

# REQUIREMENTS ENGINEERING

---

## Sentence Template for System Requirements

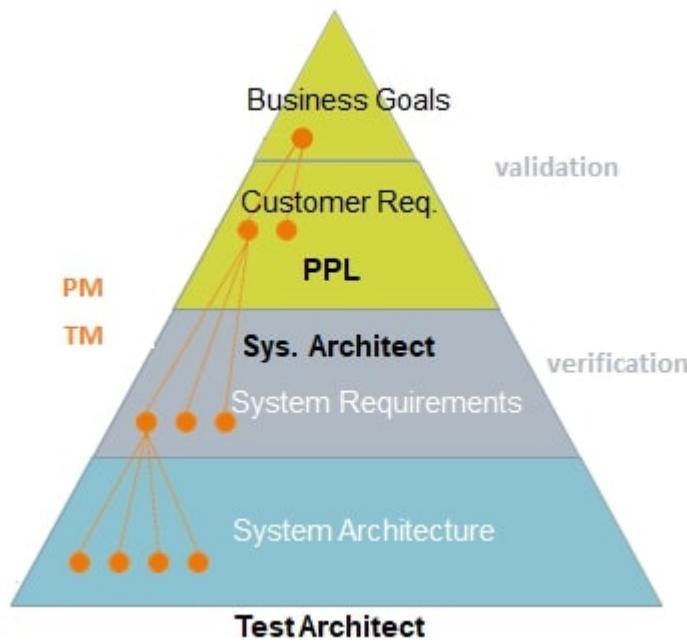
Given When Then



## Characteristics of Good Requirements

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

## Requirements Traceability



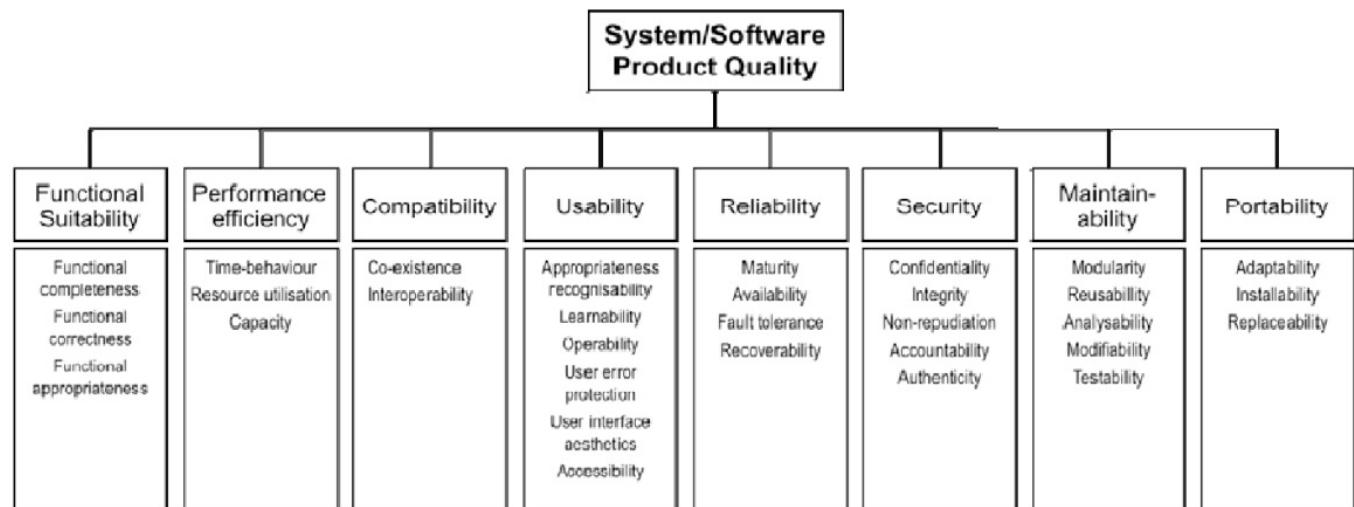
Tracing Usage:

Analysis	Description	Process
Derivation	<i>Why is this here?</i>	cost-benefit analysis
Impact	<i>What if this changed?</i>	change management
Coverage	<i>Have I covered all reqs?</i>	management report

## NFRs / Quality Attributes

[Software Quality Characteristics pdf](#)

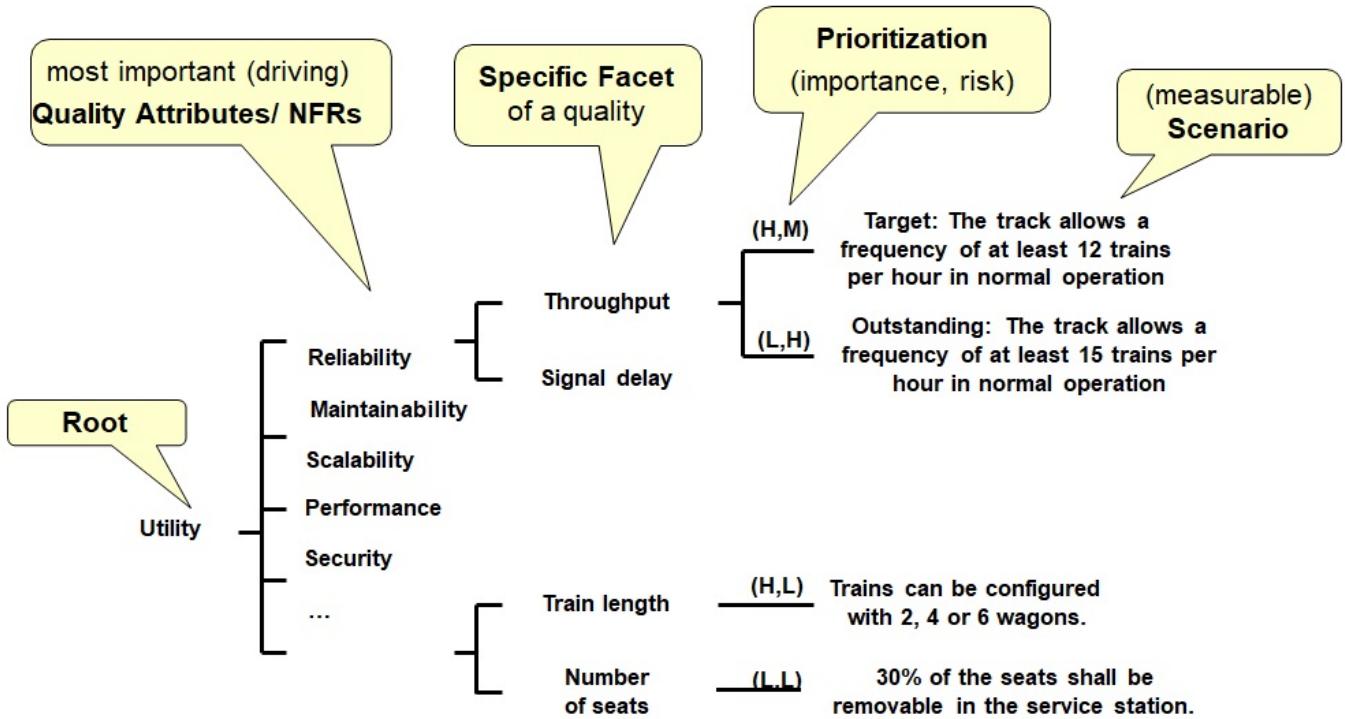
### New ISO/IEC 25010 Product quality model



Step 1: Utility Tree

## Utility Tree

Ingenuity for life



## Step 2: Scenario Description Template / Workflows

To reduce ambiguity and increase testability, each measurable scenario in the Utility Tree gets described with this template.

### Scenario description Template

**SIEMENS**  
Ingenuity for life



<b>Source of stimulus</b>	Who/what initiates the scenario.
<b>Stimulus</b>	Which periodic, stochastic or sporadic event initiates the scenario.
<b>Artifact</b>	What is the relevant unit; e.g. a (part of a) system or a feature.
<b>Environment</b>	What is the environmental condition for this scenario; e.g. normal, startup / shut down, maintenance, emergency, overload, etc.
<b>Response</b>	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).
<b>Response measure</b>	<p>How can the response be measured, using indicators like:</p> <ul style="list-style-type: none"> <li>▪ the time it takes to process the event (latency or deadline); or the variation in time (jitter)</li> <li>▪ the amount of data, material or energy that can be processed in a particular time interval (throughput)</li> <li>▪ or a characterization of the events that cannot be processed (e.g. miss rate, data / energy / material loss)</li> </ul>

## Step 3: Refine Scenarios

## Scenario refinement table

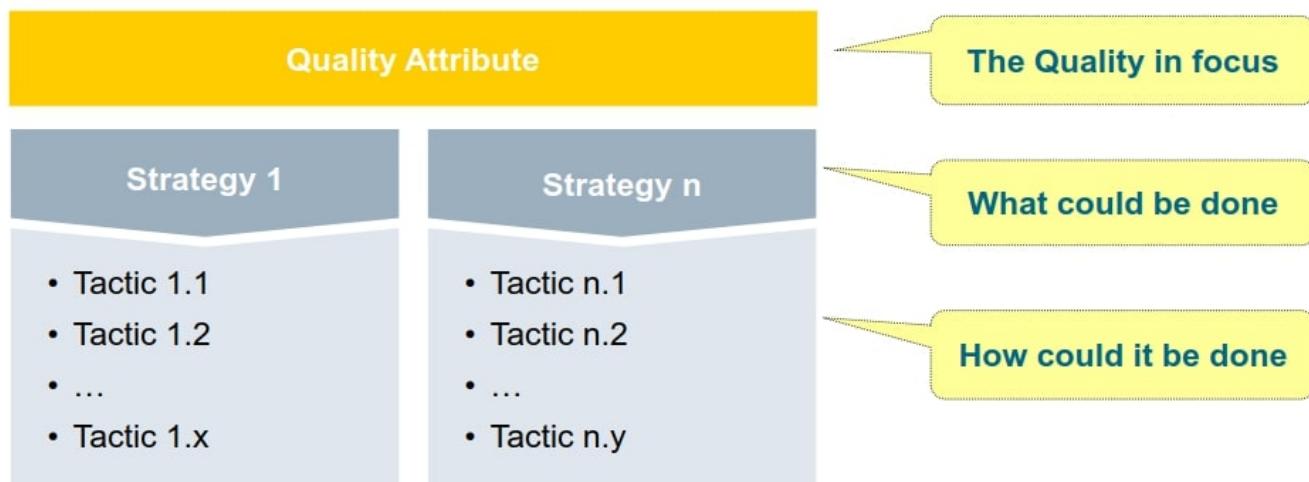
### Example from SEI Quality Attribute Workshop (QAW)

Scenario Refinement for Scenario N		
Scenario(s):	When a garage door opener senses an object in the door's path, it stops the door in less than one millisecond.	
Business Goals:	safest system; feature-rich product	
Relevant Quality Attributes:	safety, performance	
Scenario Components	Stimulus:	An object is in the path of a garage door.
	Stimulus Source:	object external to system, such as a bicycle
	Environment:	The garage door is in the process of closing.
	Artifact (If Known):	system's motion sensor, motion-control software component
	Response:	The garage door stops moving.
	Response Measure:	one millisecond
Questions:	How large must an object be before it is detected by the system's sensor?	
Issues:	May need to train installers to prevent malfunctions and avoid potential legal issues.	

Step 4: Implement Design Strategies to support the NFRs

.. and use Design patterns when needed...

### Design Strategy Template for Quality Attributes



Performance of IT Systems			Increase Testability (NFR/ Quality Attribute)		
Resource demand	Resource management	Resource arbitration	make it simple	make it observable	make it controllable
<ul style="list-style-type: none"> <li>• Increase computation efficiency</li> <li>• Reduce computational overhead</li> <li>• Manage event rate</li> <li>• Control frequency of events / raw data</li> </ul>	<ul style="list-style-type: none"> <li>• Introduce concurrency</li> <li>• Maintain multiple copies of data/services</li> <li>• Adapt HW resources (processor, memory, network)</li> </ul>	<ul style="list-style-type: none"> <li>• Scheduling policy</li> <li>• Distribution</li> <li>• Localization</li> </ul>	<ul style="list-style-type: none"> <li>• Interface driven</li> <li>• No code dupe</li> <li>• Low cyclo.complex.</li> <li>• Low coupling</li> </ul>	<ul style="list-style-type: none"> <li>• Log</li> <li>• Trace</li> <li>• Testable interfaces</li> </ul>	<ul style="list-style-type: none"> <li>• Dependency inj.</li> <li>• Law of Demeter</li> <li>• Config &amp; data files</li> <li>• Object IDs (auto.)</li> </ul>

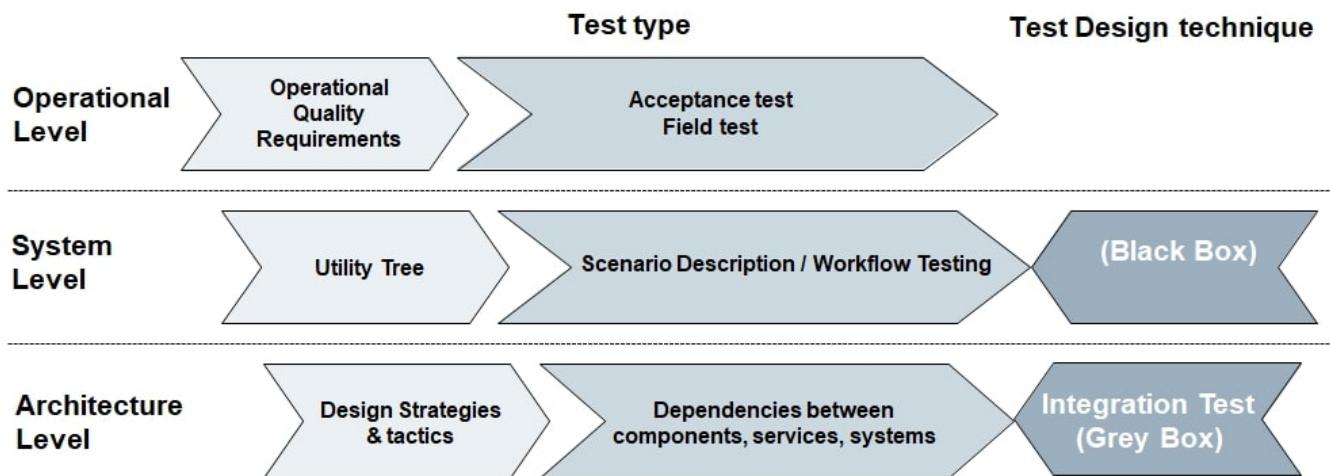
### Design Pattern example

#### "Performance of Image Processing"

Performance of Image Processing			Caching	Eager Acquisition	Resource Lifecycle Manager
Resource demand	Resource management	Resource arbitration	Pooling	Resource Pool	Resource Environment
<ul style="list-style-type: none"> <li>• Use advanced image processing algorithms (if possible GPU based)</li> <li>• Introduce caching strategies</li> <li>• Either allow upper bound of clients or scale-out mechanisms depending on resources</li> </ul>	<ul style="list-style-type: none"> <li>• Introduce a pool of worker threads for background loading, storing, Pooling</li> <li>• Don't copy mass data but use meta files and refs for image processing/copying</li> <li>• Use thumbnail images for browsing</li> <li>• Only use full resolution when images are Eager Acquisitioned</li> <li>• Apply eager</li> </ul>	<ul style="list-style-type: none"> <li>• Schedule resources preferably for processing visible images</li> </ul>	<p><b>Caching</b></p> <ul style="list-style-type: none"> <li>• Context: Systems that repeatedly access the same set of resources and need to optimize performance.</li> <li>• Solution: Resources stored temporarily in fast-access buffer (cache); subsequent access through cache instead of resource provider.</li> </ul>	<p><b>Eager Acquisition</b></p> <ul style="list-style-type: none"> <li>• Context: Systems that must satisfy high predictability and performance in resource acquisition time.</li> <li>• Solution: Resources eagerly acquired before their actual use by a provider proxy; resources then kept in an efficient container; requests intercepted by the proxy.</li> </ul>	<p><b>Resource Lifecycle Manager</b></p> <ul style="list-style-type: none"> <li>• Context: Systems that require simplified management of the lifecycle of their resources.</li> <li>• Solution: Resource usage separated from resource management through introduction of a Resource Lifecycle Manager (RLM).</li> </ul>

## Testing NFRs

## Testing NFRs

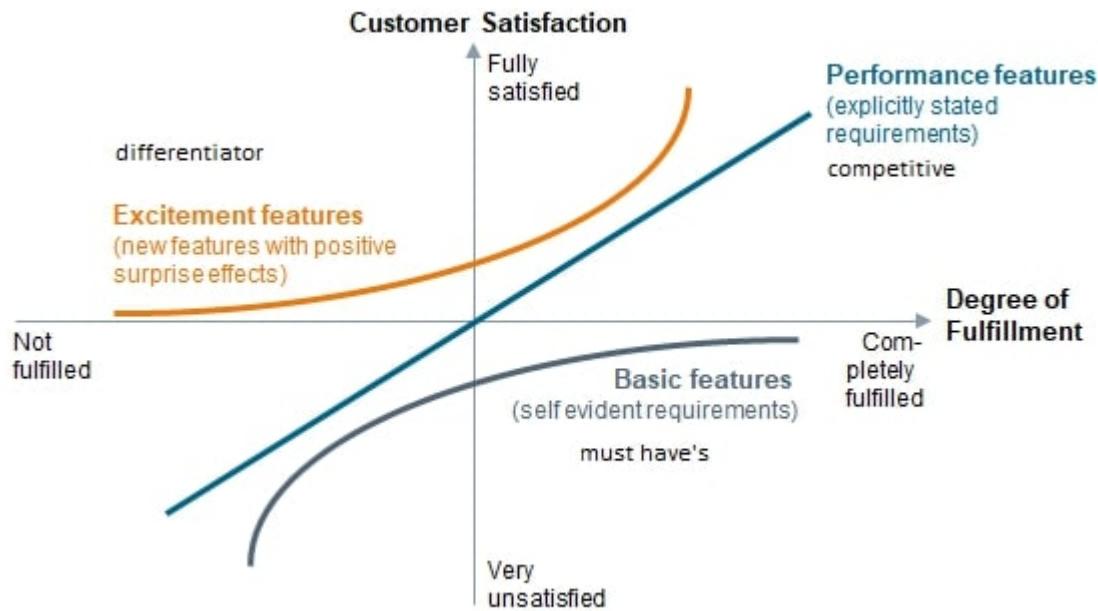


## Some NFR testing approaches

<b>Security</b>	<b>Performance</b>	<b>Conformance</b>
<ul style="list-style-type: none"> <li>▪ Fuzz Testing</li> <li>▪ Vulnerability assessment</li> <li>▪ Vulnerability testing</li> <li>▪ Penetration test</li> <li>▪ Security audit</li> <li>▪ Security review</li> <li>▪ Network scanning</li> </ul>	<ul style="list-style-type: none"> <li>▪ Load testing</li> <li>▪ Stress testing</li> <li>▪ Spike testing</li> <li>▪ Endurance/Soak testing</li> <li>▪ Benchmarking</li> </ul>	<ul style="list-style-type: none"> <li>▪ Protocol tests</li> <li>▪ Radiated immunity testing</li> <li>▪ Radiated emissions testing</li> <li>▪ Conformity assessment</li> </ul>
<b>Safety</b>	<b>Usability</b>	<b>Others</b>
<ul style="list-style-type: none"> <li>▪ Hazard and operability analysis/study (HAZOP)</li> <li>▪ Defect history tracking</li> <li>▪ Statistical testing</li> <li>▪ Probabilistic risk assessment (PRA)</li> <li>▪ Failure mode and effect analyses (FMEA)</li> <li>▪ Fault tree Analysis (FTA)</li> <li>▪ Reliability Testing</li> </ul>	<ul style="list-style-type: none"> <li>▪ Hallway testing</li> <li>▪ Remote usability testing</li> <li>▪ (Automated) Expert review</li> </ul>	<ul style="list-style-type: none"> <li>▪ Recovery testing</li> <li>▪ Volume testing</li> </ul>
<b>Additionally – on Architecture Level many test design techniques can be used, too</b>		

## KANO Model

For the Product Manager KANO is a tool to find the right mix between the different feature types. For the Test Architect KANO helps to identify risks.



## Configuration Management

Source Control, configurations, environment...

**Devops:** continuous integration, deployment, testing...

### Attributes of Professional Configuration Management

- Standard artifact repository structures
- Naming conventions
- Status Model
- List of (identified) configuration items
  
- Version management / control for frequently changing artifacts
- Change control for important artifacts, at least for released artifacts
- Baselineing for (sets of) released artifacts
- Access right management (e.g. read, write, delete)
- Access synchronization (e.g. check out / in, branch / merge)
- Appropriate tool and IT support
- Managed backups
  
- Archiving for artifacts (Documents, SW, HW) that are needed beyond project lifetime

Issue of organizational and project process

Provided or supported by a CM tool

Archive

## Change Request Management

**Change Request:** any issue that cannot go in a patch or point rev, has to go into the product asap.

**Change Management:** because requirements are incomplete, erroneous, ambiguous.

**Defect Management:** because tests reveal defects that need correction.

Tool -> Feature -> trace & Test

CCB:

## Change Request Management: Evaluate and Decide on Change



**SIEMENS**  
*Ingenuity for life*

**Understand impact on the system and all disciplines or sub-systems.**

### Decision by Change Control Board (CCB)

A group of persons with assigned responsibility and authority to make decisions on the change.

Typically includes

- Product Manager
- Project Manager
- (System) Architect

### Impact Analysis

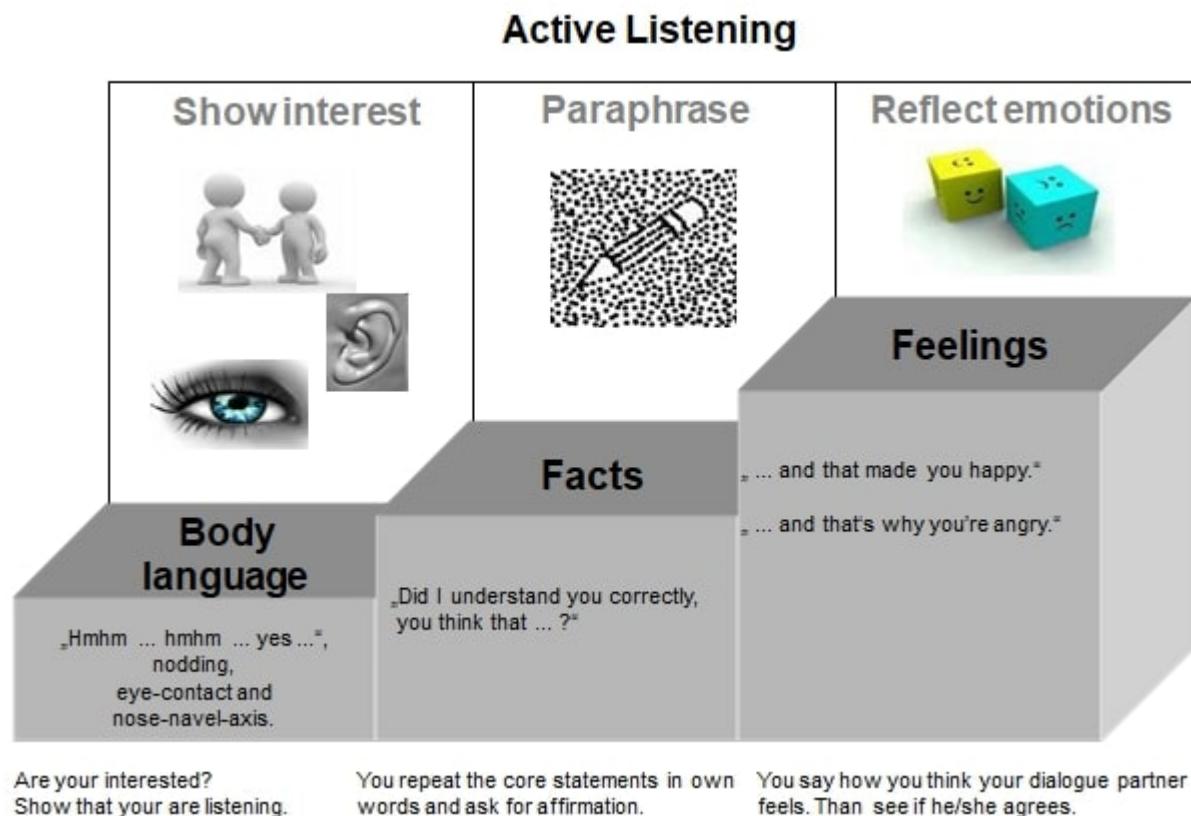
- Rough initial evaluation by the CCB to decide on further action
- Deep analysis by experts
  - on architecture, cost, schedule
  - all affected disciplines
  - all relevant sub-systems
- Propose further proceeding (accept / defer / reject)

# SOCIAL CAPABILITY

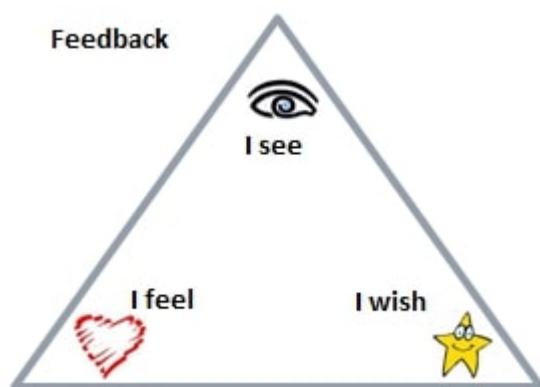
---

## Active Listening

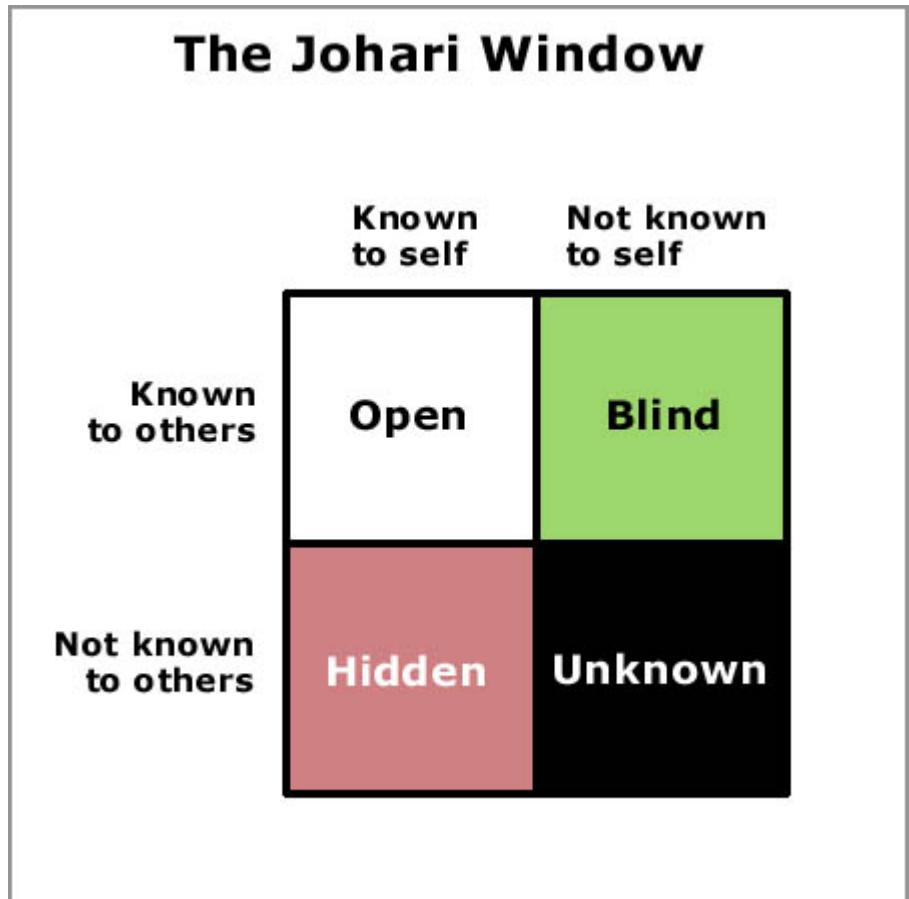
Do not give solution!



## Feedback



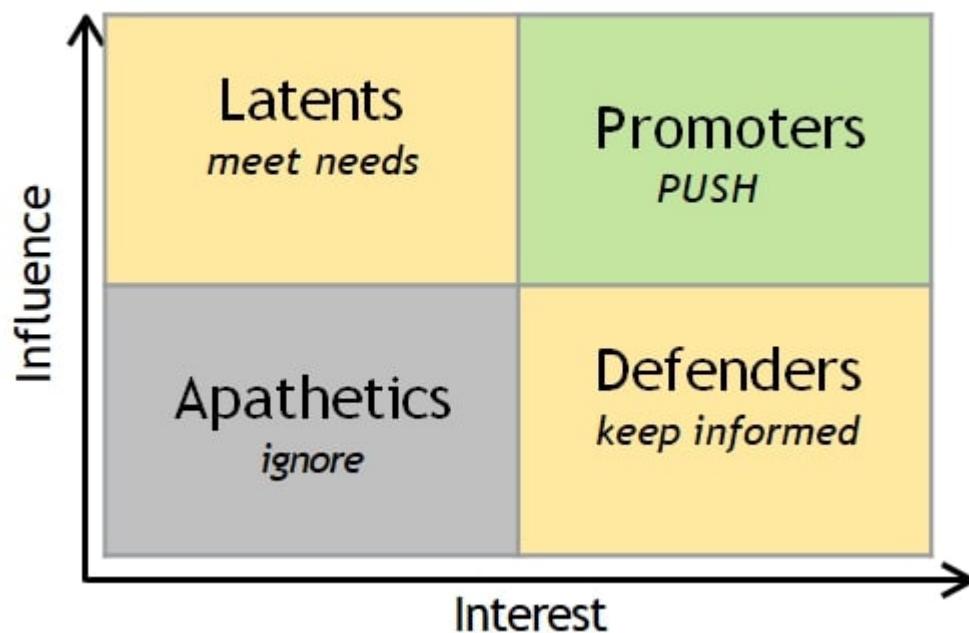
You can use feedback for blind spots.



## Stakeholders

Who's buy-in you need?

### Stakeholder Analysis



## Situational Leadership

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

Knowing	<ul style="list-style-type: none"> <li>▪ Does he know what you expect him to do?</li> <li>▪ Does he know the goal and why it is important?</li> </ul>
Willing	<ul style="list-style-type: none"> <li>▪ Is he motivated to do it?</li> <li>▪ What would motivate him?</li> </ul>
Able	<ul style="list-style-type: none"> <li>▪ Does he know how to do it?</li> <li>▪ Does he have the necessary skills and experience?</li> </ul>
Allowed	<ul style="list-style-type: none"> <li>▪ Do the circumstances allow him to fulfill the task?</li> <li>▪ What barriers have to be removed?</li> </ul>

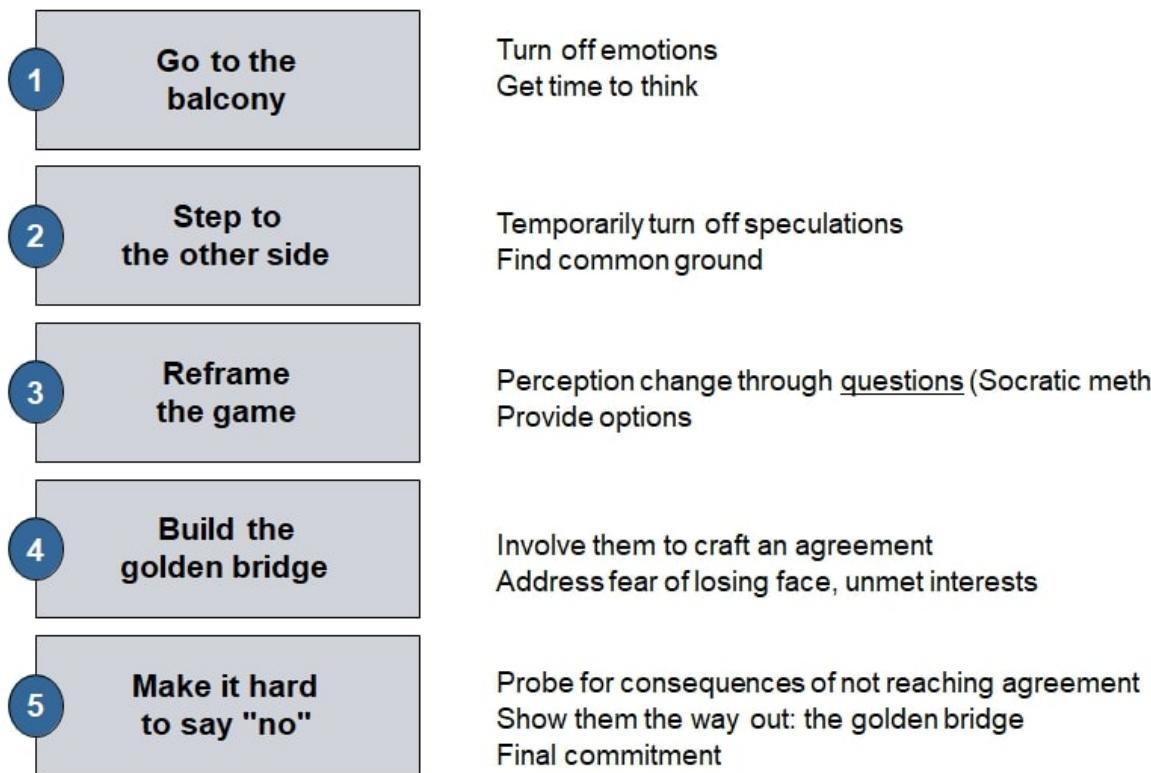
## The Situational Leadership Model



## Conflict Management

# Getting past “No”: Five steps to master difficult conversations

**SIEMENS**  
Ingenuity for l



# SYSTEMATIC ARCHITECTURE

---

What is architecture?

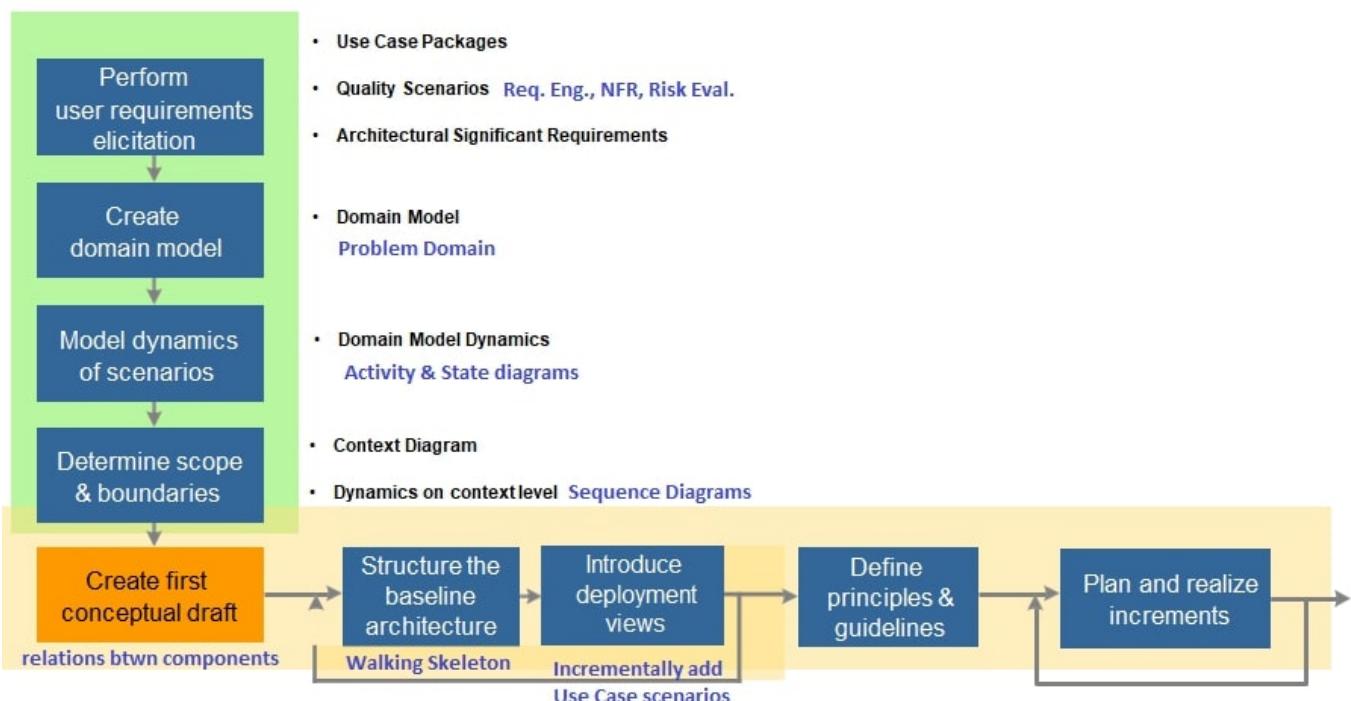
- Abstraction of reality
- Focus on fundamental & critical concepts
- Rationels / Why

Architecture Design = Creative freedom - Forces: (*func./non-func. reqs + org./biz./processes*)

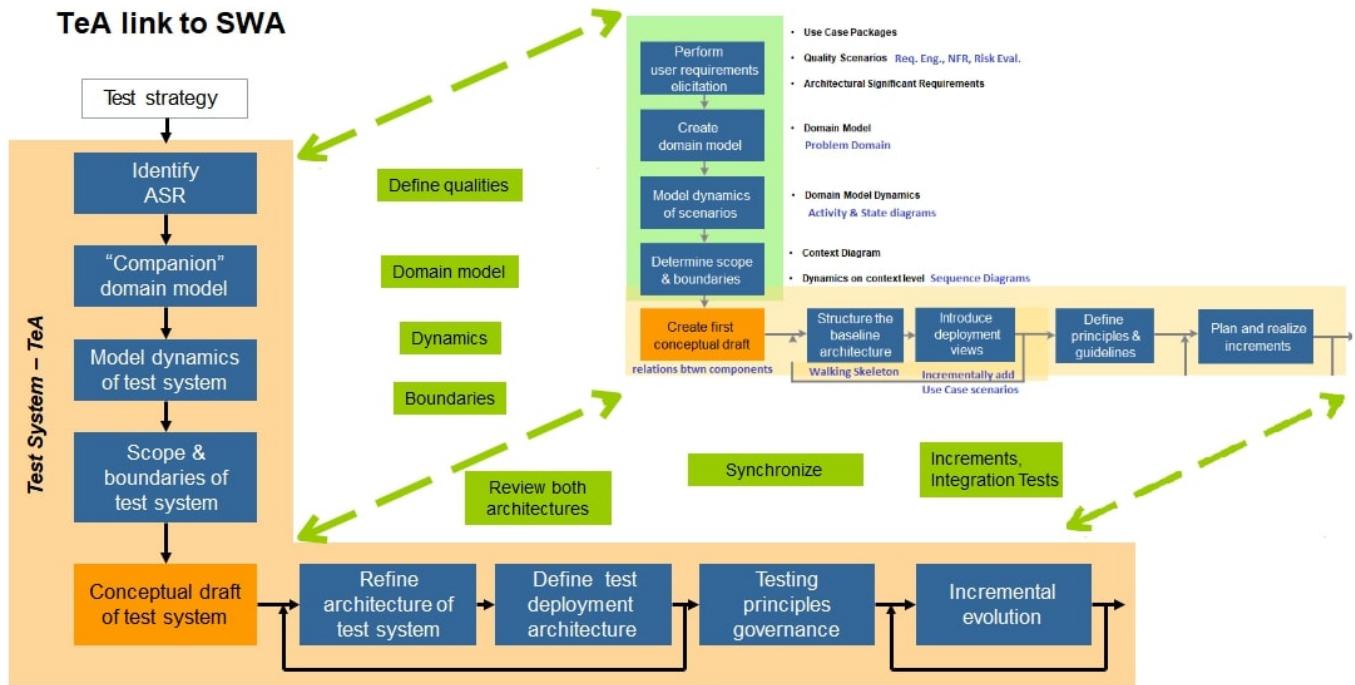
## Architecture Design - step by step

---

### Architecture Design Step by step Overview



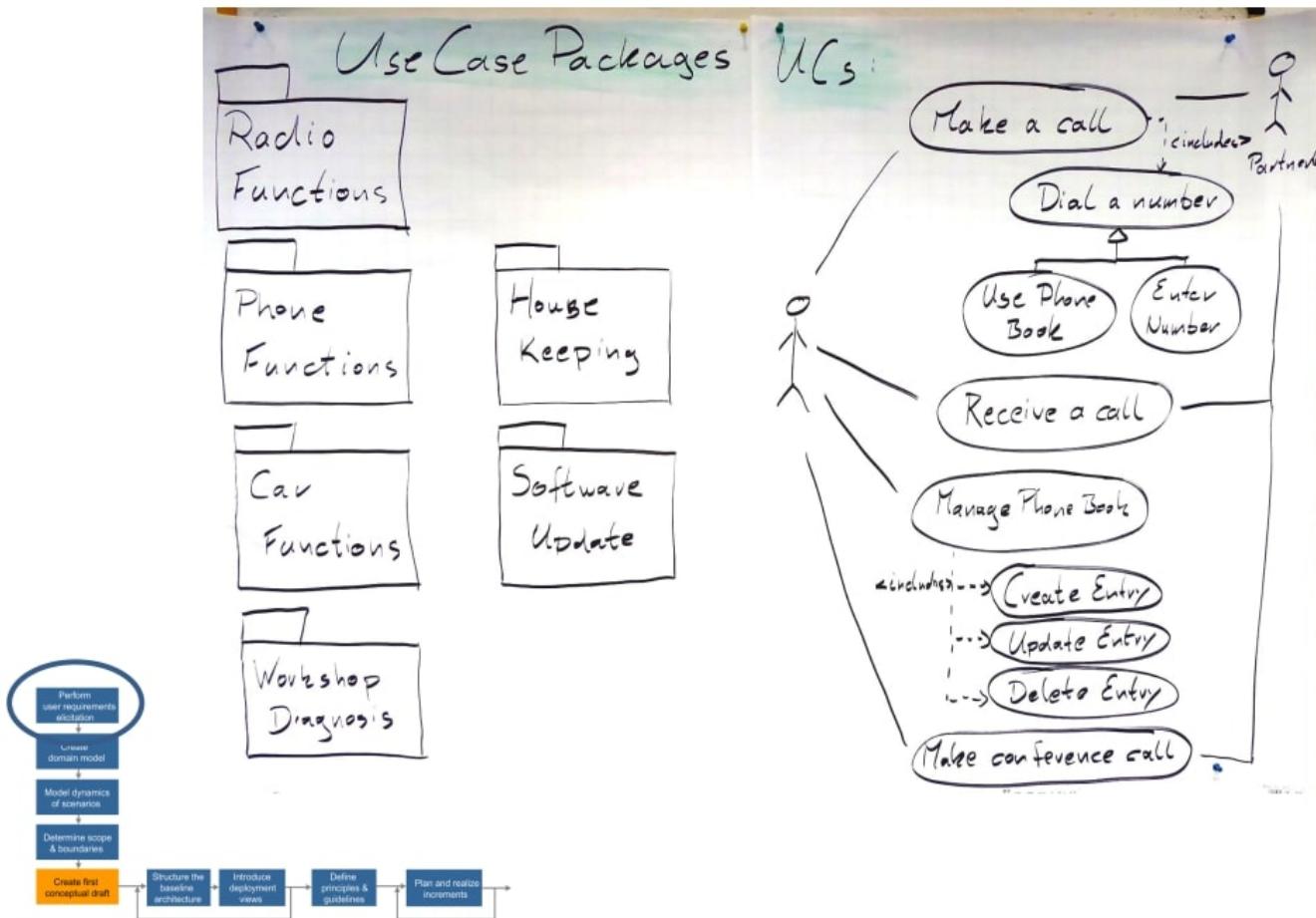
Link between TeA and SWA



## 1) SWA: Requirements Elicitation - TeA: Identify ASR's

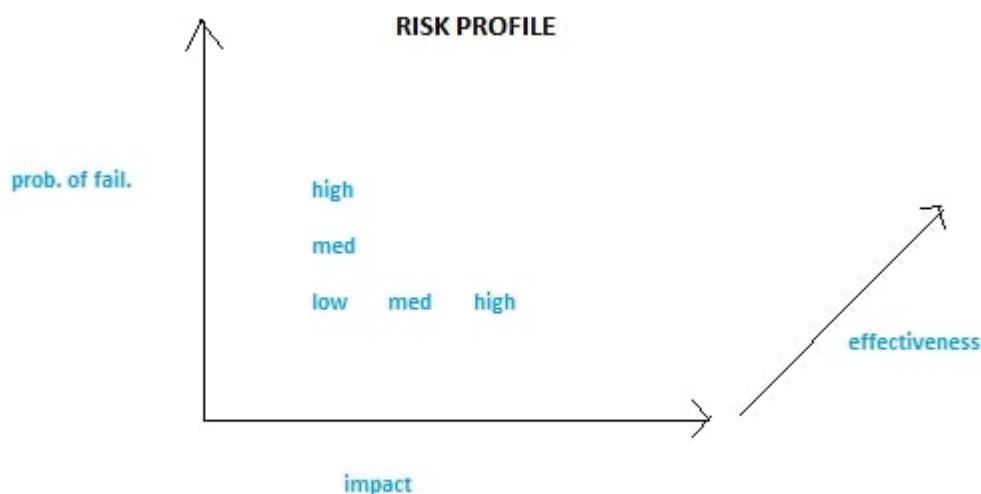
### Use Case Packages

Each use case is a requirement.



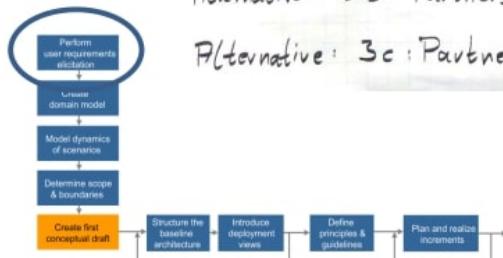
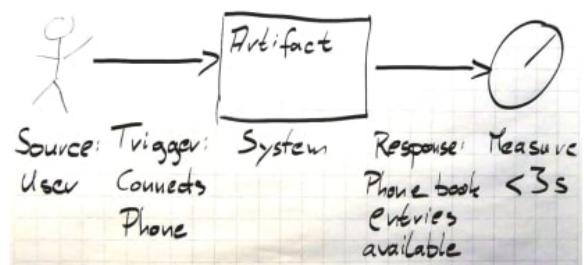
### Quality Scenarios

- Requirements Engineering: Given When Then
- NFRs
- Risk based evaluation: Risk Based Testing



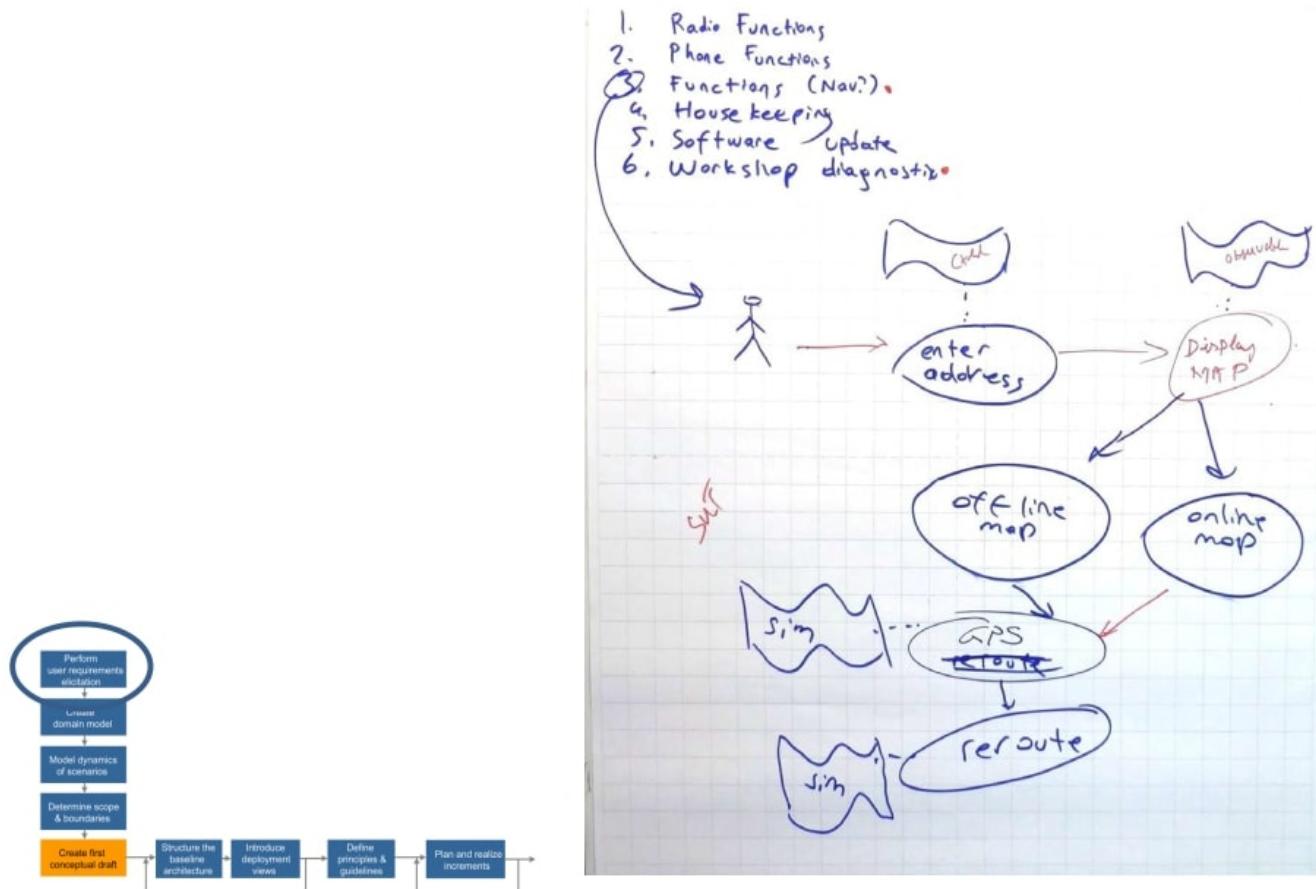
Name: Make a call  
 Factors: User, Call partner  
 Trigger: User indicates call wish  
 Pre Cond: Phone connected, idle  
 Post Cond: Idle  
 Success Scen.:  
 1. User dials a number  
 2. System establishes call  
 3. Call partner accepts  
 4. User & Partner talk  
 5. User hangs up  
 Alternative: 2a: Call cannot be established  
 Alternative: 2b: Partners line is busy  
 Alternative: 3c: Partner rejects

## Quality Scenarios



## Architectural Significant Requirements (ASR's)

## ASR's - Architectural Significant Requirements



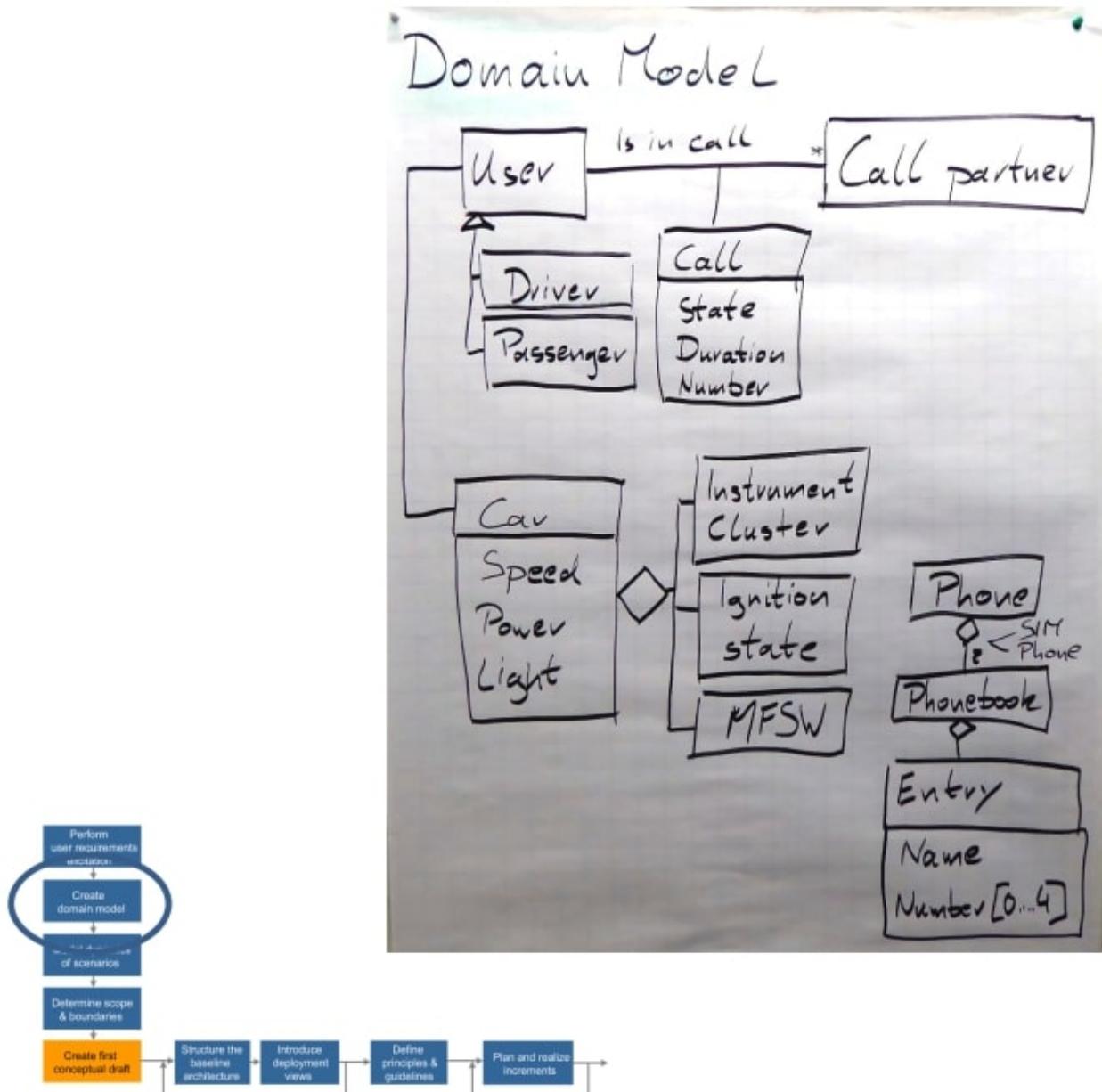
## 2) SWA: Domain Model - TeA: "Companion" Domain Model

This is not a class diagram, but the **Problem Domain**. It is a conceptual model that incorporates behavior and data.

Question: what is the relation between a Domain Model and a requirement?

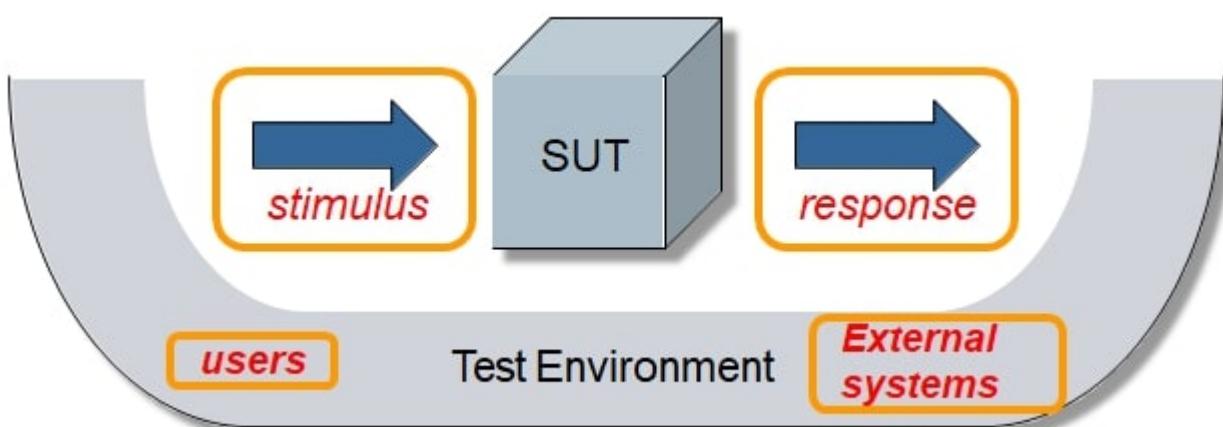
There can be long discussions on requirements – features – use cases – epics – domain models – context diagrams – package diagrams – component diagrams – etc. These are different perspectives/views/notations used to specify something with different intent, purpose, scope, level of detail.

- *Requirement* : capability or condition needed by a stakeholder
- *Domain Model* : conceptual model that incorporates behavior and data
- *Context diagram* : focuses on what is in/out of scope, boundary between system and environment



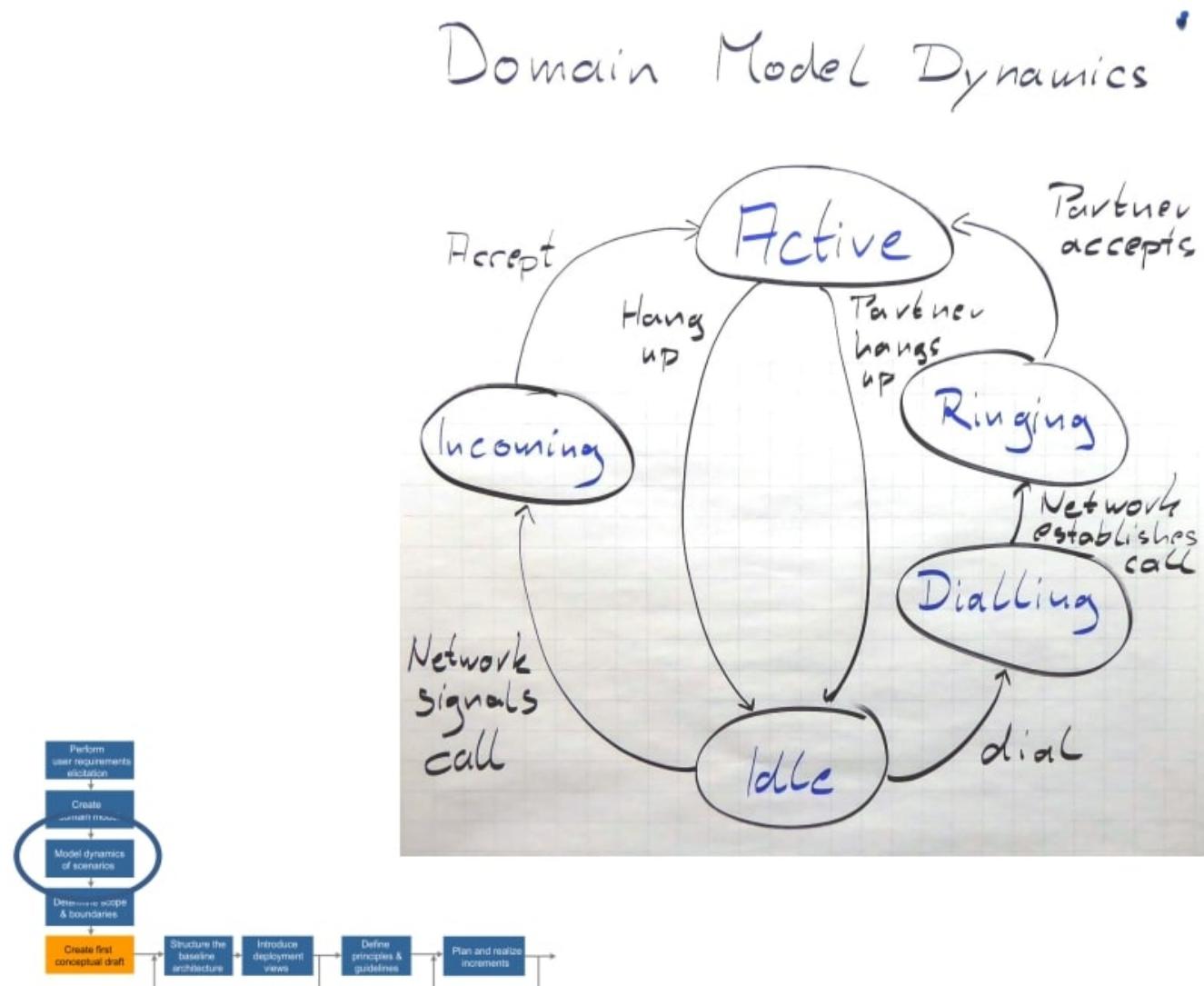
Your Test System is the "companion" Domain Model, you have to cover all the other elements:

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



### 3) SWA: Domain Model Dynamics - TeA: Test System's Model Dynamics

Activity & State Diagrams



### 4) SWA: Determine Scope Boundaries - TeA: Test System's Scope & Boundaries

Context Diagram

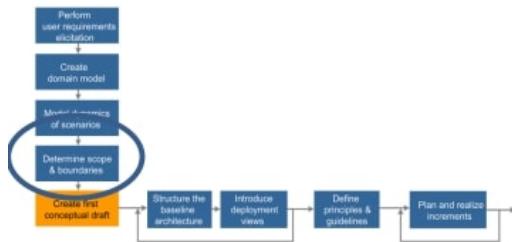
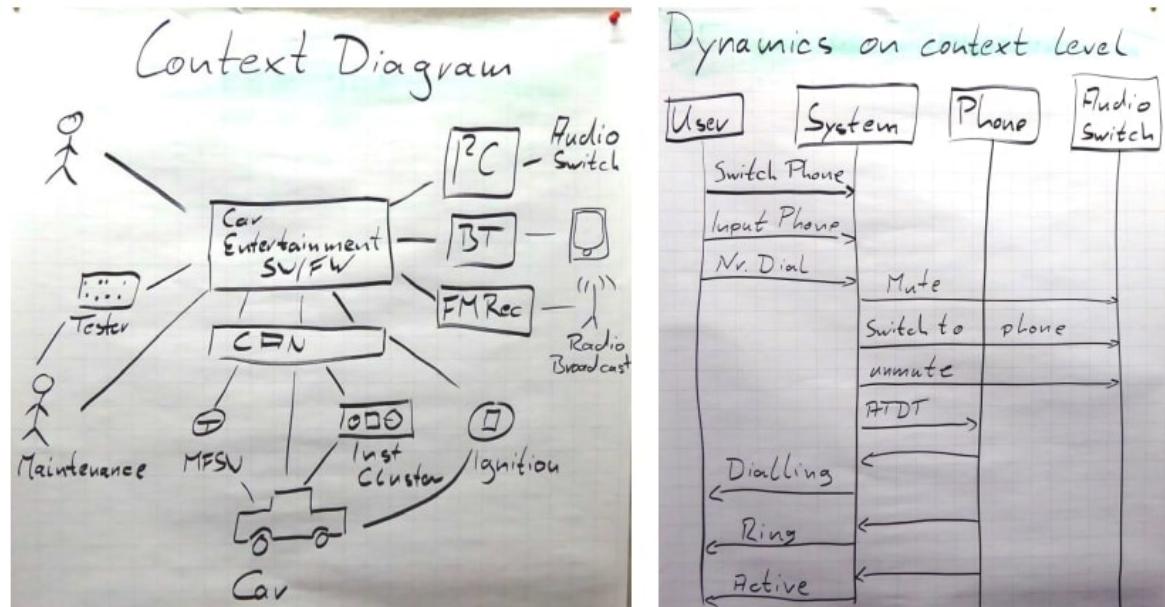
For the product/ system, clearly define the boundaries: **What is IN and what is OUT of scope, boundary between system and environment.**

Context diagram may look like a use case package diagram but has different intent:

- context diagram: describes what is in/out of scope, the so-called “context”, the interaction between system and the environment around.
- use case package diagram: shows packages and relationships between them, but can be more an “internal only” view.

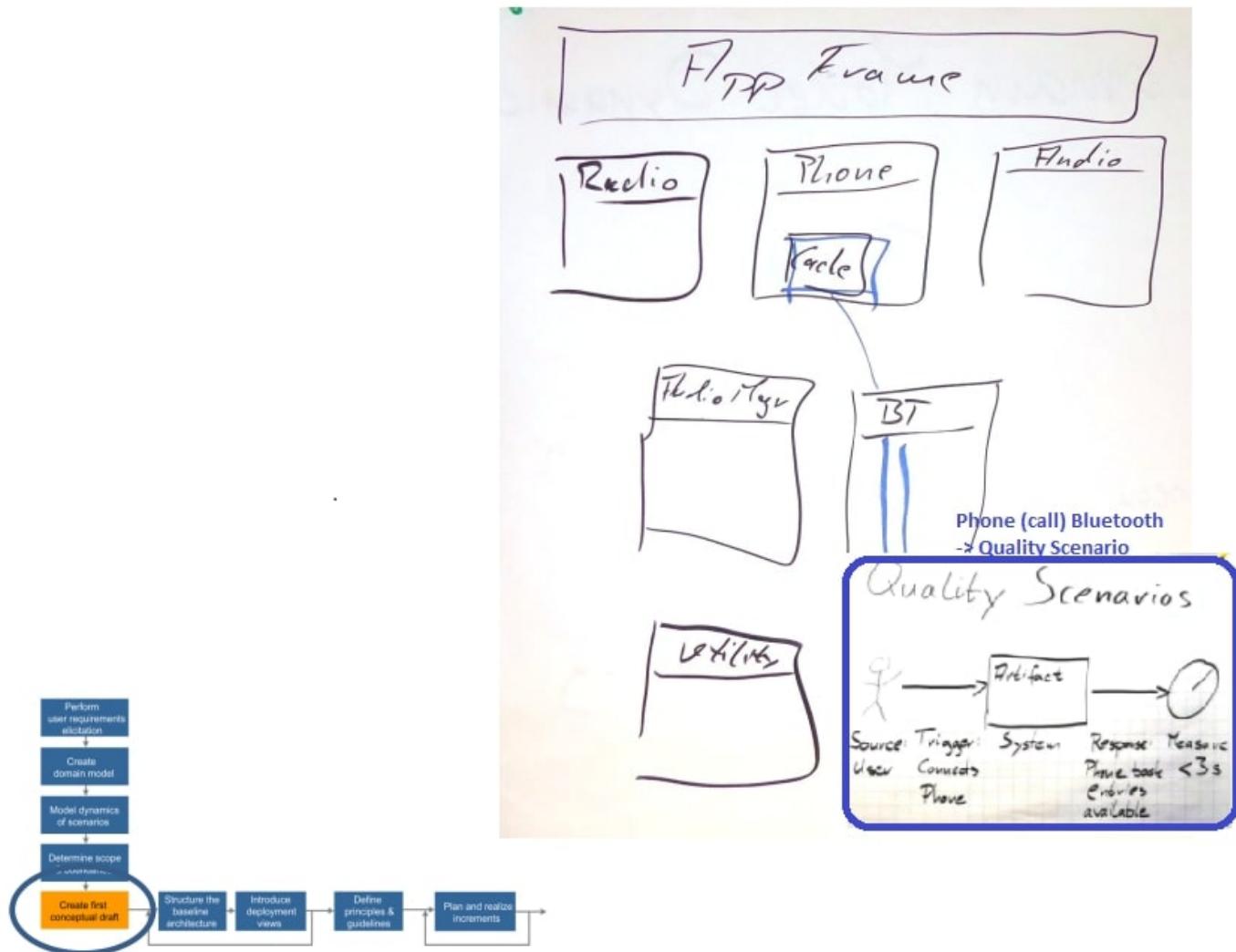
Dynamics on Context level

Sequence Diagrams



## 5) SWA: Conceptual Draft - TeA: Test System's Conceptual Draft

Relations between components.



6+) SWA: Structure the baseline Architecture & Introduce Deployment Views - TeA: Refine test architecture & Define test deployment architecture

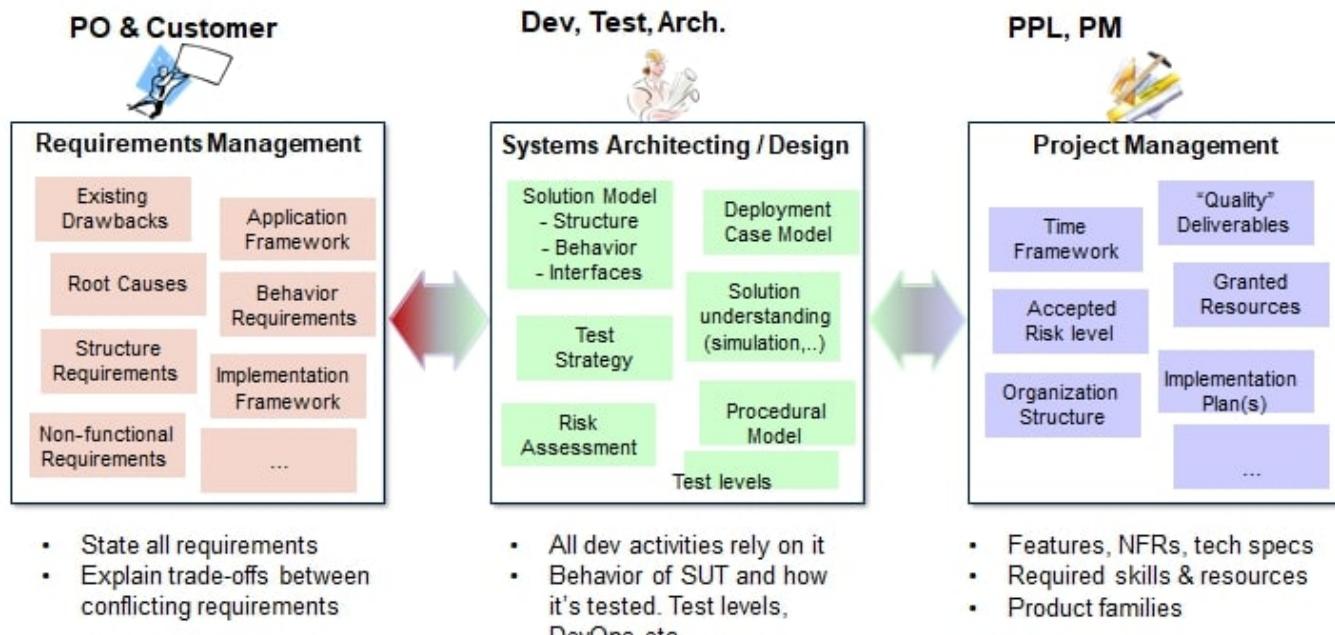
- Walking Skeleton : most important & highest risk
- Incrementally add use case scenarios

## Architectural Views & Documentation

What should be documented?

- Context & boundaries
- UML views of the architecture itself
- Design rationale
- How the architecture addresses FRs & NFRs & cross-cutting concerns

## Architectural views & documentation Stakeholders



Architectural Views: Kruchten, Zachmann...

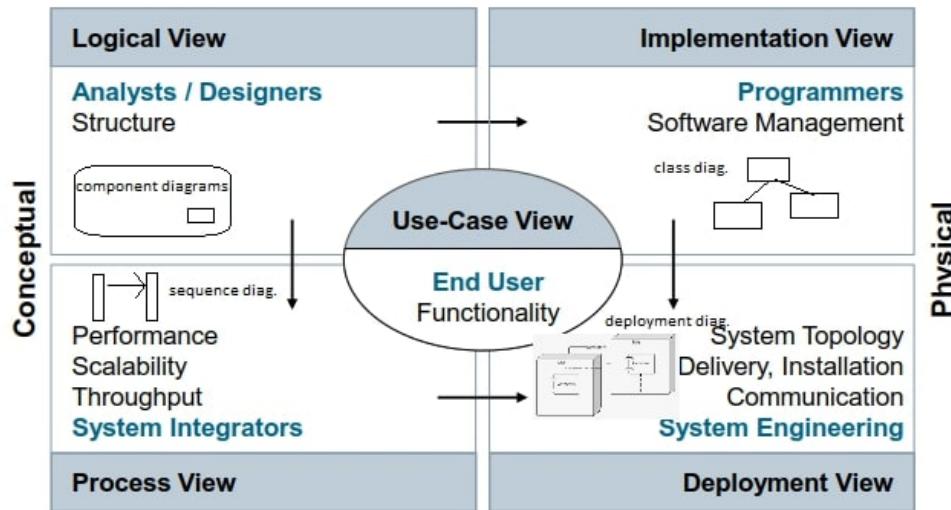
4+1 View explained:

- User view: all possible scenarios the user expects from SUT
- Functional aspects:
  - Logical View: how the functionality use cases are modeled. *Component diagrams*
  - Implementation/Development View: how the functionality is implemented (source code, libs, executables etc.) *Class diagrams*
- Non-functional aspects:
  - Process View: how the artifacts will be executed in terms of concurrency, scalability, synchronization. *Sequence, Activity, State diagrams*
  - Deployment View: maps software artifacts to hardware entities and shows the distribution of functionality. *Deployment diagrams*. one view in the 4+1 views by Kruchten, sometimes also

called physical view, see [https://en.wikipedia.org/wiki/Deployment\_diagram]

## 4+1 View (Kruchten)

Rational Unified Process 4+1 View introduced by Philippe Kruchten



## Siemens SW Architecture Doc Template

System Architecture Description	Version: 0.5
System_Architecture_Description_TEMPLATE.xlsx	Page: 3/34 Date: 2017-01-11 Status: preliminary

### Contents

<b>1 PREFACE</b>	<b>8</b>
1.1 Purpose and Audience	8
1.2 Responsibilities	8
1.3 Document Structure	8
1.4 Definitions of Terms and Abbreviations	9
1.5 References	9
<b>2 &lt;PRODUCT/SYSTEM&gt; SCOPE</b>	<b>10</b>
2.1 Business Context	10
2.2 Architecturally Significant Requirements	10
2.3 Quality Attributes	10
2.3.1 Overview	10
2.3.2 Example Quality Attribute: Performance	11
2.3.3 <Quality Attribute #2>	11
2.3.4 <Quality Attribute #N>	11
2.4 Standards	11
2.5 Boundary conditions for development	12
2.5.1 Runtime Environment	12
2.5.2 Programming Environment	12
2.5.3 Conventions	13
2.5.4 Teams and Sites	13
2.5.5 Third-party Components	13
2.5.6 Reuse of Components	13
2.6 Patents	14
2.6.1 Foreign Patents	14
2.6.2 New SIEMENS Patent Candidates	14
2.7 Architectural Risks	14
2.7.1 Risks Introduced by Boundary Conditions	15
2.7.2 Risks Introduced by Implementation Decisions	15
<b>3 OUTSIDE VIEWS</b>	<b>16</b>
3.1 Domain Model	16
3.2 System Context	16
3.2.1 High-Level View	16
3.2.2 Interfaces	17
3.2.3 Collaboration	18
3.3 Use Case View	18
<b>4 INSIDE VIEWS</b>	<b>20</b>

System Architecture Description
System_Architecture_Description_TEMPLATE.xlsx

Version: 0.5
Page: 4/34
Date: 2017-01-11
Status: preliminary

<b>4.1 Logical View</b>	<b>20</b>
4.1.1 <Component #>	20
4.1.2 <Component #B>	21
4.1.3 <Component #B>	21
4.1.4 <Interface #>	21
4.1.5 <Interface #B>	22
<b>4.2 Runtime View</b>	<b>22</b>
4.2.1 Component Distribution and Concurrency	22
4.2.2 Component Communication	22
4.2.3 Processes and Tasks	23
4.2.4 Scheduling and Deterministic Behavior	24
4.2.5 Start-up, Shut-down, and Recovery	25
4.2.6 Exception Handling and Error Processing	25
4.2.7 Persistent Memory	25
4.2.8 Logging, Tracing, Reporting	25
4.2.9 SW-Update	26
4.2.10 Transactions	26
4.2.11 Sessions	26
<b>4.3 Development View</b>	<b>26</b>
4.4 Deployment View	26
4.5 User Interface View (additional)	27
4.6 Data View (additional)	27
4.7 Variability View (additional)	28
4.8 Migration View (additional)	28
<b>5 ARCHITECTURAL QUALITIES</b>	<b>30</b>
5.1 Example Quality Attribute: Security	30
5.1.1 Security Concepts	30
5.1.2 Patch Concepts	30
5.2 <Quality Attribute #2>	30
5.3 <Quality Attribute #N>	31
<b>6 ARCHITECTURAL DECISIONS AND ALTERNATIVES</b>	<b>32</b>
6.1 <Requirement/Decision #1>	32
6.2 <Requirement/Decision #N>	32
6.3 Other Significant Decisions	32
<b>7 PRINCIPLES AND GUIDELINES</b>	<b>33</b>
<b>A APPENDIX</b>	<b>34</b>
A.1 List of open issues	34
A.2 Index	34

## Test Architecture Documentation

Driven by Architecture documentation; you must understand & review it as the TeA.

## ISO/IEC/IEEE 29119-3: Test documentation Overview

- Part 1: Concepts and Definitions
- Part 2: Test Process
- Part 3: **Test Documentation**
- Part 4: Test Techniques

### Test Documentation

- **Organizational test process**
  - Test strategy (*test levels, test goals, who does what*)
- **Test management processes**
  - Test plan
  - Test status
  - Test completion report
- (*Test Exit Criteria: test coverage, test progress, defects*)
- **Dynamic test processes**
  - Test Designs
  - Test case/specs
  - Test data
  - Test environment
  - Test execution log
  - Defects

Test Plan Template

## Example: Test Concept for <XYZ> Template

*Ingenuity for life*

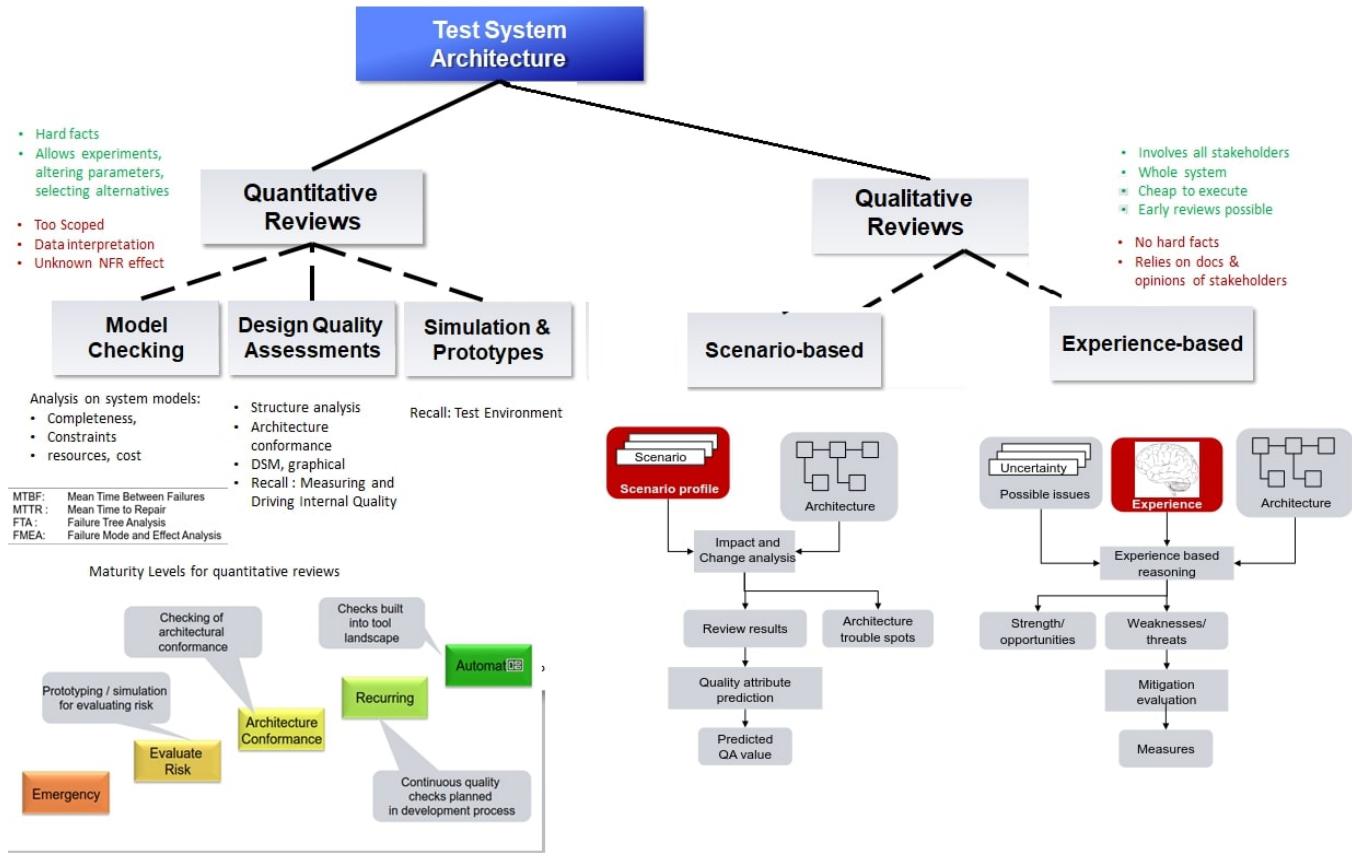
1. Scope .....	5.4 Test data .....
1.1 Test basis .....	5.5 Test deliverables and documentation .....
1.2 Open issues and topics to be clarified .....	5.6 Defect and change management .....
2. Stakeholders .....	6. Test environment .....
3. Test goals .....	6.1 Hardware / Software / Operational environment for testing ...
3.1 Goals and responsibilities of testing .....	6.2 Test configurations and setup .....
3.2 Assumptions, dependencies, and constraints .....	6.3 Deployments and releases .....
4. Test scope .....	7. Test automation .....
4.1 Test objects, test items, system under test (SUT) .....	7.1 Development environment, CI .....
4.2 Major product risks (risk-based testing) .....	7.2 Frameworks .....
4.3 Features to be tested .....	7.3 Test tools .....
4.4 Features not to be tested .....	7.4 Test scripts .....
4.5 Non-functional requirements (NFRs) to be tested .....	8. Test logistics .....
4.6 Non-functional requirements (NFRs) not to be tested ....	8.1 Schedule, timeline .....
4.7 Test data requirements .....	8.2 Staffing .....
5. Test strategy and approach .....	
5.1 Test levels, entry/exit criteria .....	
5.2 Test types .....	
5.3 Test techniques .....	

## Architecture Quality & Reviews

---

Criteria for good architecture (recall NFRs and Quality Characteristics) :

- Reliable
- Maintainable
- Scalable
- Performance
- Security



## Qualitative Review Phase

- Prep: clarify review goals / *reviewers*
- Collect: interviews with stakeholders, docs, source *reviewers & stakeholders*
- Elaborate / *reviewers*
  - SWOT analysis: Strengths, Weaknesses, Opportunities, Threats
  - Dealing with Weaknesses
  - ATAM (SWOT alternative): Architectural Tradeoff Analysis
- Consolidation : final report / *reviewers*
- Presentation to stakeholders / *reviewers*
- Workshop (optional) *reviewers & stakeholders*

## Qualitative Review Toolbox

*Ingenuity for life*

	Active design review	Industry practice	Architecture Tradeoff Analysis Method	System Architecture Analysis for the Field
Type	Experience-based, scenario-based	Experience-based	Scenario-based	Scenario-based
Intention	Improve design, find errors	SWOT analysis, identify measures	Clarify and prioritize requirements, evaluate suitability of architecture for change scenarios	Improve design, find errors, identify measures for mitigating typical lifecycle issues

## Error classification

*Ingenuity for life*

Error type	Examples
Inconsistencies	<ul style="list-style-type: none"> <li>▪ Places where the design will not function properly</li> <li>▪ Contradicting behavior</li> </ul>
Inefficiencies	<ul style="list-style-type: none"> <li>▪ Unnecessary complexity</li> <li>▪ Inefficient use of resources, channels, etc</li> </ul>
Ambiguities	<ul style="list-style-type: none"> <li>▪ Unclear requirements</li> <li>▪ Undocumented assumptions</li> <li>▪ Elements of the design specification which can be understood in different ways</li> </ul>
Inflexibility	<ul style="list-style-type: none"> <li>▪ Elements making change requests difficult</li> <li>▪ Obstacles for dealing with lifecycle issues (e.g. configuration, service, upgrade, etc.)</li> </ul>

## Comparison of qualitative reviews

	Active Design Review	Industry Practice	Architecture Tradeoff Analysis Method	System Architecture Analysis for the Field
<b>Interaction</b>	Designer, reviewer	Interviews	Workshop	Workshop
<b>Phase</b>	Detailed component / module design ready	After architecture has been designed	Architecture design complete enough for walkthroughs	Detailed component / module design ready
<b>Strength</b>	Focused on finding defects in design	Concrete measures	Bring stakeholders together, requirement prioritization	Concrete measures
<b>Key restriction</b>	Small scale	No common understanding of requirement priorities	No measures	Focus on system in operational conditions
<b>Duration</b>	2 days / reviewer	Four weeks regular 1 day flash	Two weeks	Two weeks

## Overview of qualitative architecture reviews

*Ingenuity for life*

	SAAM	ATAM	ADR	Industry practice
Type	Scenario-based	Scenario-based	Experience-based, scenario-based	Experience-based
<b>Intention</b>	Clarify and prioritize requirements, evaluate suitability of architecture for change scenarios	Clarify and prioritize requirements, find risks, sensitivity points, tradeoffs	Improve design, find errors	SWOT analysis, identify measures
<b>Interaction</b>	Workshop	Workshop	Designer, reviewer	Interviews
<b>Phase</b>	Architecture design complete enough for walkthroughs	Architecture design complete enough for walkthroughs	Detailed component / module design ready	After architecture has been designed
<b>Strength</b>	Bring stakeholders together, requirement prioritization	Like SAAM, but deeper architectural evaluation	Focused on finding defects in design	Concrete measures
<b>Key restriction</b>	No risks, no measures	No measures	Small scale	No common understanding of requirement priorities
<b>Duration</b>	2–3 days	Two weeks	2 days / reviewer	Four weeks regular 1 day flash

# TESTING & QUALITY

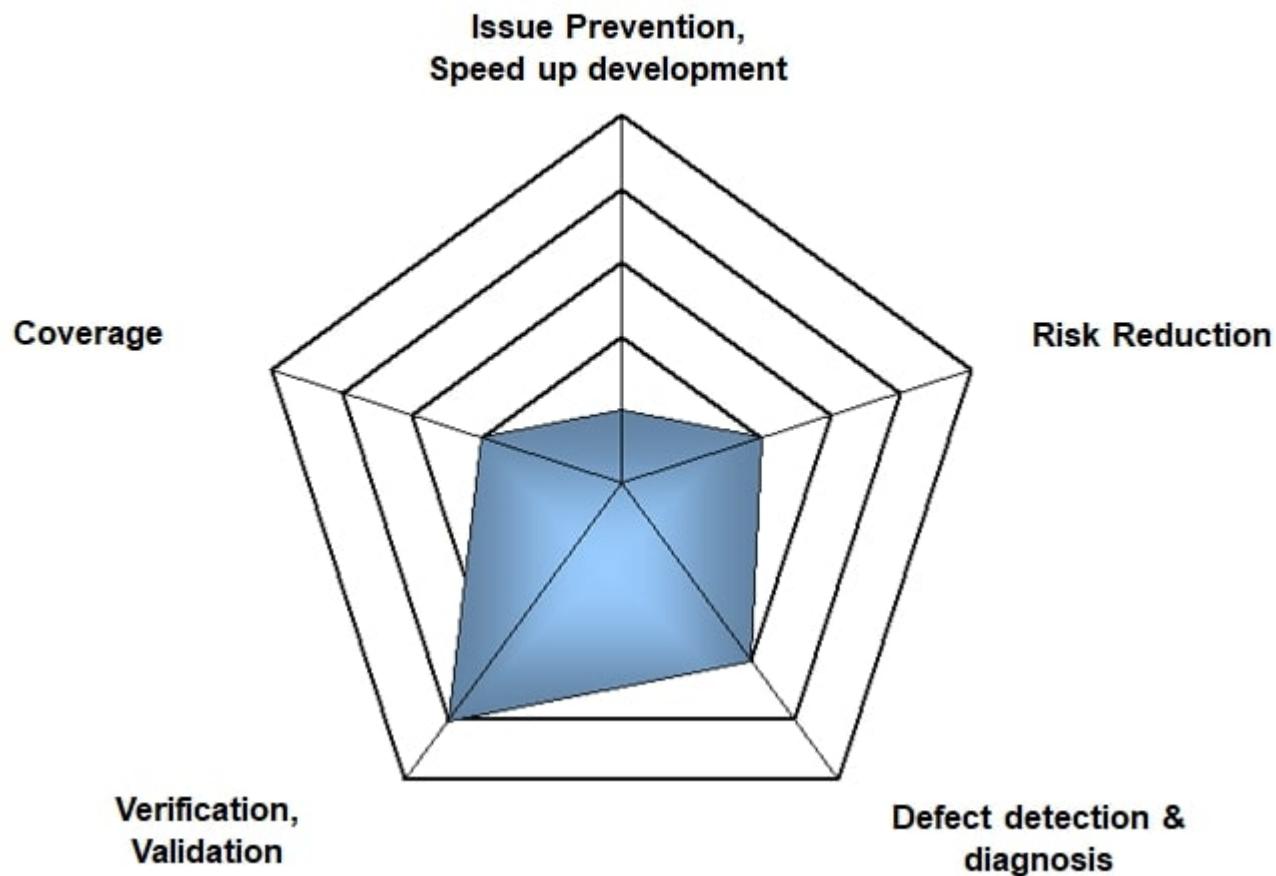
## Value of Testing

What is Testing? : investigation of the SUT to provide **information** that results in **improvements**.

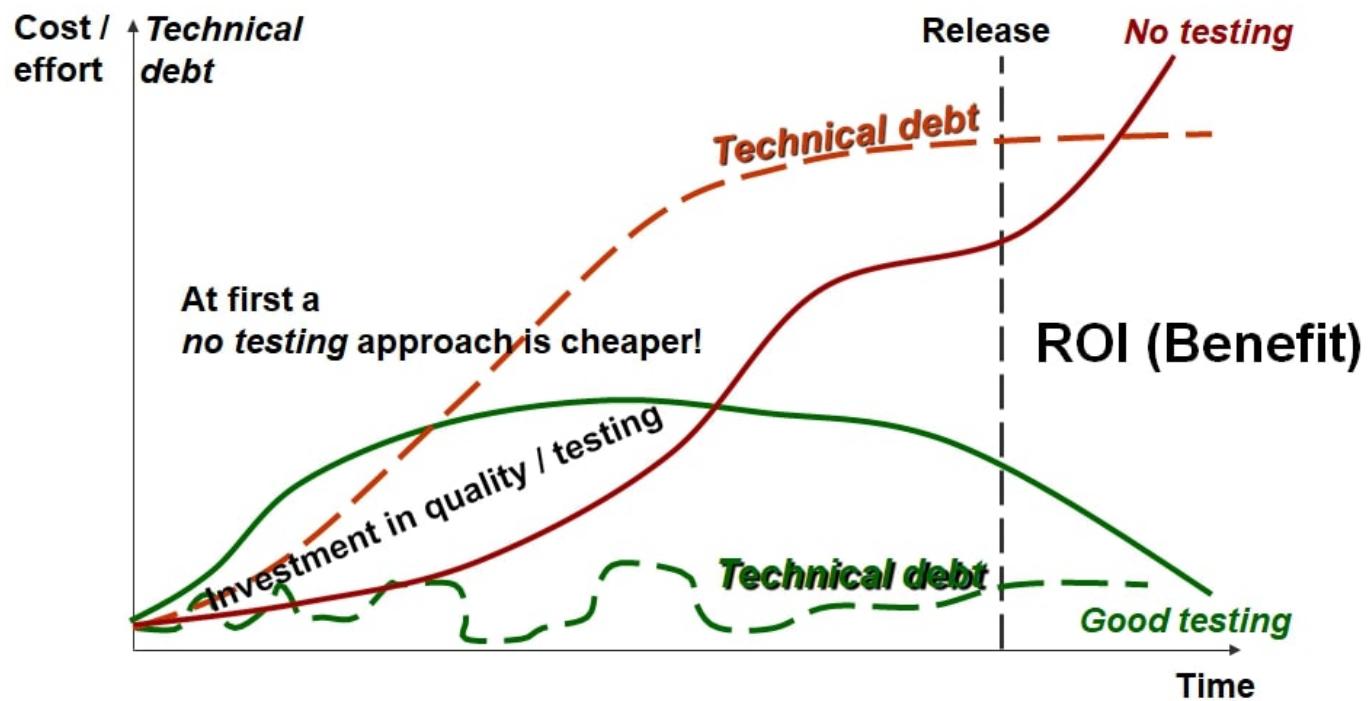
The 5 Dimensions of Testing

Coverage: an assessment for the thoroughness or completeness of testing with respect to our test model - *Paul Gerrard*

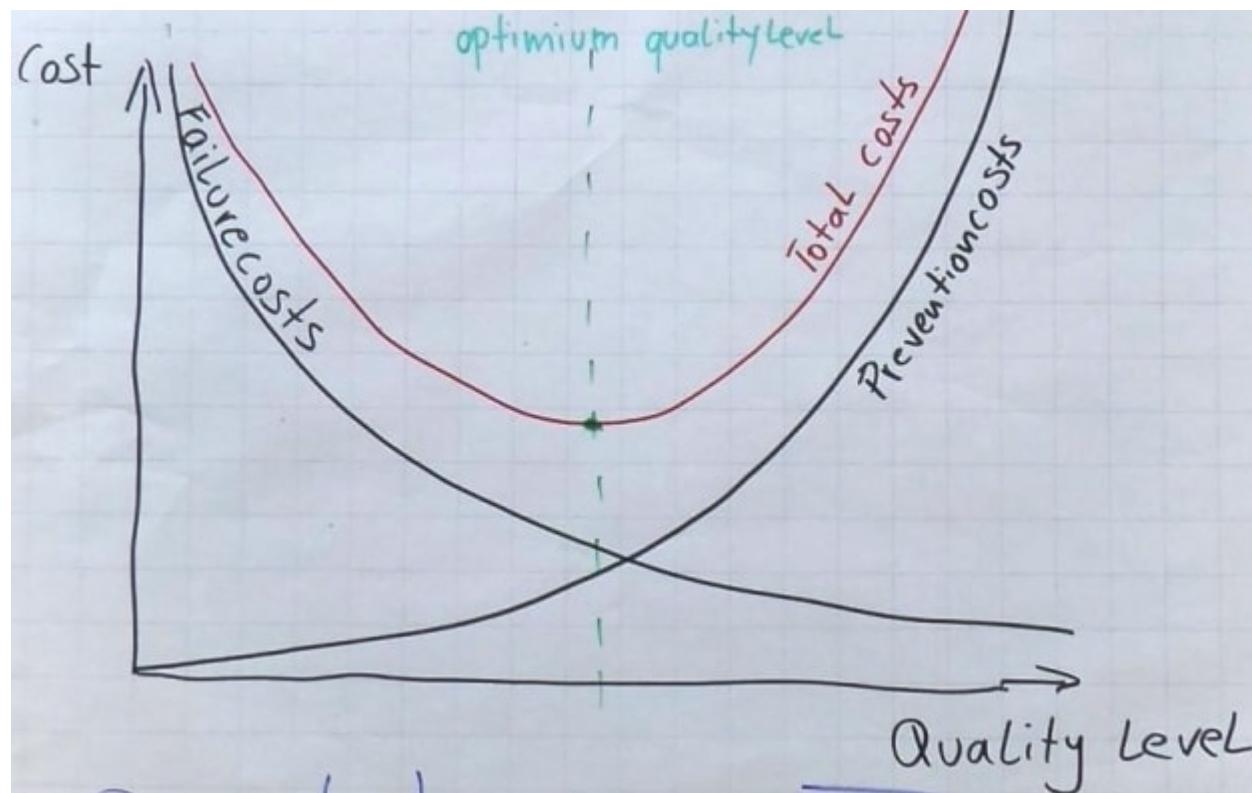
## 5 dimensions of testing



ROI of Testing

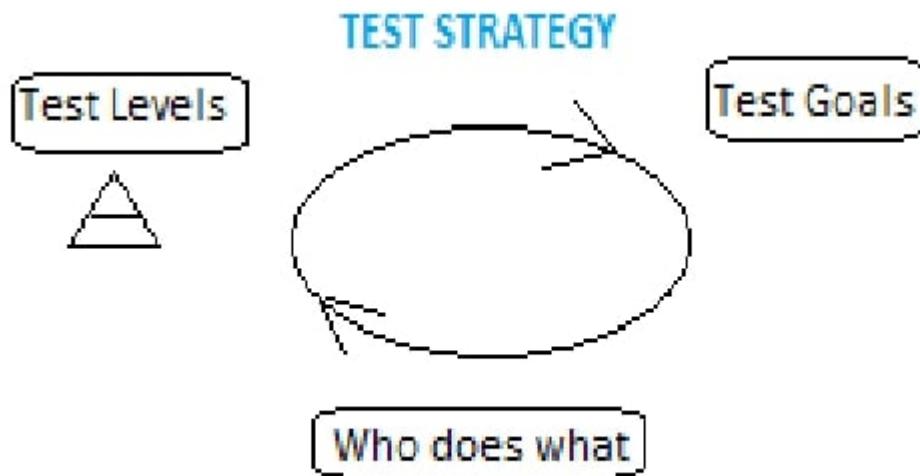


## Cost of Quality

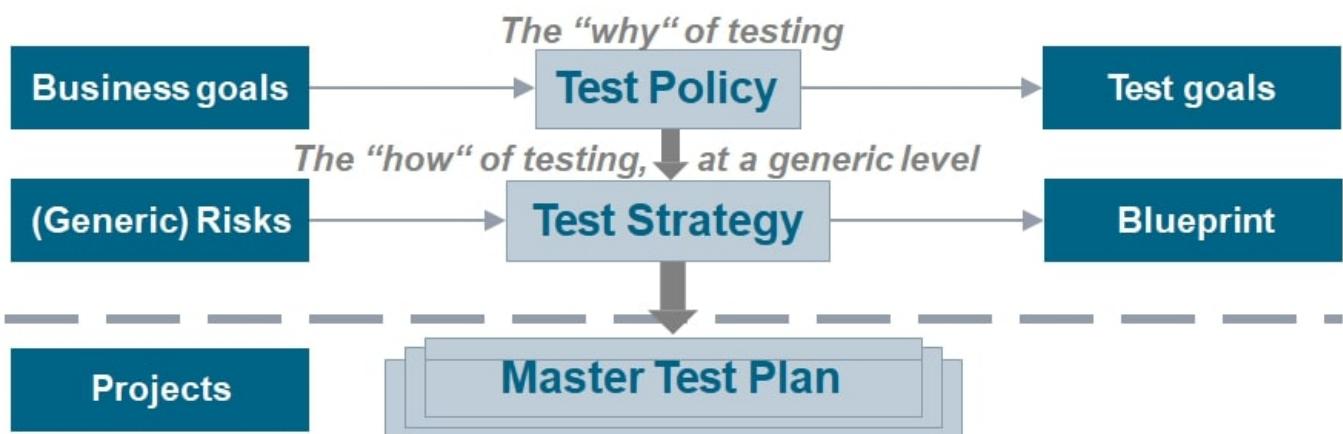


## Test Strategy

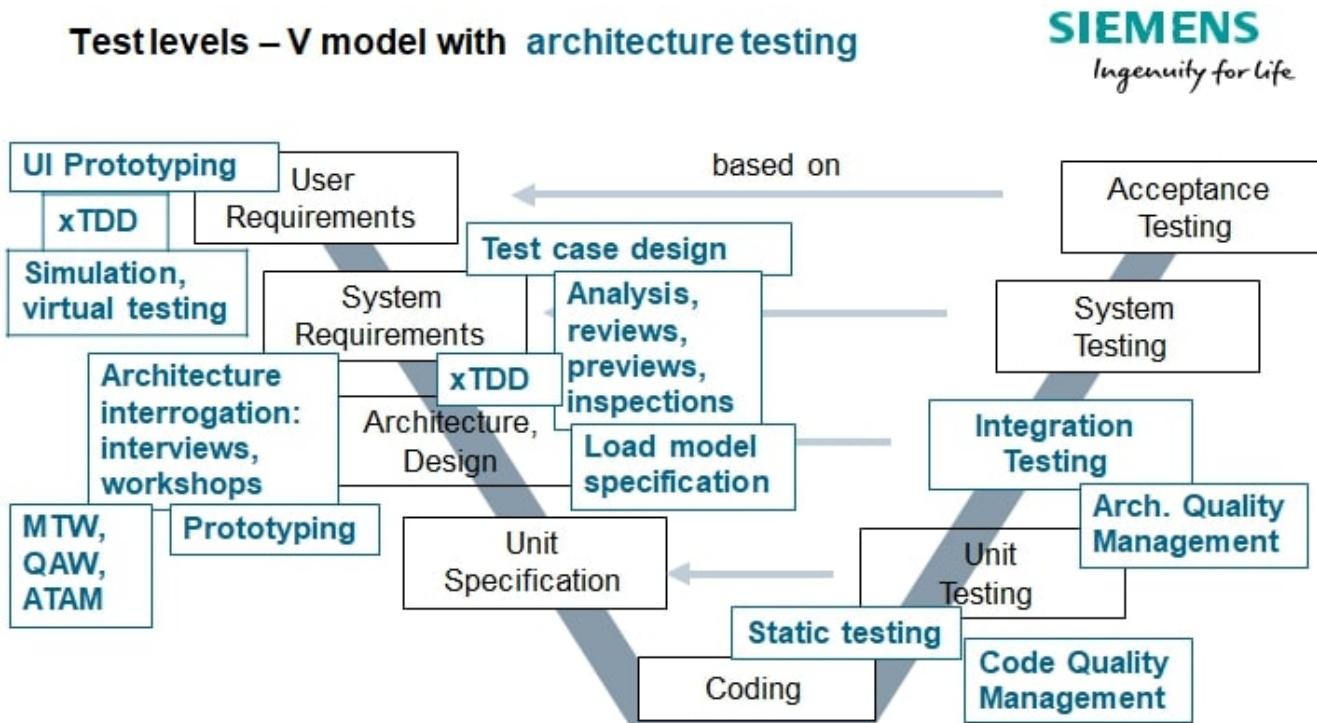
---



Testing serves a purpose (*test mission*) that has goals (*test policy*) and requires a map (*test strategy*).



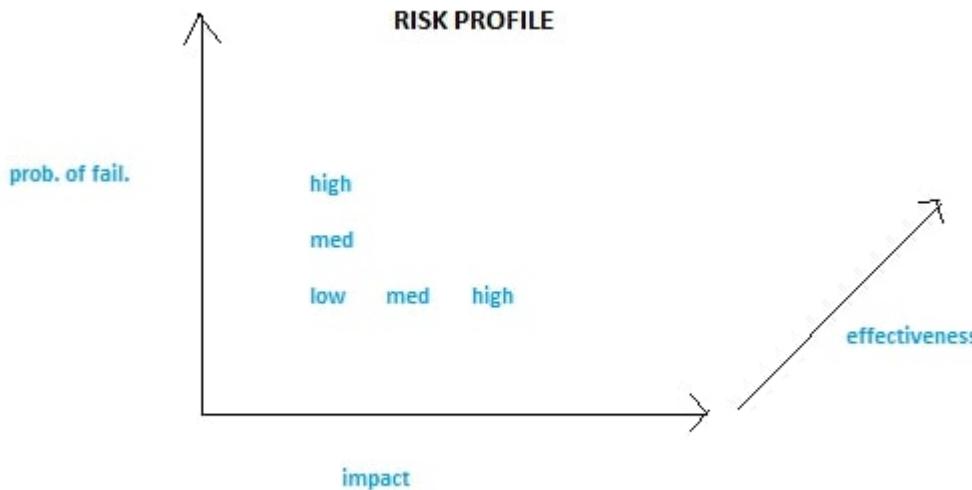
Test levels – V model with architecture testing



# Risk Based Testing

---

## Risk Profile



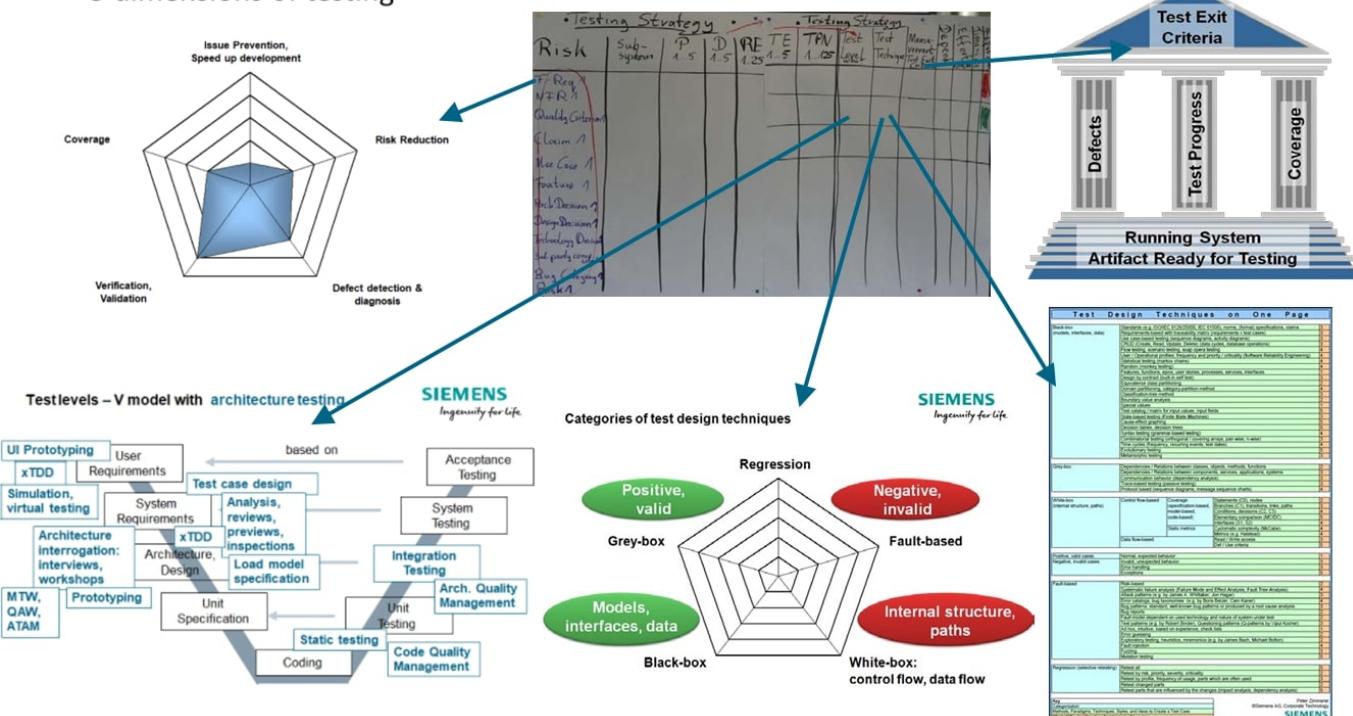
## Risk Based Testing Worksheet. You can download the worksheet in xls .

Description	Identifier	Risk (If any) (Model)	Objectives/Benefits Threatened	Subsystem	Probability	Damage (Consequence & Cost)	Exposure	Test Effectiveness	Test Priority Number	Test Objective(s)	Test Level	Test Technique	Measurement	Dependence	Effort	Timescale	Reporting
Score, Ranges, and Examples		For example: Quality criteria	Scores from 1 to 5: 1 Very low importance = 5 Highest importance		Scores from 1 to 5: 1 Low probability, chances are slight 2-3-4=High probability 5 Very likely, almost certain	What is the damage (consequence & cost) of this mode of failure? - Consequence & cost for business - Consequence & cost for test - Consequence & cost for usage	Risk exposure, that is product of probability and consequence (Cost)	How confident are the testers that they can address this risk? - Test effectiveness	Product of Probability and Consequence (Cost)	In which test level is this testing required to prove that the objective(s) have been met? - Exposure and Test Effectiveness	What method or technique is to be used in testing?	What method or technique is to be used in testing? - Exposure and Test Effectiveness	What do the testers assume or depend on to do the testing? - Test effectiveness	How much effort is required to do the testing? - Test effectiveness	How much elapsed time is required to do the testing? - Test effectiveness	Reporting	
1	Functional requirement																
2	Non-functional requirement (NFR)																
3	Quality criterion																
4	Claim																
5	User story																
6	Feature																
7	Function																
8	Epic																
9	User story																
10	Process																
11	Service																
12	API																
13	Architectural decision																
14	Design decision																
15	Technology selection																
16	Business domain selection (frameworks, open source, external partners)																
17	Core asset in PLE																
18	Open variant space in system ecosystem																
19	Buy category																
20	Risk																
21																	

## Relations in RBT worksheet

### Building blocks of Risk-based Test Strategy

#### 5 dimensions of testing



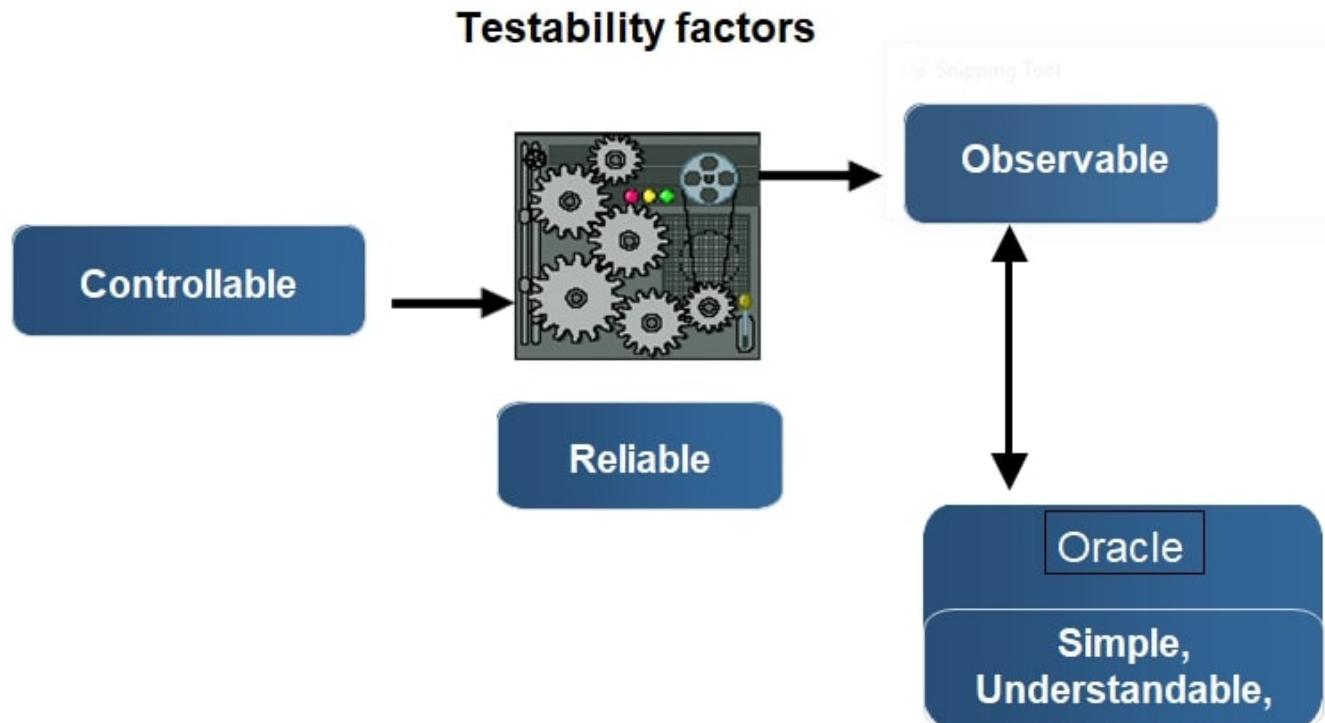
## Design for Testability

Goal : Controllable, Observable, Reliable : *Intrinsic Testability* . More On Heuristics of Testability

Why : reduce the cost of testing, diagnosis, maintenance.

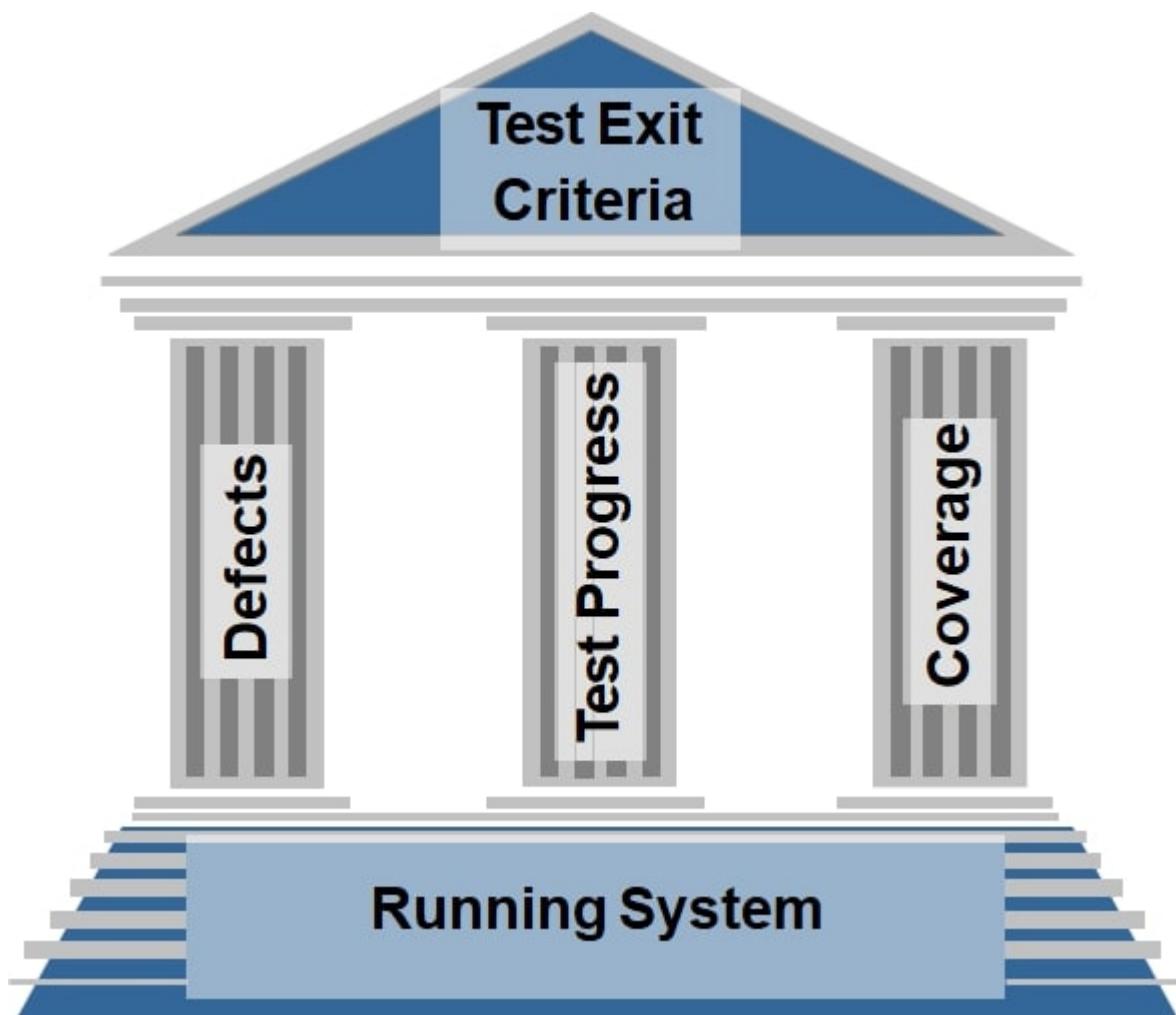
Who : system, software and test architects

How : TDD, Loose Coupling, Inversion of control, SOLID, follow the best practices of [clean code & architecture](#).



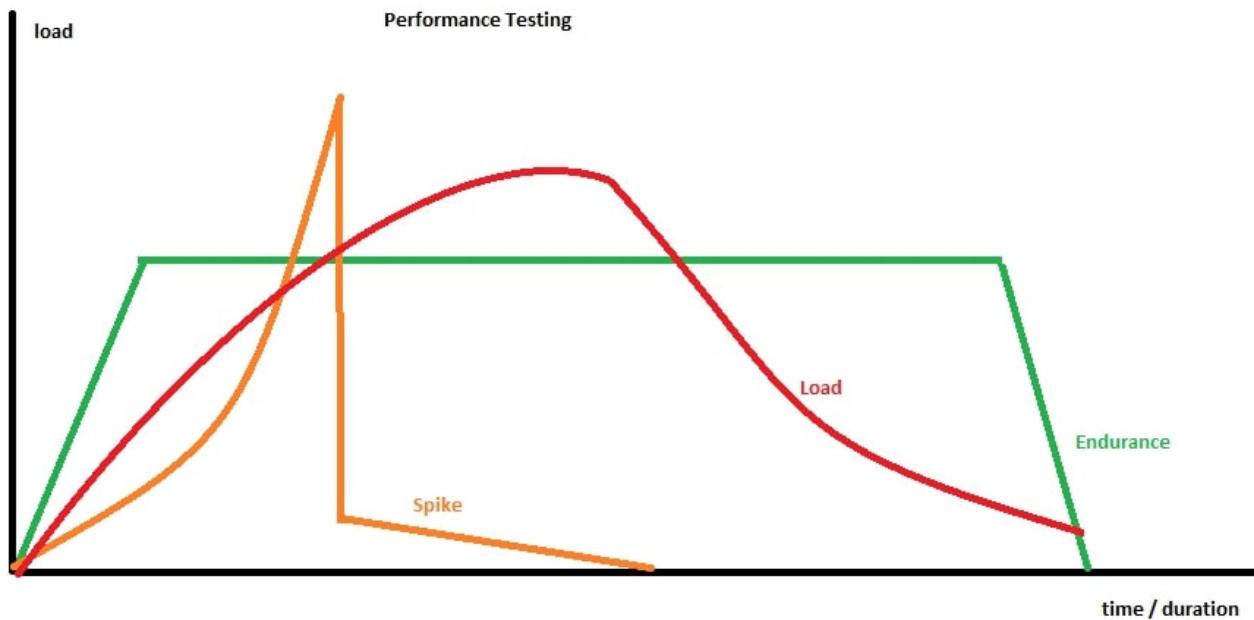
## Test Exit Criteria

---

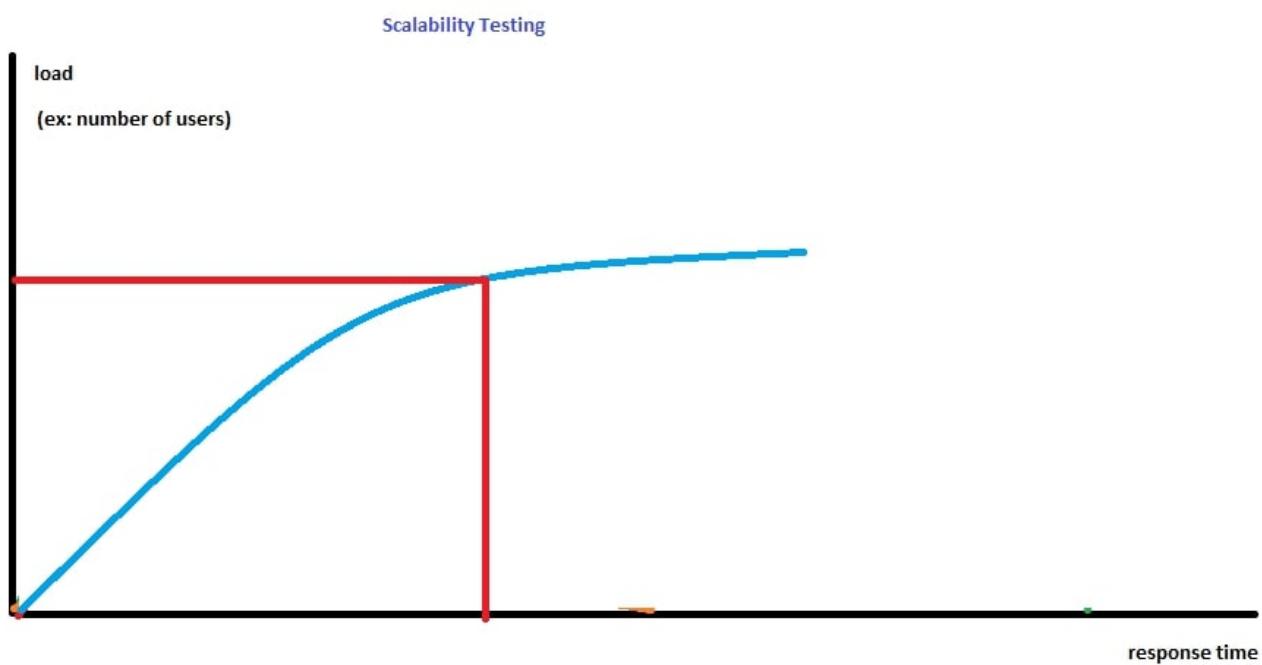


# Performance Testing and Scalability

---



At a certain load, the response time sky-rocks.



## TDD

---

From Req. to unit test level. The most effective way of specifying something is to describe how you would test it.

# Test Design Techniques

---

## [Test Design Techniques pdf](#)

- **Black-box:** req. based, workflow, statistical/markov, eq.class & boundary value, state-based, combinatorial, model based
- **Gray-box:** interfaces between components, services, systems
- **White-box:** statement, branch, path: cyclomatic complexity ( $Edges-Nodes-2 = independent\ paths$ )
- **Fault-based:** exploratory, fuzzing, mutation. [Data Type Attacks and Web Tests pdf](#)
- **Regression:** Risk Based Testing, testing firewall (re-test parts influenced by changes)

Test Design Techniques on One Page				
Black-box (models, interfaces, data)	Standards (e.g. ISO/IEC 9126/25000, IEC 61508), norms, (formal) specifications, claims	3		
	Requirements-based with traceability matrix (requirements x test cases)	3		
	Use case-based testing (sequence diagrams, activity diagrams)	3		
	CRUD (Create, Read, Update, Delete) (data cycles, database operations)	3		
	Flow testing, scenario testing, soap opera testing	4		
	User / Operational profiles: frequency and priority / criticality (Software Reliability Engineering)	4		
	Statistical testing (markov chains)	4		
	Random (monkey testing)	4		
	Features, functions, epics, user stories, processes, services, interfaces	1		
	Design by contract (built-in self test)	3		
	Equivalence class partitioning	2		
	Domain partitioning, category-partition method	4		
	Classification-tree method	3		
	Boundary value analysis	2		
	Special values	1		
	Test catalog / matrix for input values, input fields	5		
	State-based testing (Finite State Machines)	3		
	Cause-effect graphing	5		
	Decision tables, decision trees	5		
	Syntax testing (grammar-based testing)	4		
	Combinatorial testing (orthogonal / covering arrays, pair-wise, n-wise)	3		
	Time cycles (frequency, recurring events, test dates)	4		
	Evolutionary testing	5		
	Metamorphic testing	3		
Grey-box	Dependencies / Relations between classes, objects, methods, functions	2		
	Dependencies / Relations between components, services, applications, systems	3		
	Communication behavior (dependency analysis)	3		
	Trace-based testing (passive testing)	3		
	Protocol based (sequence diagrams, message sequence charts)	4		
White-box (internal structure, paths)	Control flow-based	Coverage (specification-based, model-based, code-based)	Statements (C0), nodes	2
			Branches (C1), transitions, links, paths	3
			Conditions, decisions (C2, C3)	4
			Elementary comparison (MC/DC)	5
			Interfaces (S1, S2)	4
		Static metrics	Cyclomatic complexity (McCabe)	4
			Metrics (e.g. Halstead)	4
	Data flow-based		Read / Write access	3
			Def / Use criteria	5
Positive, valid cases	Normal, expected behavior			1
Negative, invalid cases	Invalid, unexpected behavior			3
	Error handling			3
	Exceptions			5
Fault-based	Risk-based			2
	Systematic failure analysis (Failure Mode and Effect Analysis, Fault Tree Analysis)			4
	Attack patterns (e.g. by James A. Whittaker, Jon Hagar)			3
	Error catalogs, bug taxonomies (e.g. by Boris Beizer, Cem Kaner)			4
	Bug patterns: standard, well-known bug patterns or produced by a root cause analysis			3
	Bug reports			2
	Fault model dependent on used technology and nature of system under test			2
	Test patterns (e.g. by Robert Binder), Questioning patterns (Q-patterns by Vipul Kocher)			3
	Ad hoc, intuitive, based on experience, check lists			1
	Error guessing			2
	Exploratory testing, heuristics, mnemonics (e.g. by James Bach, Michael Bolton)			2
	Fault injection			4
	Fuzzing			3
	Mutation testing			5
Regression (selective retesting)	Retest all			5
	Retest by risk, priority, severity, criticality			2
	Retest by profile, frequency of usage, parts which are often used			3
	Retest changed parts			2
	Retest parts that are influenced by the changes (impact analysis, dependency analysis)			5

## Test Automation Patterns website

[Test Automation Design Patterns paper](#)

## Test Environment

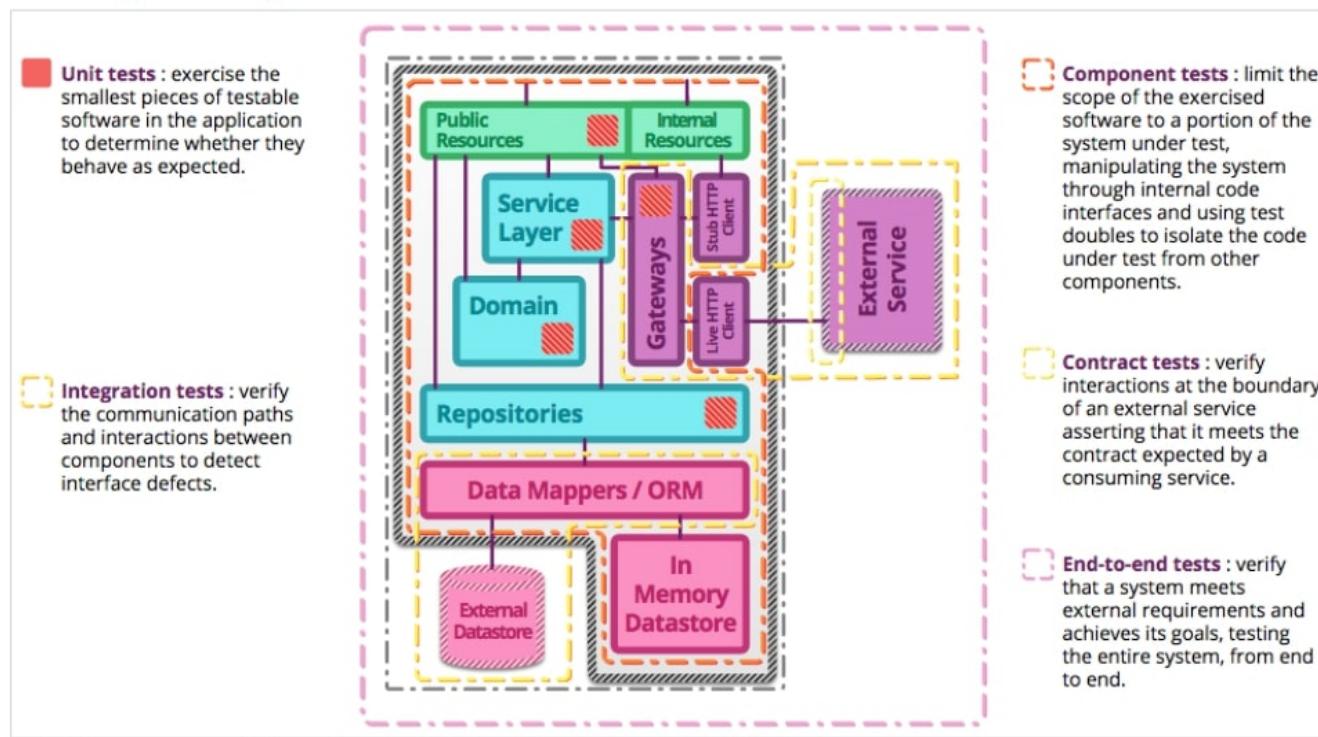
**Test environment:** test rig

**Test infrastructure:** test rig + tools + office network etc.

**Test suite architecture:** test levels

## Testing strategies in a microservice architecture

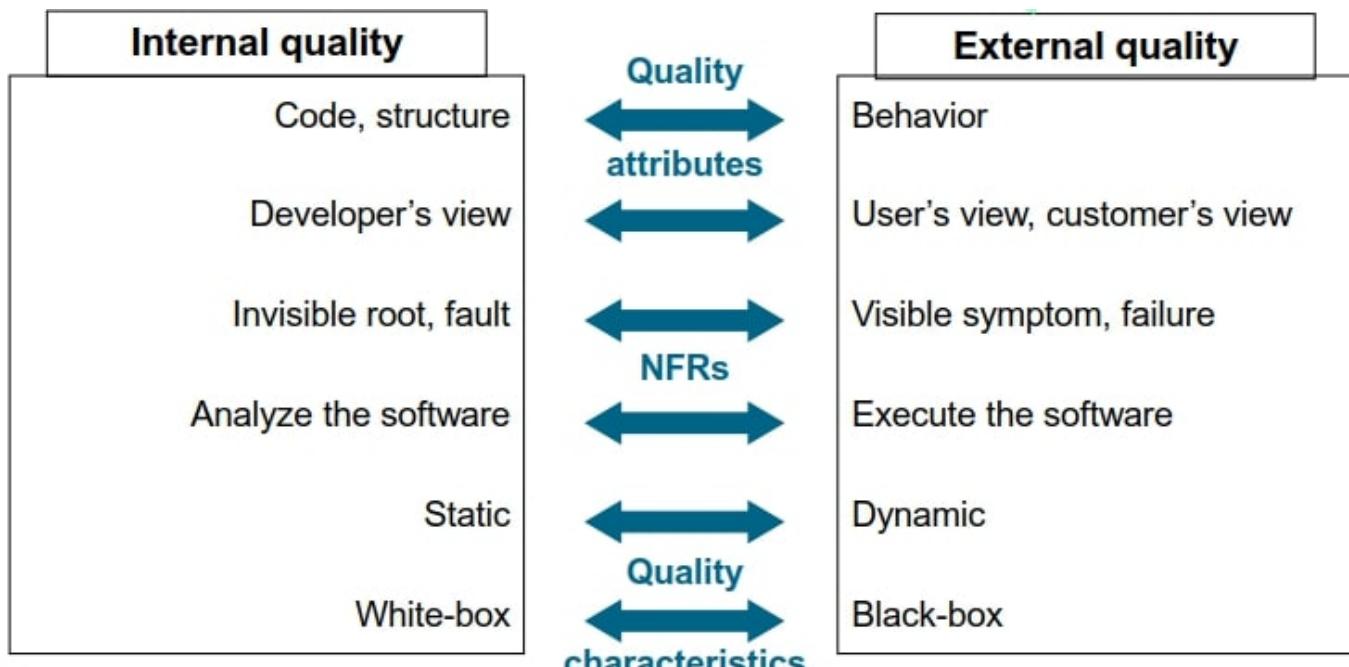
Ingenuity for life



## Internal Quality

Negative effects:

- Slows development with unplanned activities
- Rising cost of maintenance, new features, change
- Rising cost of regression testing, system testing for hotfixes
- Rising cost of onboarding
- Complex & risky integration

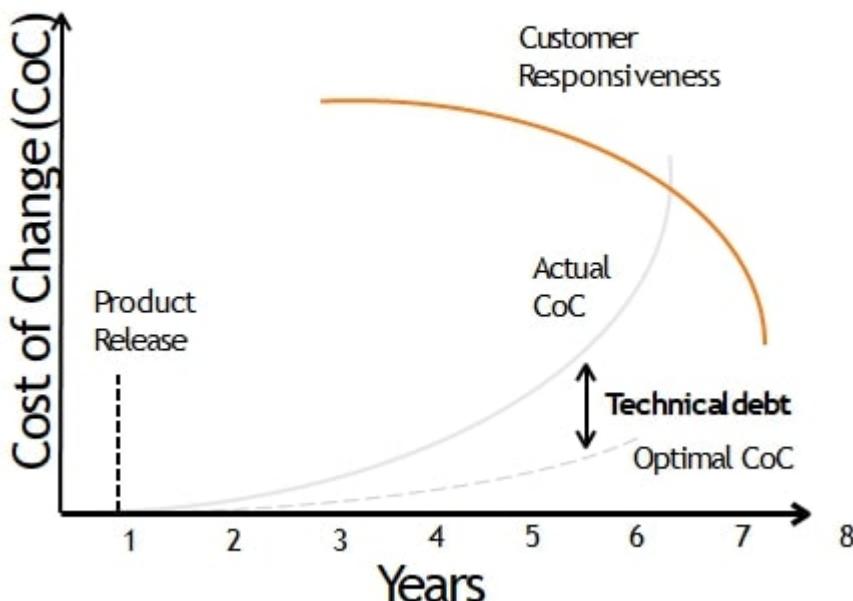


**Internal metrics** measure the software itself, **external metrics** measure the behavior of the computer-based system that includes the software. (ISO/IEC 9126-1)

## Technical Debt

Lack of internal quality results in technical debt.

## What is technical debt?



## Measuring and Driving Internal Quality

To measure internal quality

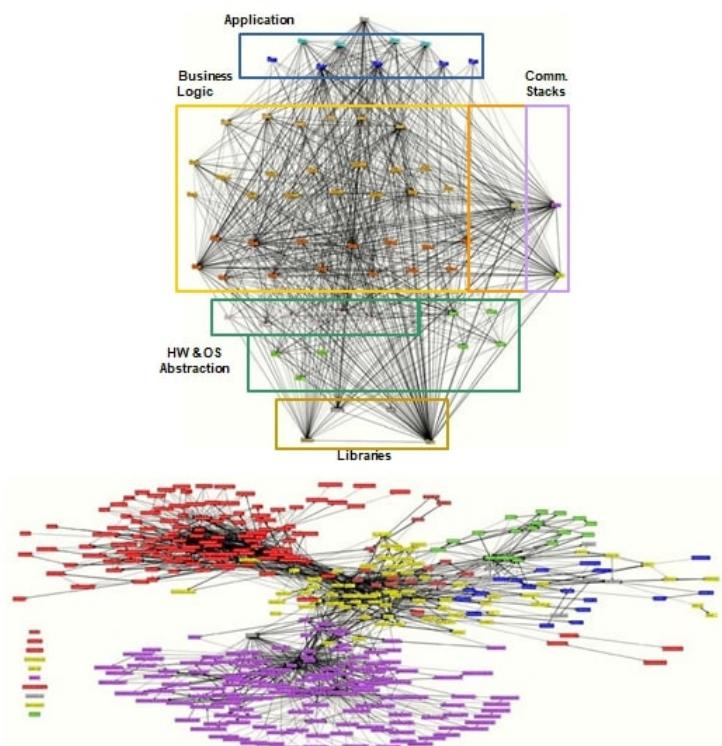
- Static code analysis, linters etc.
- Req. trace

- On-boarding feedback
- Visualize with tools, reviews
- Test gap analysis
- Automated document analysis

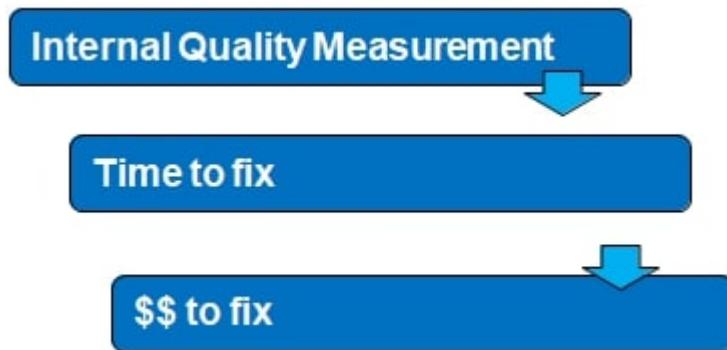
## Visualizing internal quality – Architecture Dependency structure matrix (DSM) & dependency graphs

**SIEMENS** Ingenuity for life

	.1	.2	.3	.4	.5	.6	.7	.8	.9	.10	.11	.12	.13	.14	.15	.16
1	.	3	263	125	164	45	.	.	.	.	142	128	.	.	.	.
2	42	.	259	7	342	74	335	172	75	69	.	4	213	1492	.	.
3	452	2172	.	397	75	215	1293	47	18	.	149	1318	955	.	.	.
4	605	1274	437	.	557	2157	1329	1224	.	.	63	784	1715	.	.	.
5	826	3427	062	1812	.	3100	436	1514	13	.	308	2136	2908	.	.	.
6	6071	748	6302	2882	543	.	18584	984	1113	1999	7917	6	2008	2501	3	.
7	228	1085	316	969	81	707	.	900	1	5	1841	156	70	.	.	.
8	1299	18	487	690	592	954	691	.	997	131	250	2279	1453	.	.	.
9	305	1150	488	665	840	1573	1914	1071	.	156	600	2	73	244	.	.
10	44	53	30	23	63	176	91	6	12	.	9	175	.	2	.	.
11	33	40	40	75	92	.	52	66	.	795	.	210	.	.	.	.
12	521	216	678	866	237	443	498	1191	167	680	.	.	8	48	.	.
13	317	22	1892	1448	1524	7334	1620	1126	8	.	68	.	662	760	.	.
14	11	11	28	395	.	84	.	89	6	.	2	.	1521	.	.	.
15	7845	842	8088	339	774	0056	9078	4542	5951	6391080	0110	892	7694	.	27	.
16	388	151	151	2515	1163	3543	1051	1523	1	.	69	.	177	507	.	.



To drive internal quality, you must monetize it:



## Test Code & Architecture Quality Management

Test Code Quality at different levels:

Micro	code	tools
Macro	hacky code	review
Architecture	UML	review, some tools for architecture analysis

## Software Test Code Engineering (STCE)

### ***End-to-end test script engineering and test script management***

- Use (test) patterns as guidelines to ensure quality
- Functional-quality attributes of test code
  - Correctness in properly testing the SUT
  - Effectiveness in fault detection (→ assess and verify test suite quality)
    - If the test case fails, does the SUT really have a fault?
    - If the SUT has a fault, does the test suite detect it?
- Nonfunctional-quality attributes of test code
  - Maintainability, understandability, readability
  - Reliability
  - Test smells, e.g. test redundancy
- Comaintenance
  - Test antipatterns
  - Determine test case sensitivity
  - Minimize coupling with SUT

