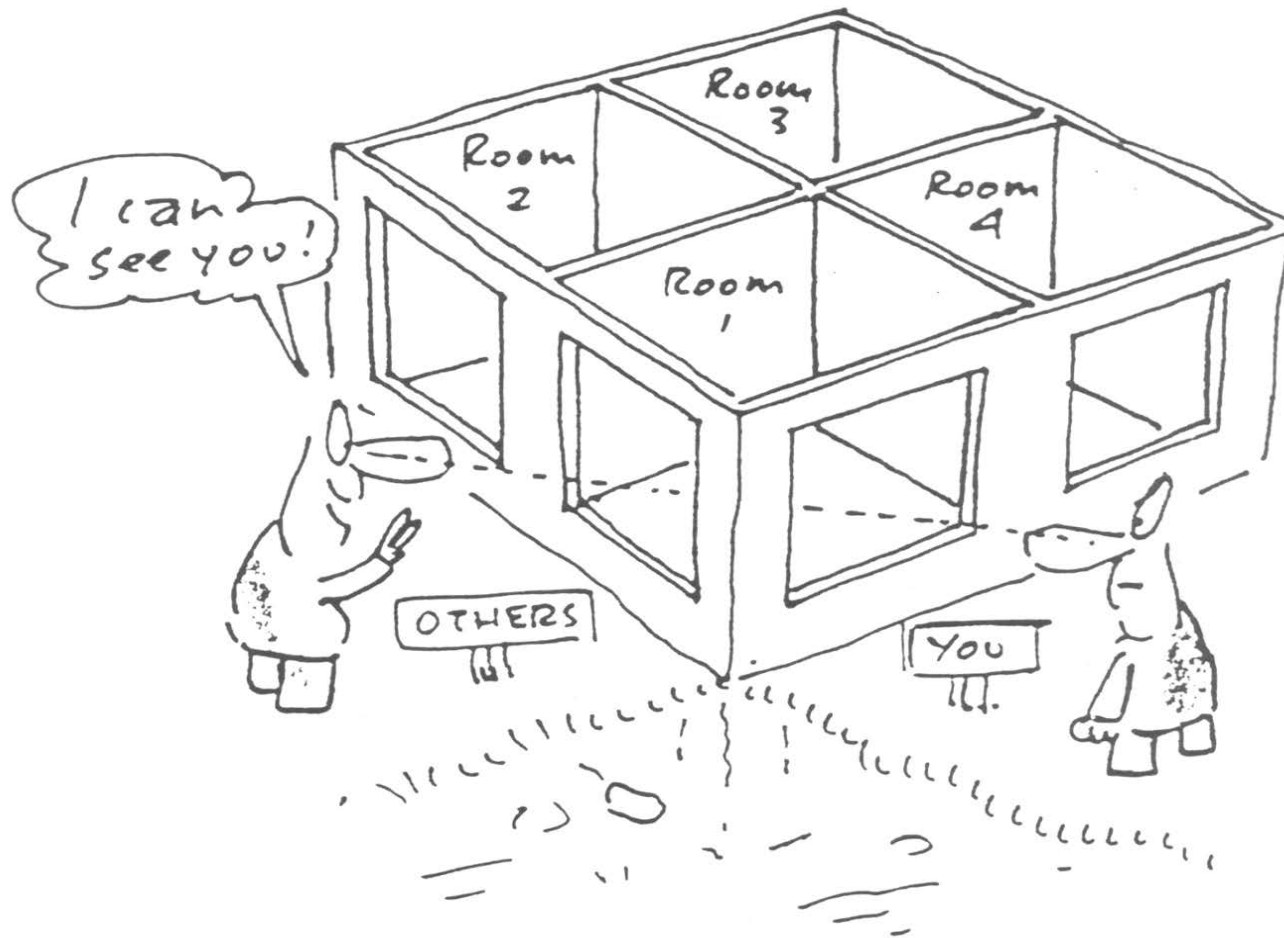


Are your interested?  
Show that your are listening.

You repeat the core statements in own  
words and ask for affirmation.

You say how you think your dialogue partner  
feels. Than see if he/she agrees.

# The Johari window is a useful model for communication and cooperation



## Providing WWW Feedback is a chance to grow

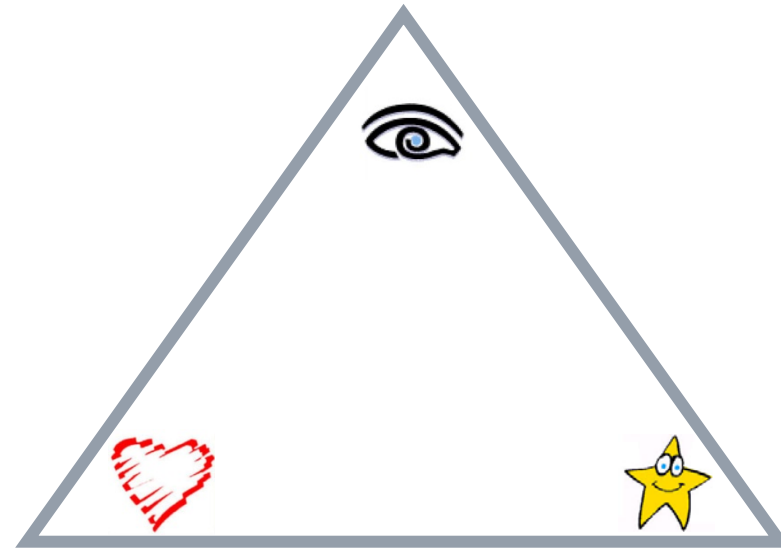
**W** hat did I see?  
**W** hat did I feel?  
**W** hat do I wish?

*In German:*

**W**ahrnehmung  
**W**irkung  
**W**unsch

### Perception (objective)

Describe a concrete  
perception of a behavior



### Effect (subjective)

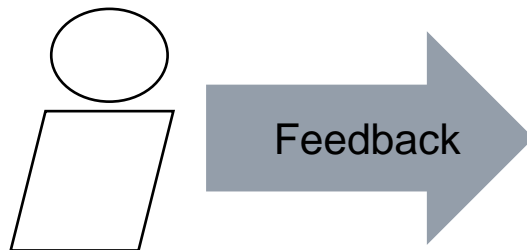
Describe the effect of  
the observed behavior

### Wish (acceptable)


Formulate a wish and  
give opportunity to clarify

# Providing WWW Feedback is a chance to grow

## When you **give** feedback ...



Perception  What did you see and hear?

Effect  What effect did it have on you? How did you feel?

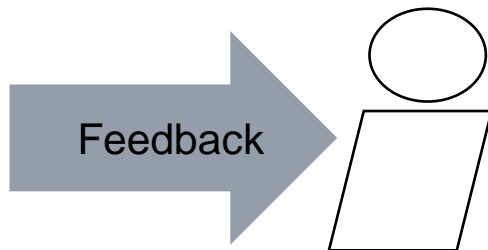
Wish  What kind of behavior would you like to see?

### ... don't forget:


- ✓ Use "I" instead of "one" or other general terms
- ✓ Speak directly to him / her (not about him / her)
- ✓ One positive comment is often more effective than ten negative remarks


# Providing WWW Feedback is a chance to grow

## When **receiving** feedback ...



**Perception**  If not given – ask for a specific situation (example)?

**Effect**  Try to understand how the feedback partner felt?

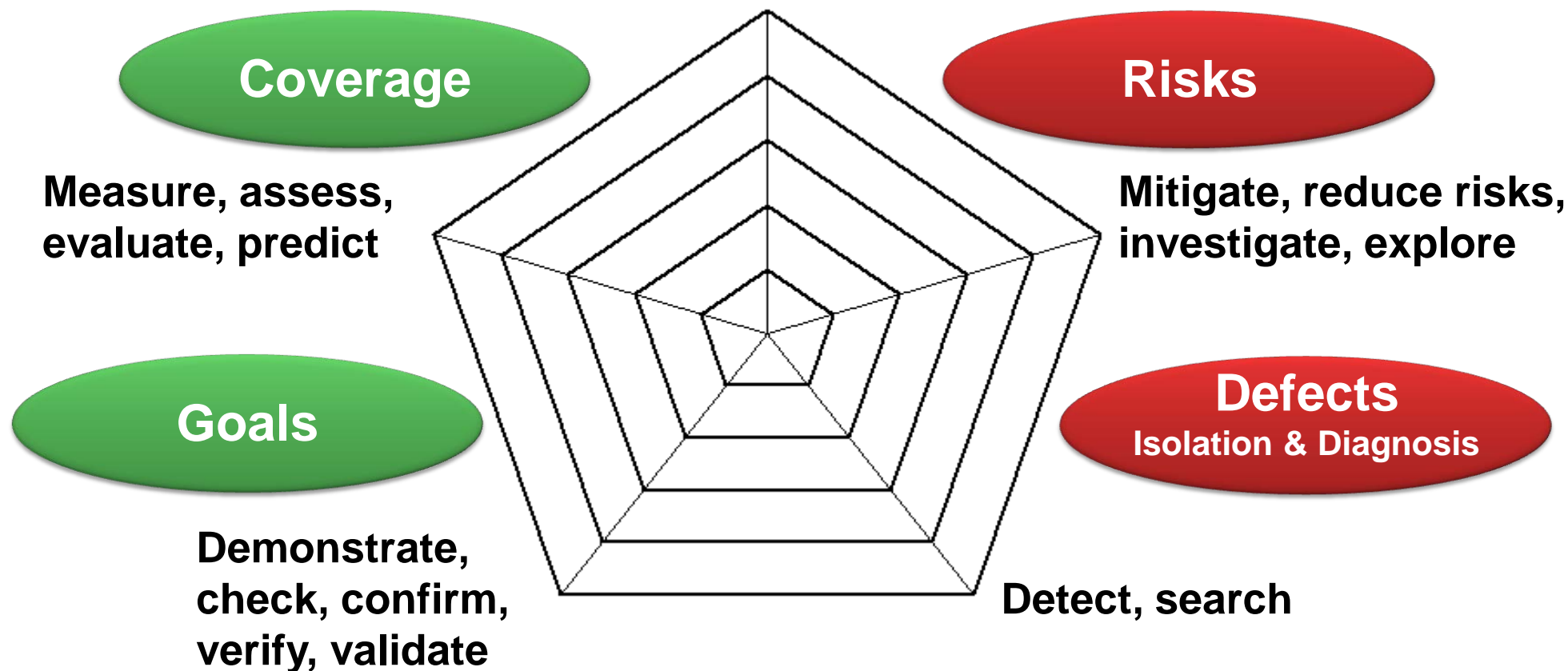
**Wish**  What kind of behavior would he/she like to see?

### ... don't forget:

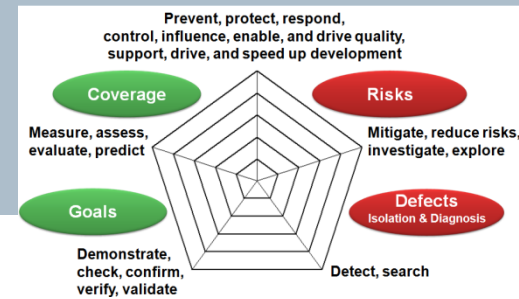
- ✓ See receiving feedback as an opportunity
- ✓ Listen carefully
- ✓ If you are not able to listen now – say when
- ✓ Never defend, explain or justify !!!

## Why do we test? Dimensions of testing

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

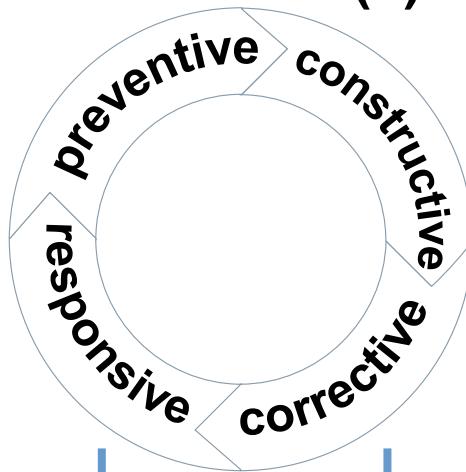


Reference: Peter Zimmerer, "[The Value of Testing in 5 Dimensions](#)", *Testing Experience*, September 2010



## Why do we test?

### Historical and future (?) view



*History consists of a series of accumulated imaginative inventions.*  
Voltaire, 1694–1778

Show it works

Find defects

Assess quality  
Evaluation  
Measure quality

Influence quality  
Control quality

Investigate  
Explore  
Learn

Speed up  
Protect  
Respond

**Demonstration**

**Detection**

**Prediction**

**Prevention**

**Exploration**

**Productivity**

1950s view

1975s view

1980s view

1990s view

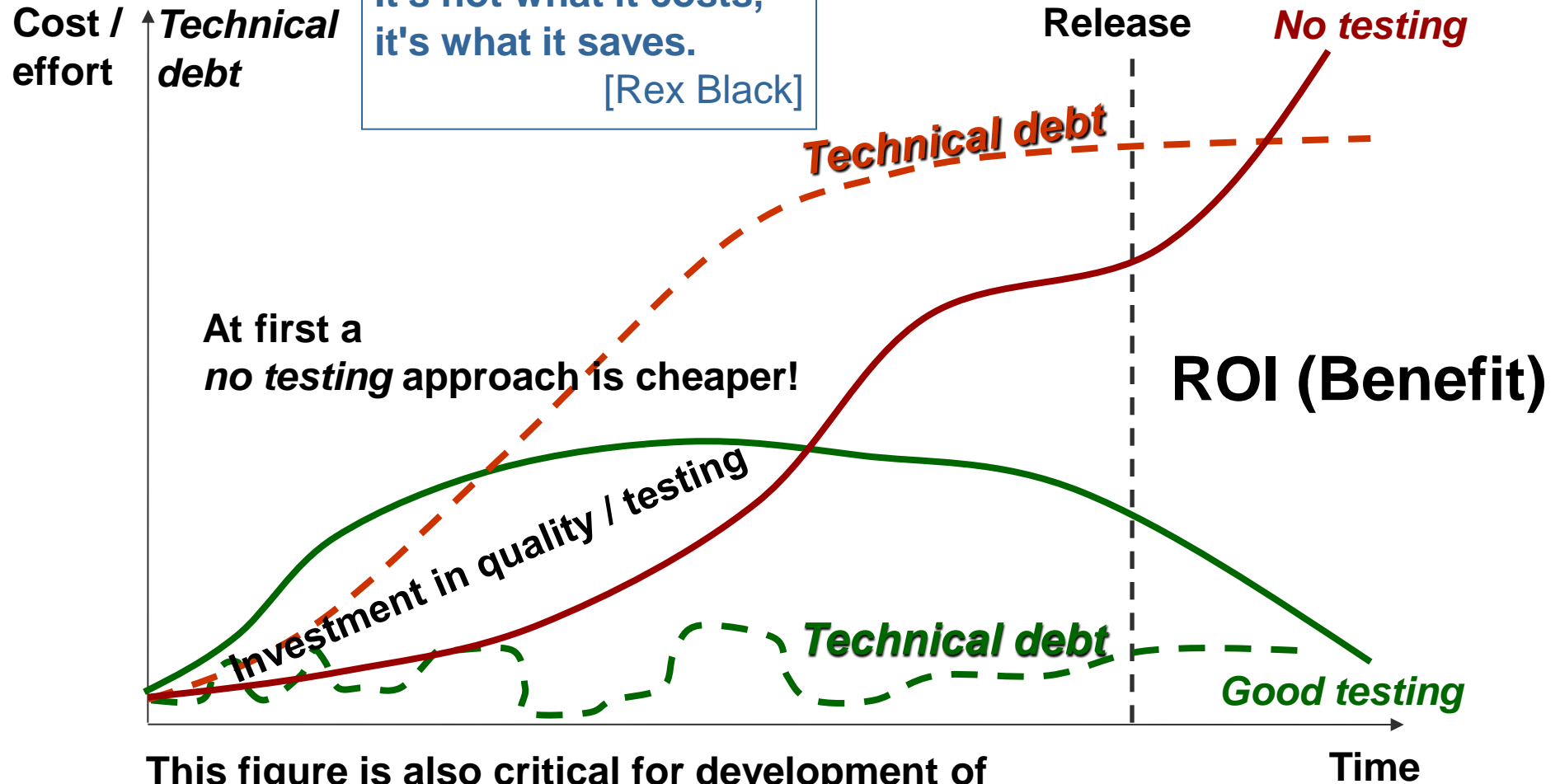
2000s view

2010s view



## Overall cost & effort – Technical debt

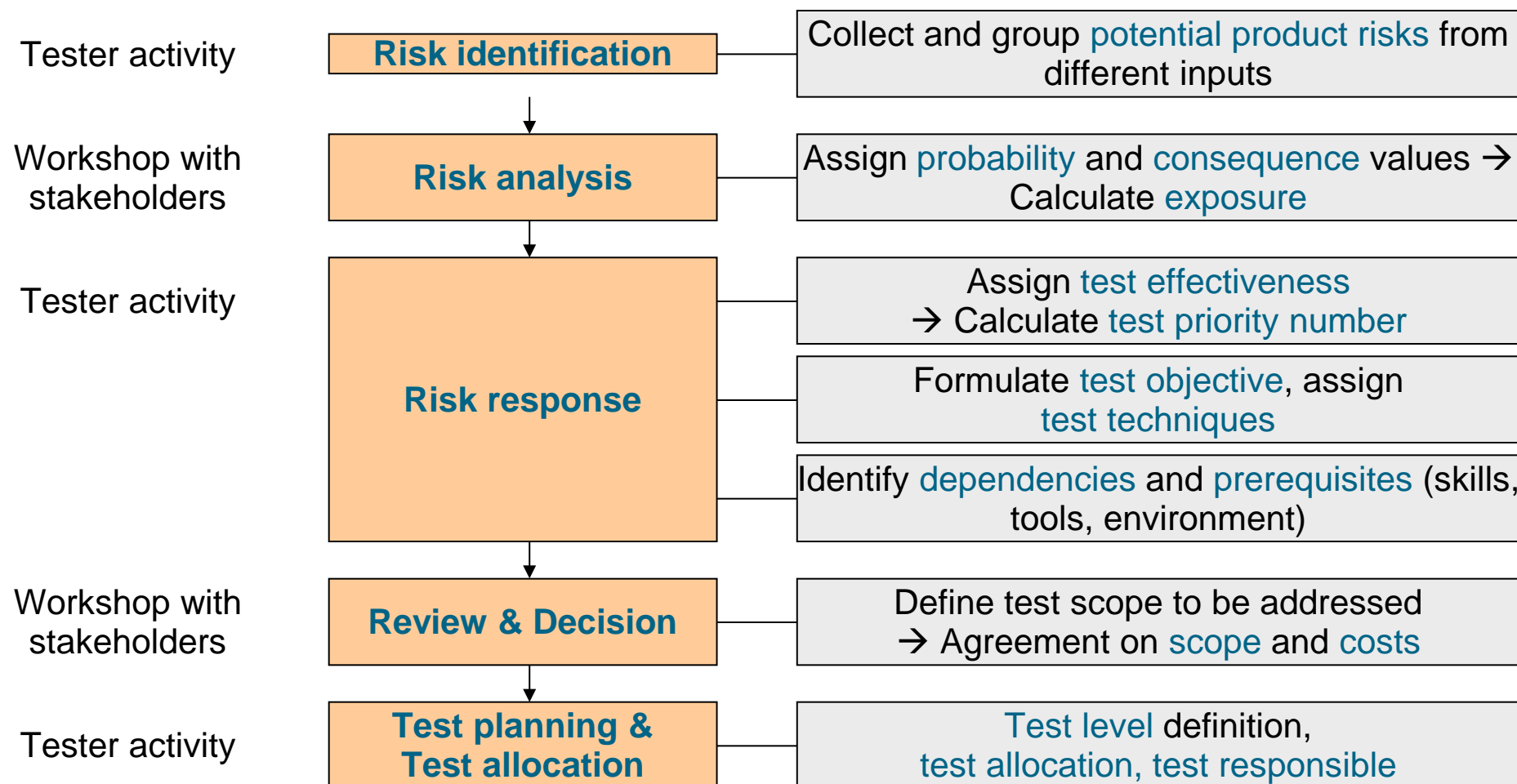
It's not what it costs,  
it's what it saves.  
[Rex Black]



This figure is also critical for development of frameworks, platforms, product lines, ecosystems!



## Risk-based test strategy – Steps

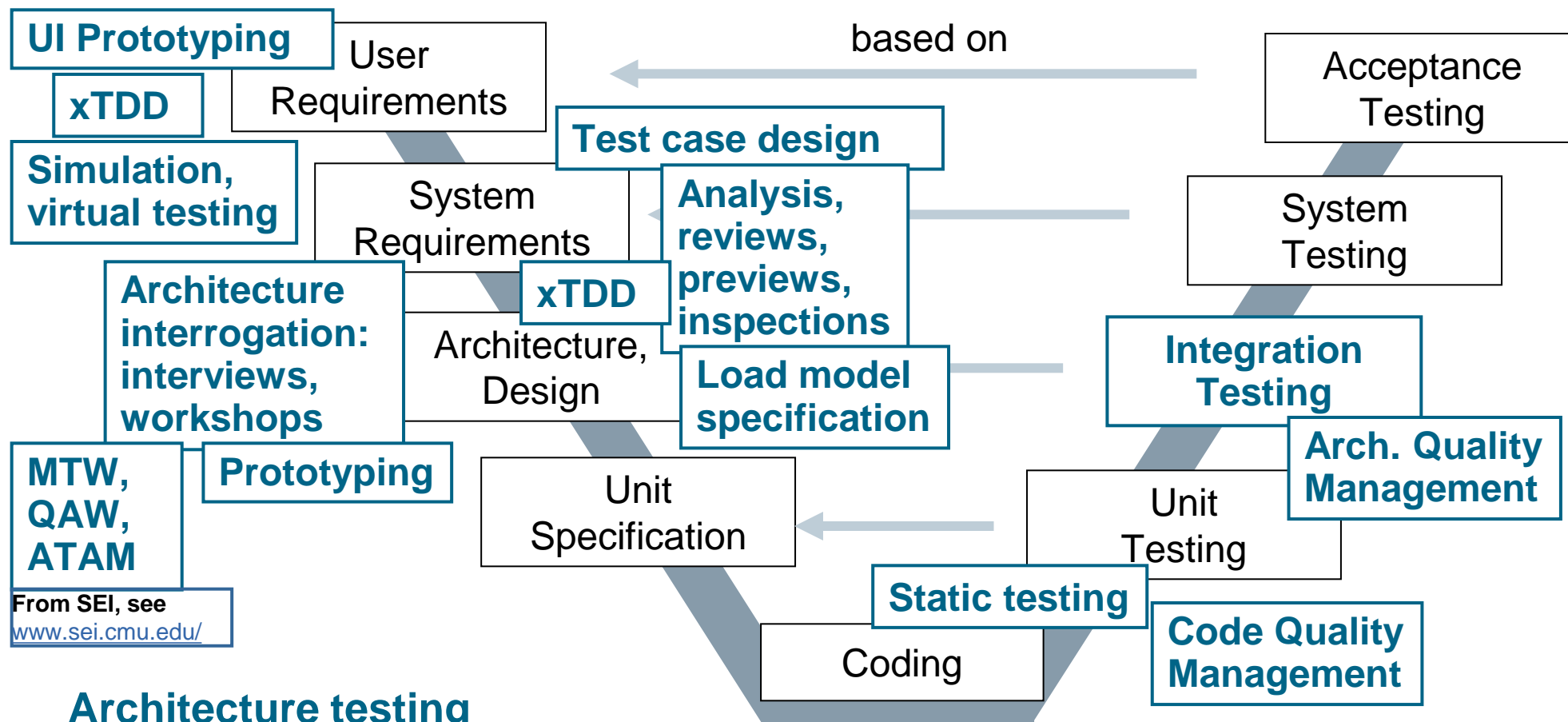


Adapted from: Paul Gerrard, Neil Thompson: Risk-Based E-Business Testing, 2002

## Risk-based test strategy – On one page ...

Testing Strategy					Testing Strategy							
Risk	Sub-system	P 1...5	D 1...5	RE 1...25	TE 1...5	TPN 1...125	Test Level who?	Test Technique	Measurement Test Exit Criteria	Depend.	Effort h/m/h	Time scale
F-Req. 1												
NFR 1												
Quality Criterion 1												
Claim 1												
Use Case 1												
Feature 1												
Arch. Decision 1												
Design Decision 1												
Technology Decision 1												
3rd party comp. selection												
Bug Category 1												
Risk 1												

# Test levels – Example V model with architecture testing



## Architecture testing

is any testing of architecture and architectural artifacts

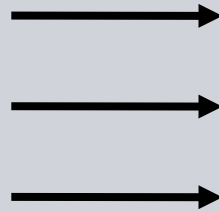
→ mapping of architectural risks to test levels

## Example: Integration Testing



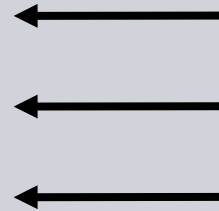
### Control system

- Set operation mode
- Control robot actions
- Visualize status



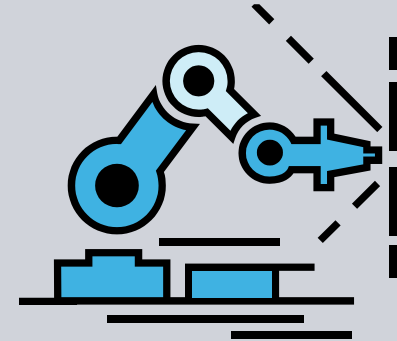
### Requests

- Init/Shutdown
- Move to position
- Stop moving
- Get operation status



### Replies

- ACK/NACK
- Return state



### Robot system

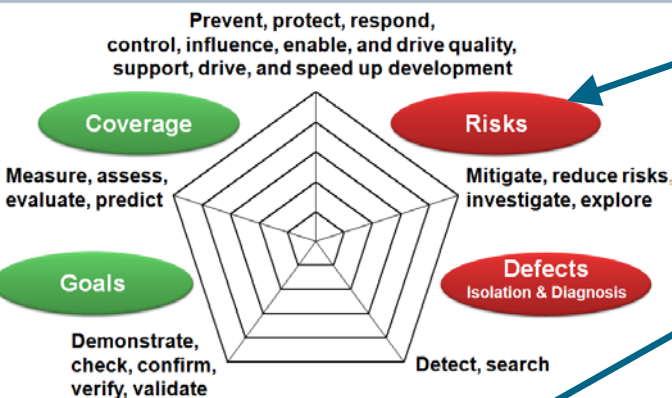
- Operational states
- Error states
- Real time constraints
- Safety constraints

- What aspects of this system would you test on **Integration Test** level?
- What kind of input would you need for these tests?
- Which stakeholder(s) can provide these inputs?
- What do you – as **Test Engineer / Test Architect** – have to provide?



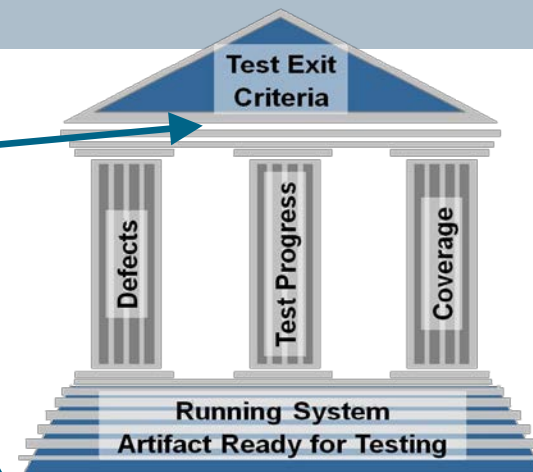
# Summary: Building blocks of a Risk-based Test Strategy (RBT)

## Why do we test? Dimensions of testing

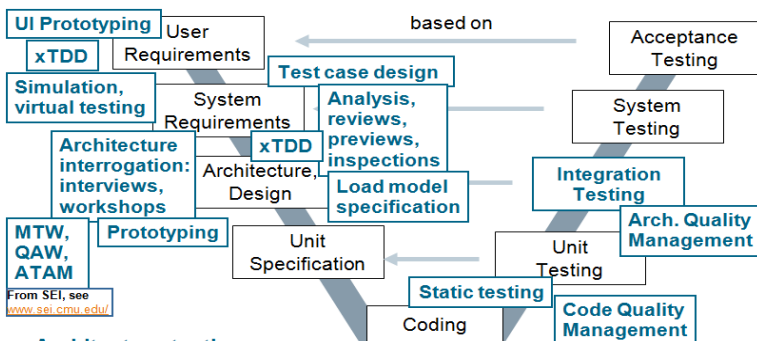


Testing Strategy

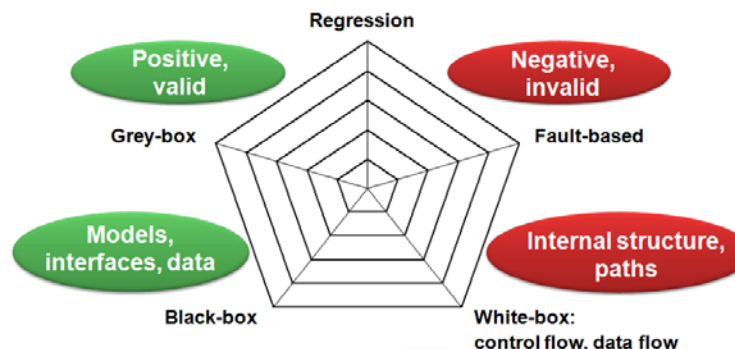
Risk	Sub-system	P	D	RE	TE	TPN	Test Level	Test Technique	Measurements	Test Exit Criteria
Req. NFR 1		1..5	1..5	1.25	1..5	1..125				
Quality Criteria										
Claim 1										
Use Case 1										
Feature 1										
Arch Decision 1										
Design Decision 1										
Technology Decision 1										
and partly completed										
Bug Category 1										
Risk 1										



## Test levels – Example V model with architecture testing



## Categories of test design techniques



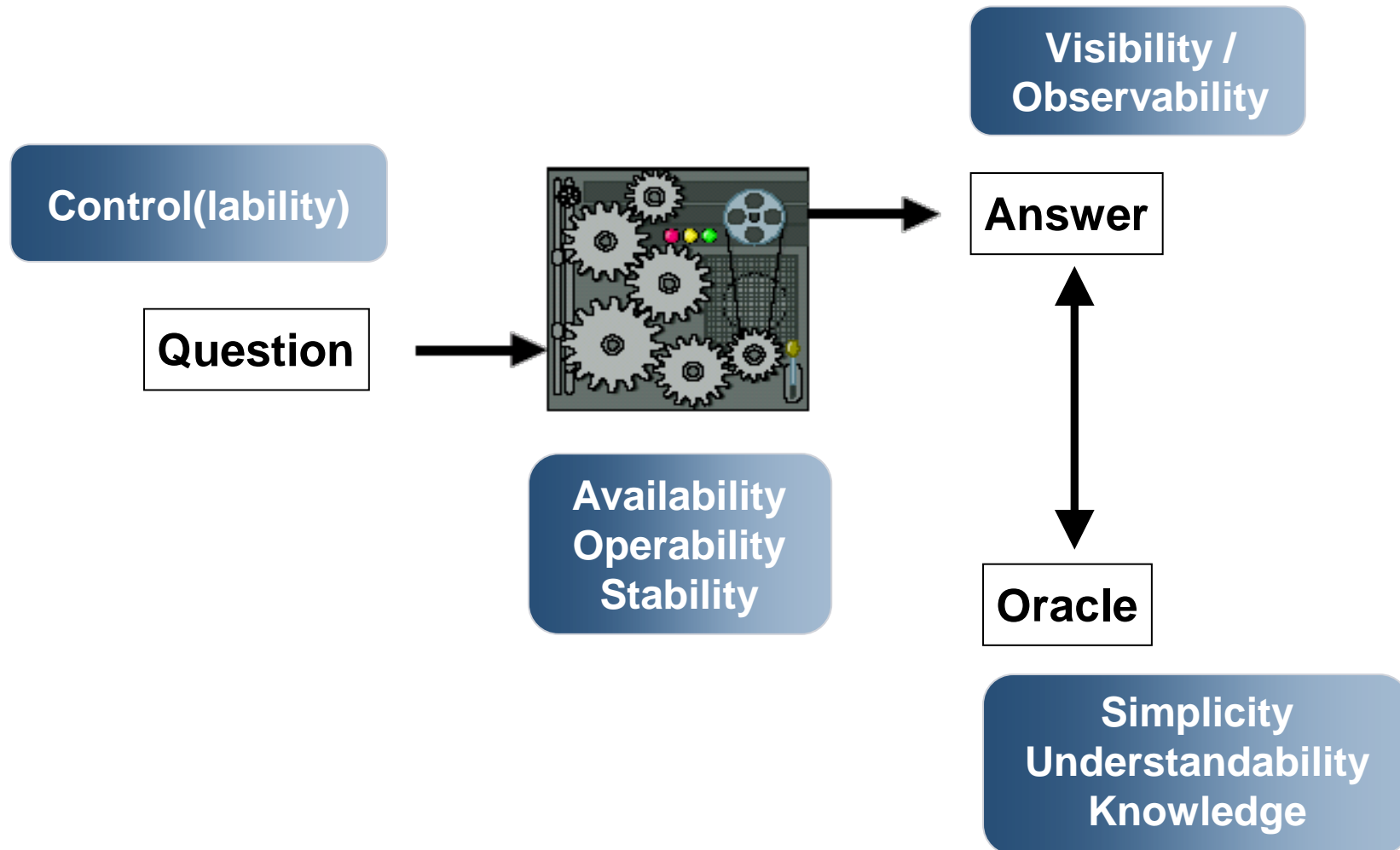
Test Design Techniques on One Page

Category	Technique	Page
Black-box	Standards (e.g. ISO/IEC 9126-2:2006, IEC 61508) norms, formal specifications, claims	1
Black-box	Requirements (aligned with traceability matrix requirements & test cases)	2
Black-box	User (user-based testing: sequence diagrams, activity diagrams)	3
Black-box	Model (state-based testing: state machines, state space analysis)	4
Black-box	Flow testing, scenario testing, soap, open testing	5
Black-box	User (operational profiles, frequency and priority / usability / software reliability engineering)	6
Black-box	Functional testing (markov chains)	7
Black-box	Random (random testing)	8
Black-box	Pathways, functions, spots, user stories, processes, services, interfaces	9
Black-box	(design) by contract (built-in self test)	10
Black-box	(discreet case partitioning)	11
Black-box	Domain partitioning, category partition method	12
Black-box	Classification tree method	13
Black-box	Boundary, value analysis	14
Black-box	General values	15
Black-box	Test catalog / matrix for input values, input fields	16
Black-box	State-based testing (Finite State Machines)	17
Black-box	Cause-effect graphing	18
Black-box	Decision tables, decision trees	19
Black-box	Combinatorial testing (orthogonal / covering arrays, pair-wise, n-wise)	20
Black-box	Time (cycle frequency, accuracy, timing, test data)	21
Black-box	Evolutionary testing	22
Black-box	Metamorphic testing	23
Grey-box	Dependencies / Relations between classes, objects, methods, functions	24
Grey-box	Dependencies / Relations between components, services, applications, systems	25
Grey-box	Communication (behavior dependency analysis)	26
White-box	Coverage (statements (S), nodes)	27
White-box	Branches (B), conditions, decisions (C), C/C++	28
White-box	Static metrics	29
White-box	Control flow-based	30
White-box	Data flow-based	31
White-box	Normal, expected behavior	32
White-box	Invalid, unexpected behavior	33
White-box	Error handling	34
White-box	Exceptions	35
Fault-based	Risk based	36
Fault-based	Impact analysis (Failure Mode and Effect Analysis, Fault Tree Analysis)	37
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	38
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	39
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	40
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	41
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	42
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	43
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	44
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	45
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	46
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	47
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	48
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	49
Fault-based	Defect patterns (e.g. by James A. Whittaker, Jon Hager)	50
Regression (selective retesting)	Retest all	51
Regression (selective retesting)	Retest by risk, priority, severity, criticality	52
Regression (selective retesting)	Retest by profile, frequency of change, parts which are often used	53
Regression (selective retesting)	Retest changed parts	54
Regression (selective retesting)	Retest parts that are influenced by the changes (impact analysis, dependency analysis)	55

Key  
 Configuration  
 Metrics, Parameters, Techniques, Types, and Values to Create a Test Case  
 Effort / Difficulty / Repeating Test Intensity (3 Levels)

SIEMENS

## Testability – Factors (3)



# Heuristics of Testability

## Understanding of product (status)

### Epistemic Testability

- Prior Knowledge of Quality
- Tolerance for Failure

*How narrow is the gap between what we know and what we need to know about the status of the product under test?*

## Understanding of project

### Project-Related Testability

- Change Control
- Information Availability
- Tool Availability
- Test Item Availability
- Sandboxing
- Environment Controllability
- Time

*Testability influenced by changing the conditions under which we test*

## Practical Testability

## Control & observe the product

### Subjective Testability

- Product Knowledge
- Technical Knowledge
- Domain Knowledge
- Testing Skill
- Engagement
- Helpers
- Test Strategy

*Testability influenced by changing the tester or the test process*

### Intrinsic Testability

- Observability
- Controllability
- Algorithmic Simplicity
- Unbugginess
- Smallness
- Decomposability
- Similarity (to known and trusted technology)

*Testability influenced by changing the product itself*

## Understanding of users & usage

### Value-Related Testability

- Oracle Availability
- Oracle Authority
- Oracle Reliability
- Oracle Precision
- Oracle Inexpensiveness
- User Stability & Unity
- User Familiarity
- User Availability
- User Data Availability
- User Environment Availability
- User Environment Stability & Unity

*Testability influenced by changing the quality standard or our knowledge of it*

Reference: James Bach

<http://www.satisfice.com/tools/testability.pdf>

## Design for Testability (DfT) – How?

Suitable test architecture, good design principles  
Choice of technologies, libraries, frameworks, services

Interaction with the system under test through  
well-defined **control** points and **observation** points

Additional (scriptable) interfaces, ports, hooks, mocks, interceptors for testing  
purposes (setup, configuration, simulation, recovery)

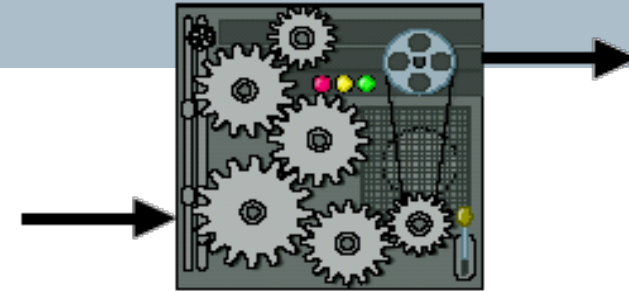
Coding guidelines, naming conventions  
Internal software quality (architecture, code)

Built-in self-test (BIST), built-in test (BIT)  
Consistency checks (assertions, design by contract, deviations)

Logging and tracing (AOP, counters, monitoring, probing, profiling)

Diagnosis and dump utilities, black box (internal states, resource utilization,  
anomalies at runtime, post-mortem failure analysis)

*Think test-first (xTDD): how can I test this?*

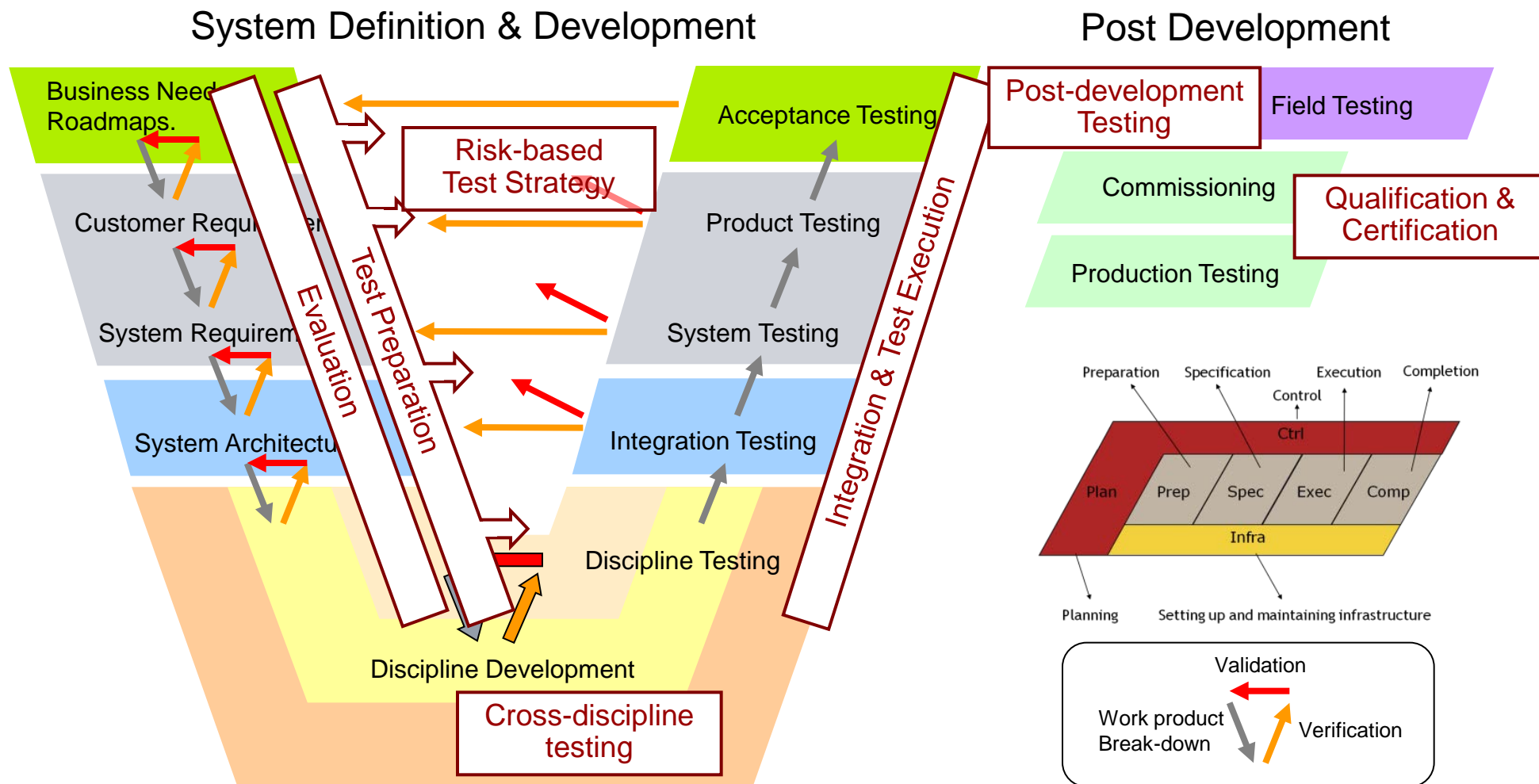


**Control(lability)**

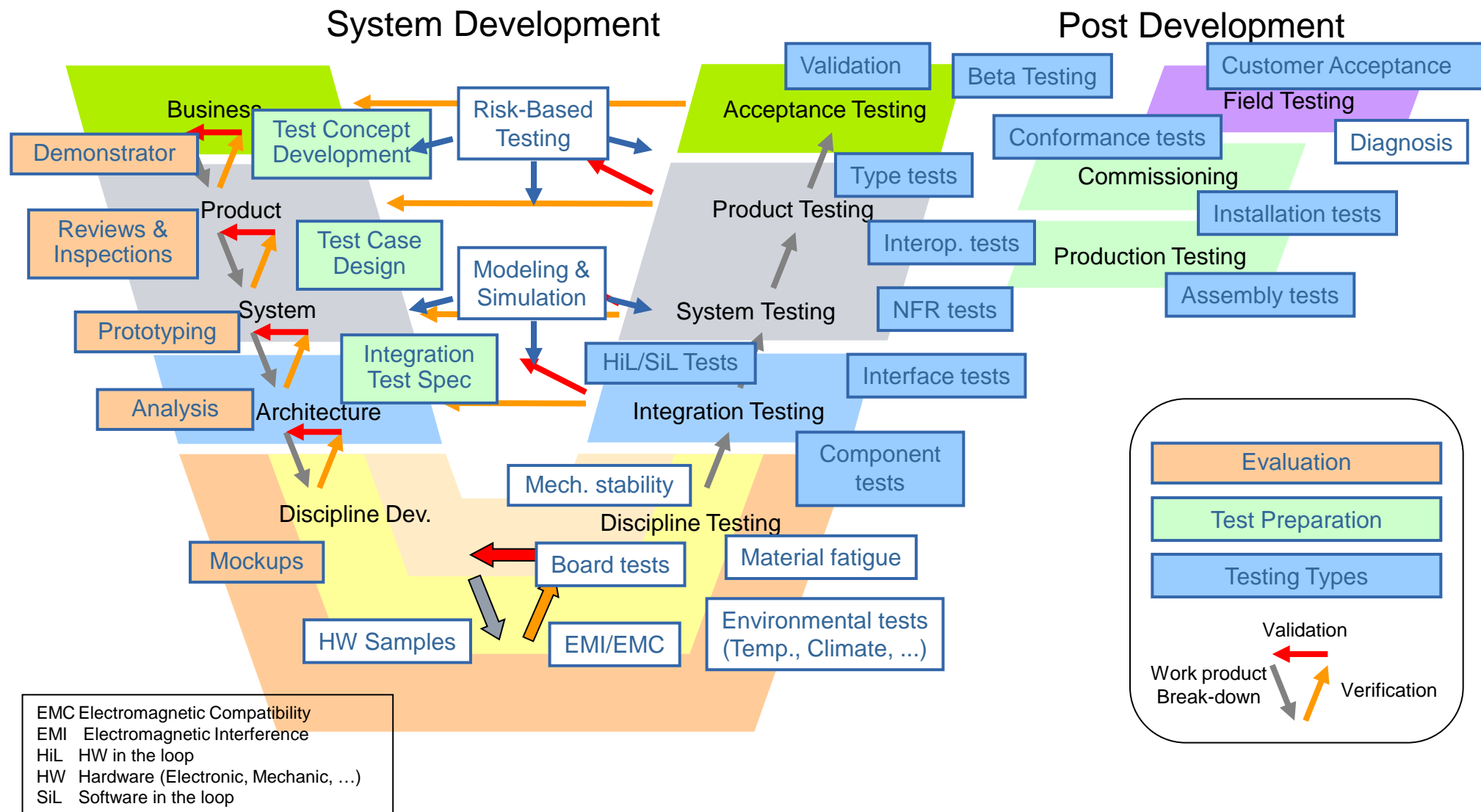
**Visibility / observability**



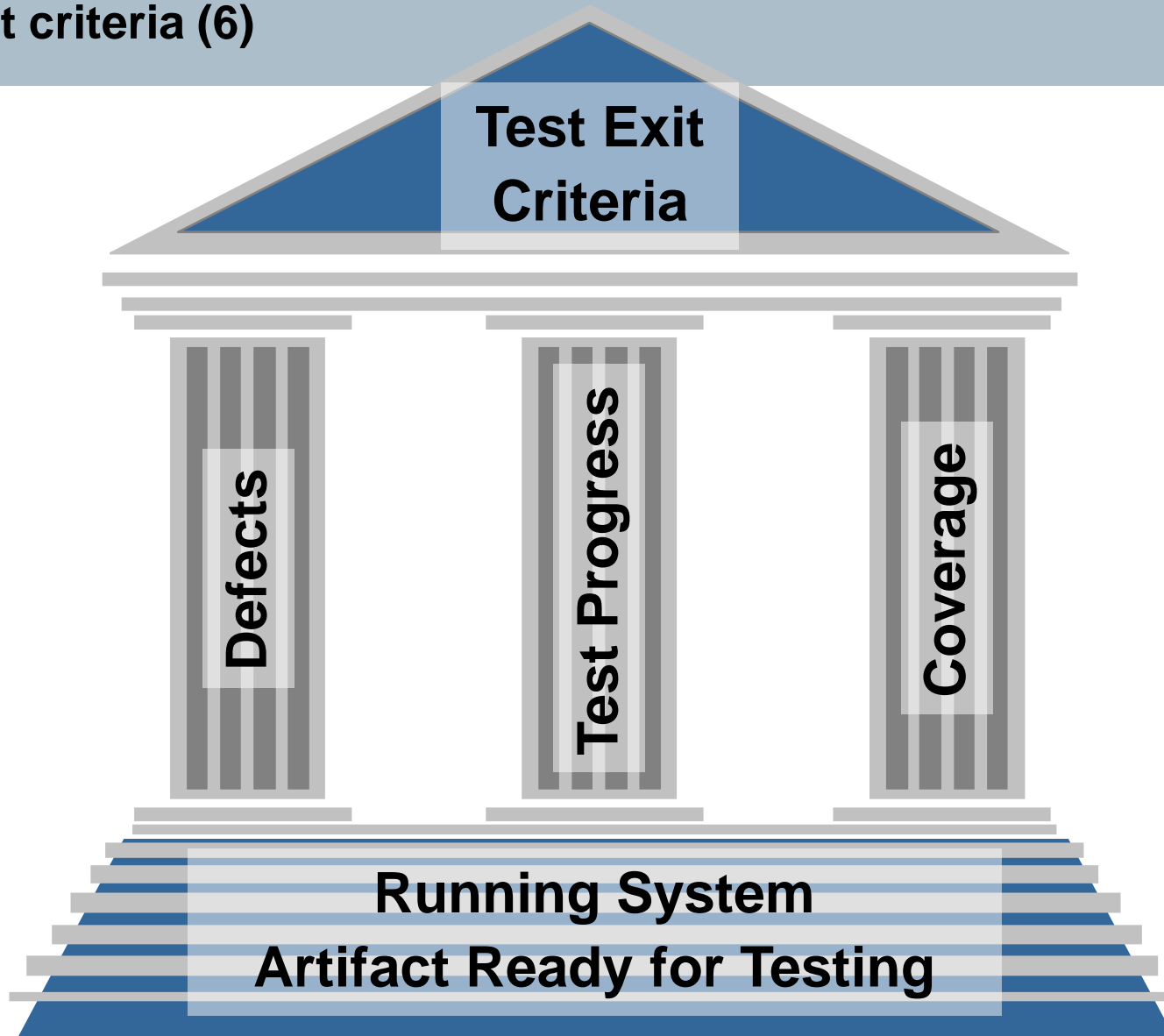
# Testing and evaluation throughout the lifecycle



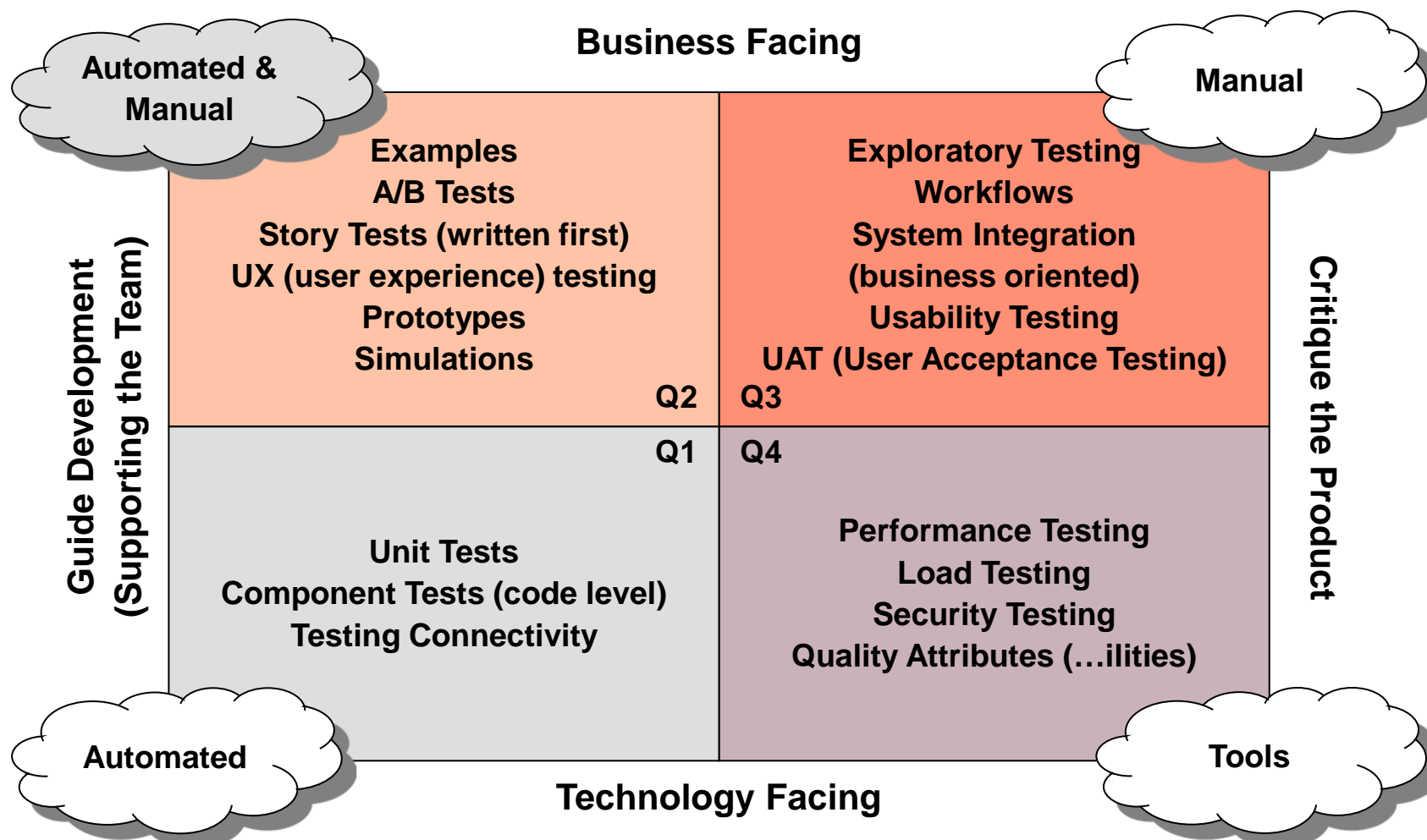
# Example: Evaluation methods and testing types



## Test exit criteria (6)

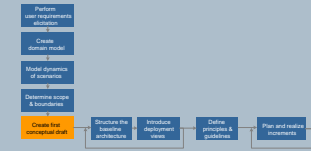


# Agile testing quadrants



Reference: Brian Marick, Lisa Crispin, Janet Gregory

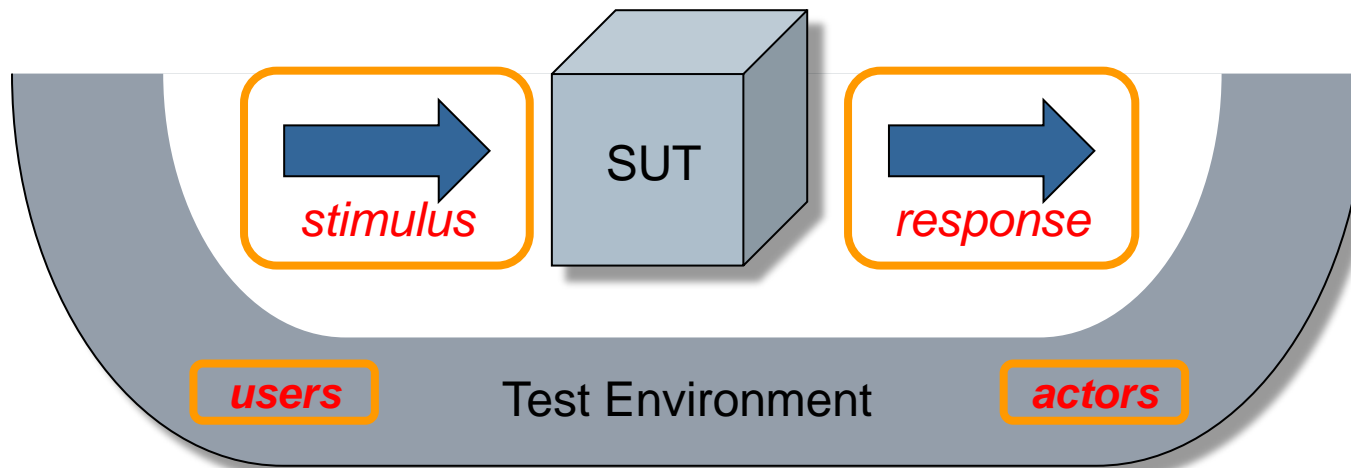
## Domain model is fundamental!



For the product / system it clearly defines the **boundaries**  
what is **IN** and what is **OUT of scope**

Your test system is a companion, you have to **cover** all the **other** elements,  
e.g.

- Human user interaction (simulated or real)
- Communication with other actors (external systems, ...)
- Identify capabilities of these elements and design test cases accordingly



## Architecture in a nutshell

***“Architecture is about everything costly to change”***



### Main activities

- Taking decisions, based on **architecture significant requirements (ASR)**, aka. **architectural drivers**
- **Structuring** and **partitioning** the **technical realization**, based on domain knowledge
- Defining **interfaces** and **functional responsibilities**
- Assigning **functional components** to **hardware deployments**
- Documenting and disseminating **architecture descriptions**

### Architecture governance

- Make sure that the **implementation follows the architecture**
- **Supporting** and **coaching** other **stakeholders**

➔ TeA works in close collaboration with other architects!



# Why creating an architecture for a test system?

Test architect has responsibility to create a test environment that **other stakeholders** can use

- Unit test environment: developer creates test cases
- Test automation

Your test environment may become a **product!**

Integrators, factory, CS, key customers, ... can use your test environment

- Test environment requires product quality
- Requirement to create copies of the test environment in an efficient way
- Test the test environment for effectiveness

Test architect is the owner of vital **properties of test environment**

- Functional correctness of test execution
- Determinism (repeatability, dependability), extensibility, adaptability, cost efficiency, performance, configurability, availability

➔ Follow a systematic method to satisfy these qualities!

