REQUIREMENTS ENGINEERING

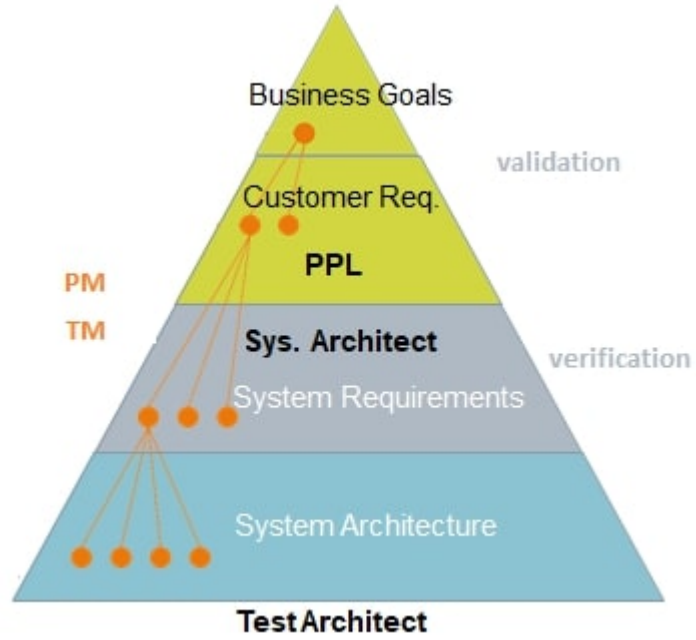
Sentence Template for System Requirements

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

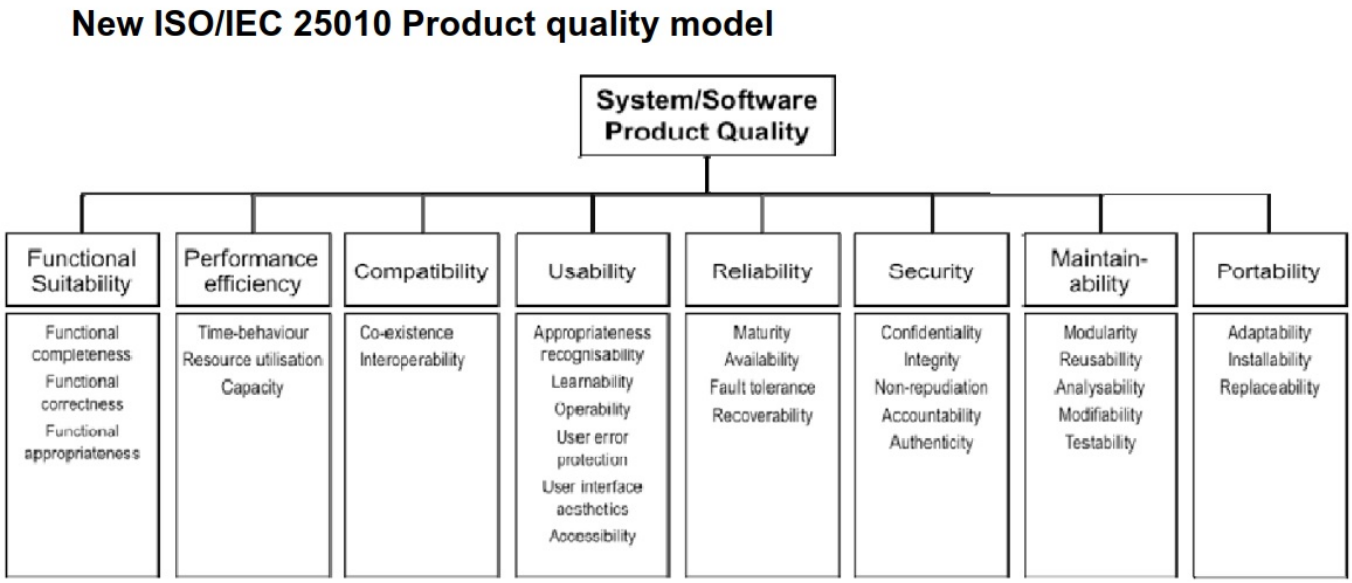


Tracing Usage:

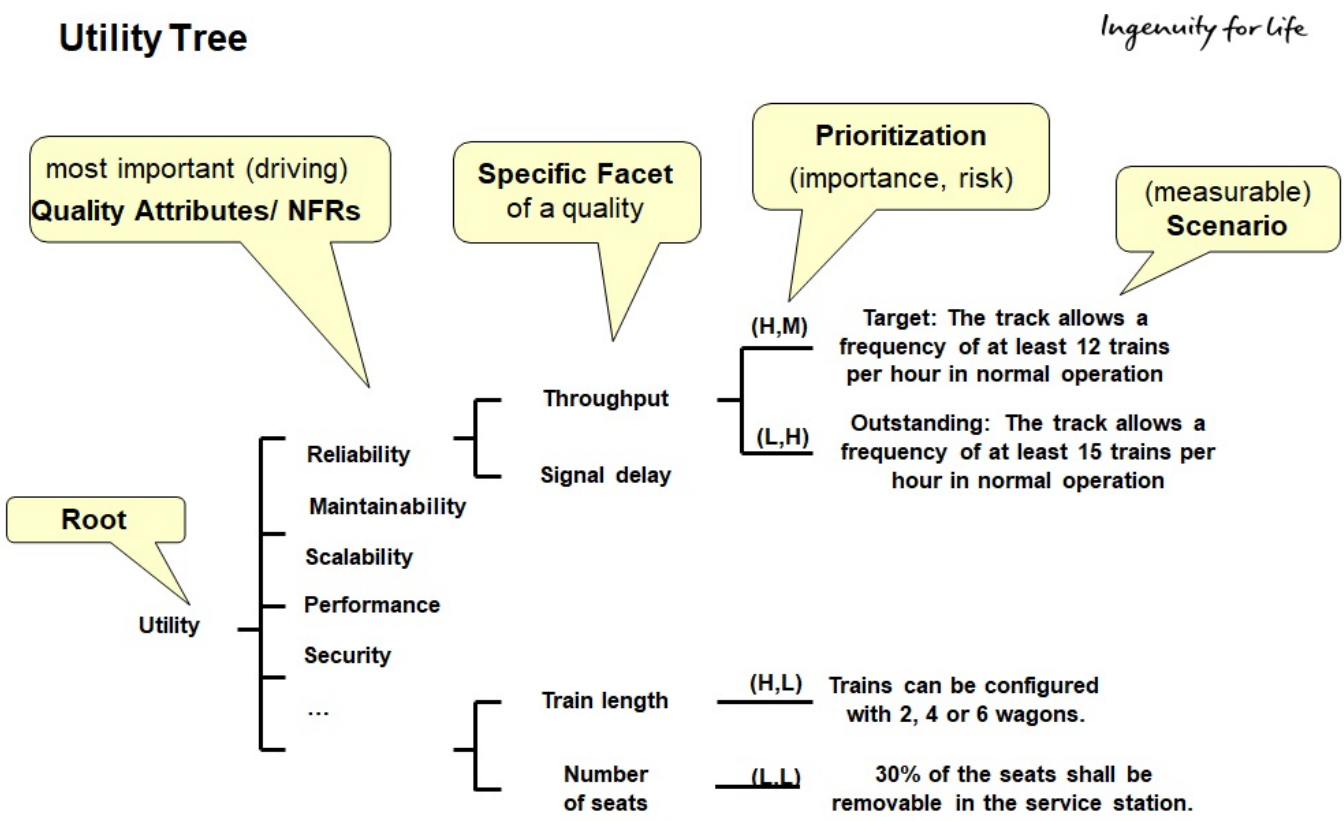
Analysis	Description	Process
Derivation	*Why is this here?*	cost-benefit analysis
Impact	*What if this changed?*	change management
Coverage	*Have I covered all reqs?*	management report

NFRs / Quality Attributes

Software Quality Characteristics pdf

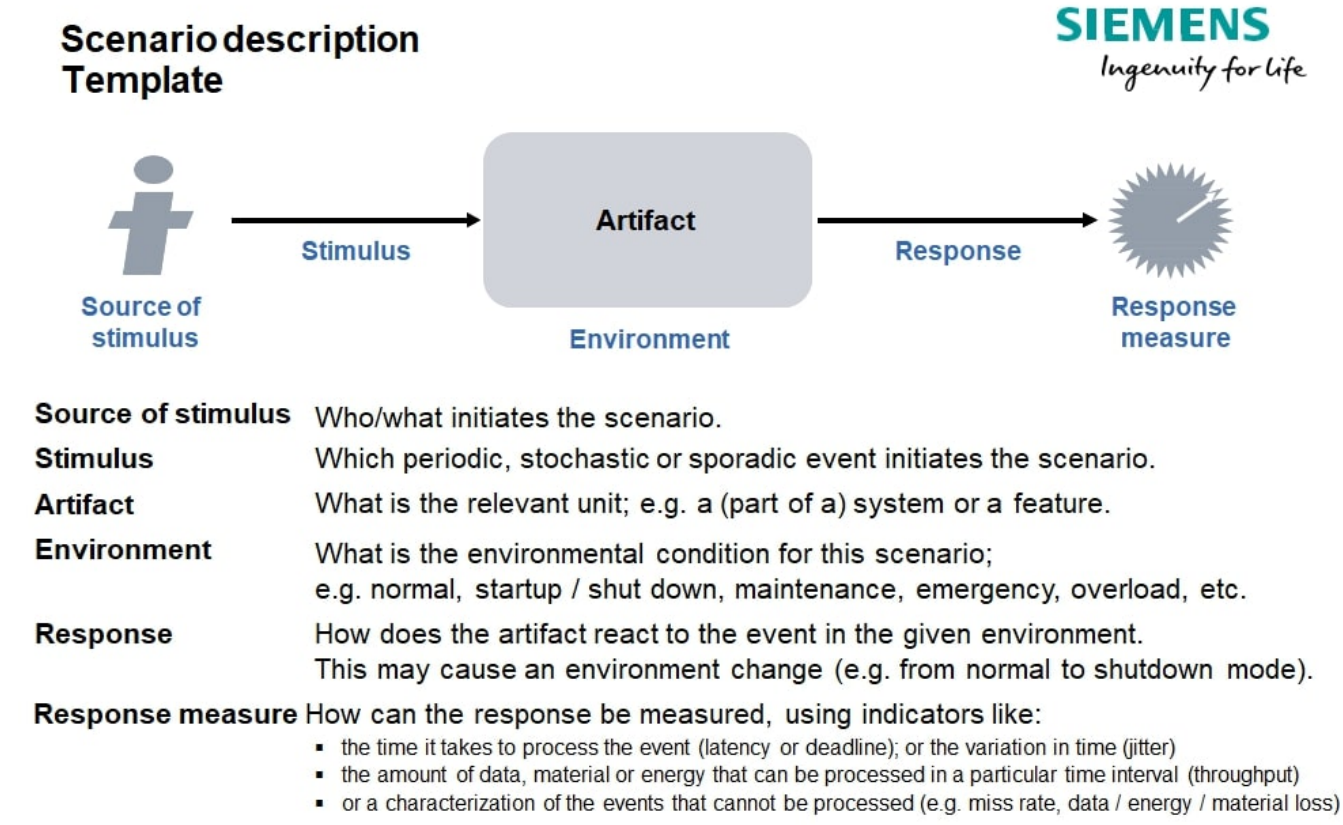


Step 1: Utility Tree



Step 2: Scenario Description Template / Workflows

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



Step 3: Refine Scenarios

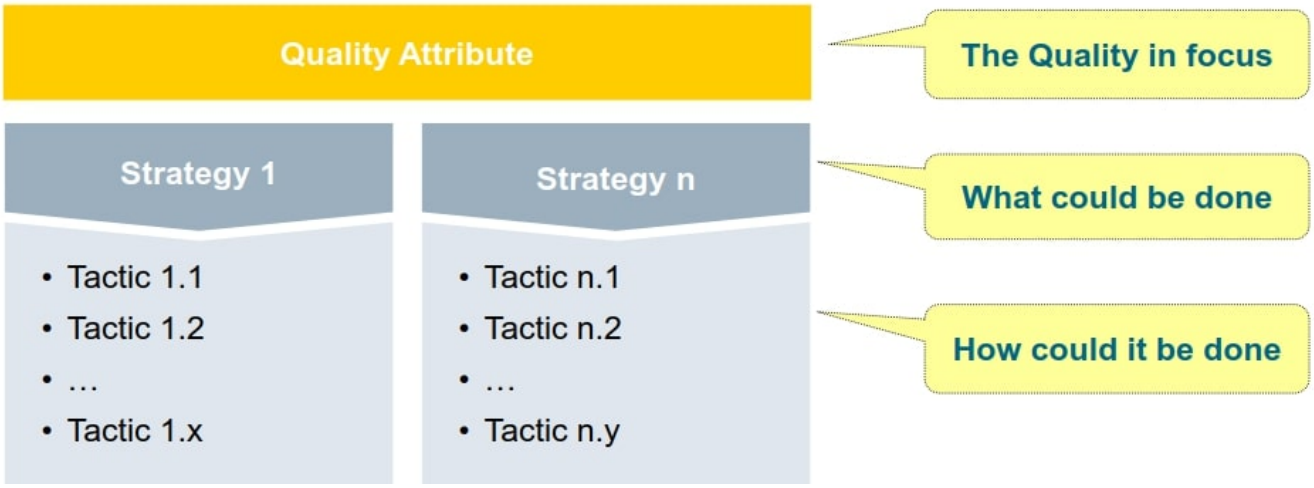
Scenario refinement table
Example from SEI Quality Attribute Workshop (QAW)

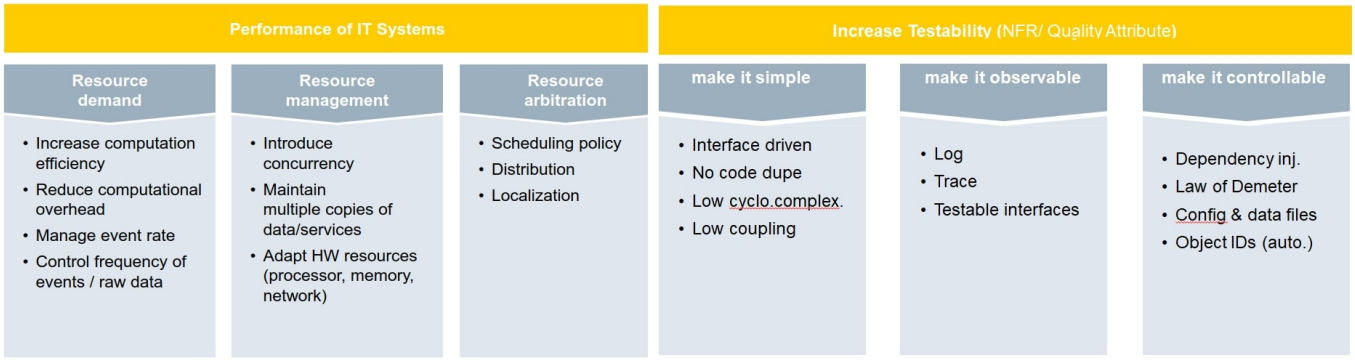
Scenario Refinement for Scenario N		
Scenario(s):		When a garage door opener senses an object in the door's path, it stops the door in less than one millisecond.
Business Goals:		safest system; feature-rich product
Relevant Quality Attributes:		safety, performance
Scenario Components	**Stimulus:**	An 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

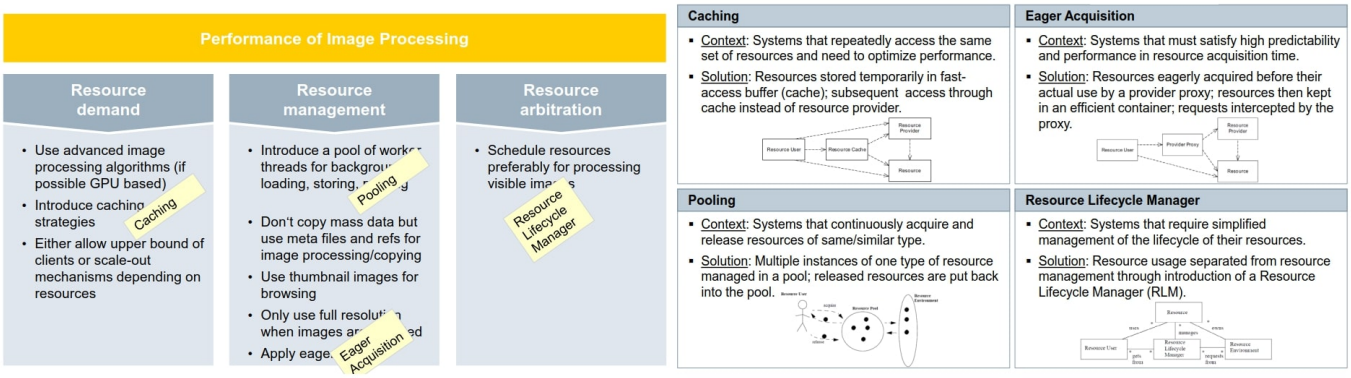
.. and use Design patterns when needed...

Design Strategy Template for Quality Attributes

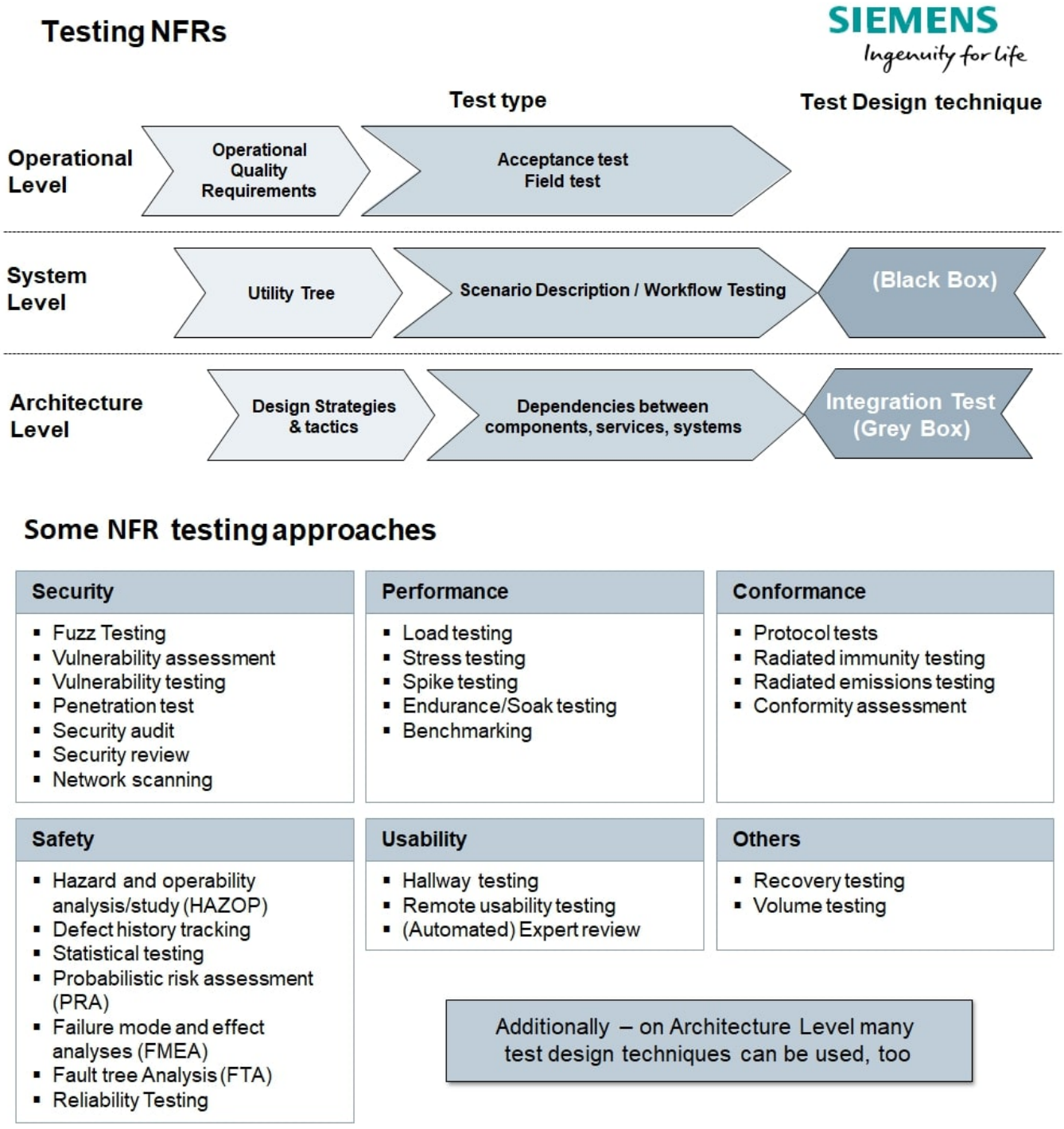




Design Pattern example
"Performance of Image Processing"



Testing NFRs



Some NFR testing approaches

Security
- Fuzz Testing
- Vulnerability assessment
- Vulnerability testing
- Penetration test
- Security audit
- Security review
- Network scanning

Performance
- Load testing
- Stress testing
- Spike testing
- Endurance/Soak testing
- Benchmarking

Conformance
- Protocol tests
- Radiated immunity testing
- Radiated emissions testing
- Conformity assessment

Safety
- Hazard and operability analysis/study (HAZOP)
- Defect history tracking
- Statistical testing
- Probabilistic risk assessment (PRA)
- Failure mode and effect analyses (FMEA)
- Fault tree Analysis (FTA)
- Reliability Testing

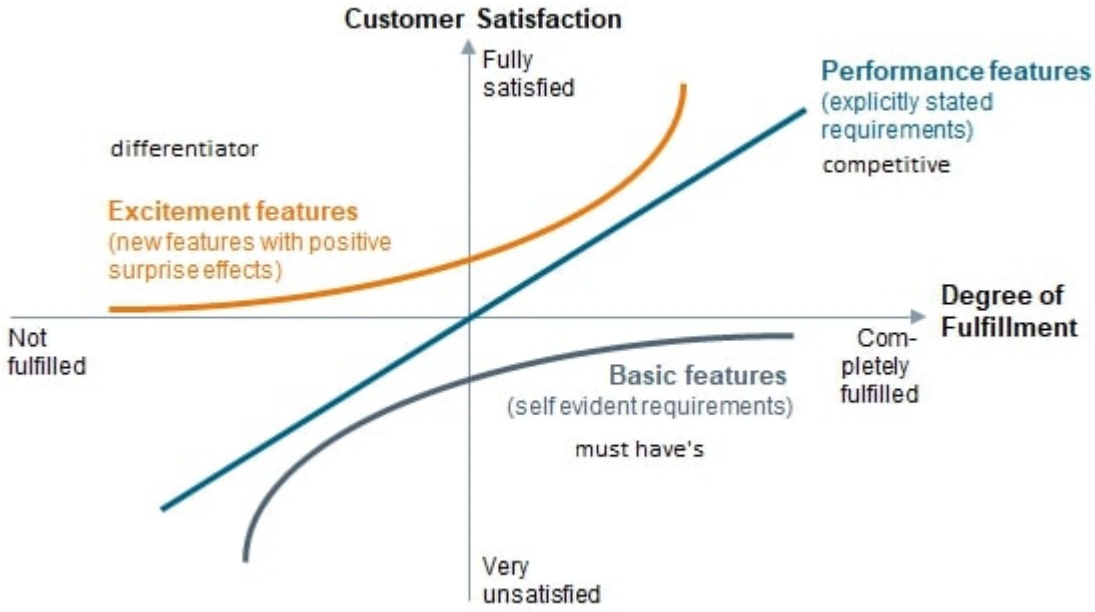
Usability
- Hallway testing
- Remote usability testing
- (Automated) Expert review

Others
- Recovery testing
- Volume testing

Additionally – on Architecture Level many test design techniques can be used, too

KANO Model

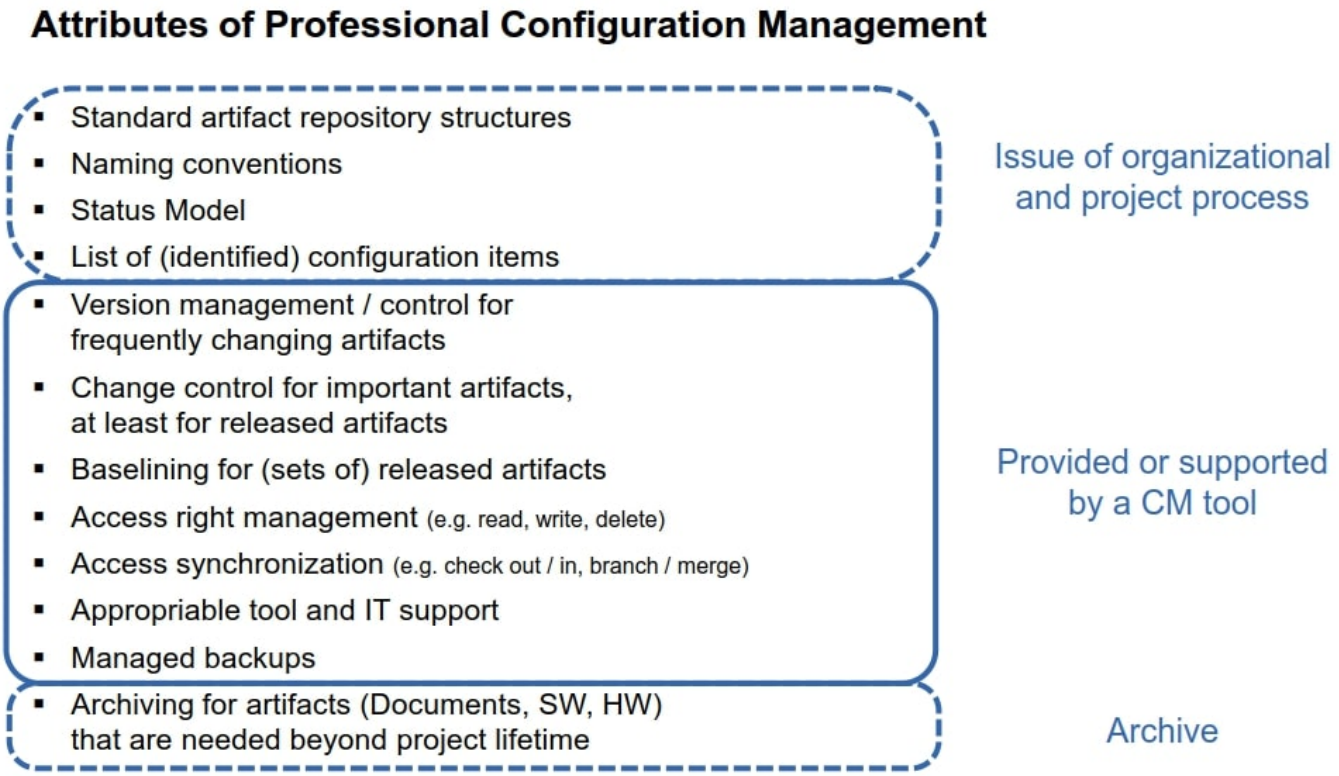
For the Product Manager KANO is a tool to find the right mix between the different feature types. For the Test Architect KANO helps to identify risks.



Configuration Management

Source Control, configurations, environment...

Devops: continious integration, deployment, testing...



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

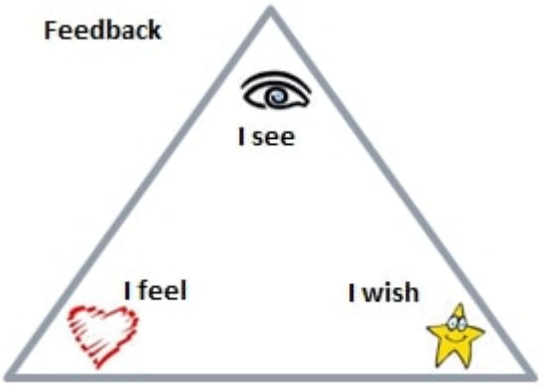
- Rough initial evaluation by the CCB to decide on further action
- Deep analysis by experts
  - on architecture, cost, schedule
  - all affected disciplines
  - all relevant sub-systems
- Propose further proceeding (accept / defer / reject)

SOCIAL CAPABILITY

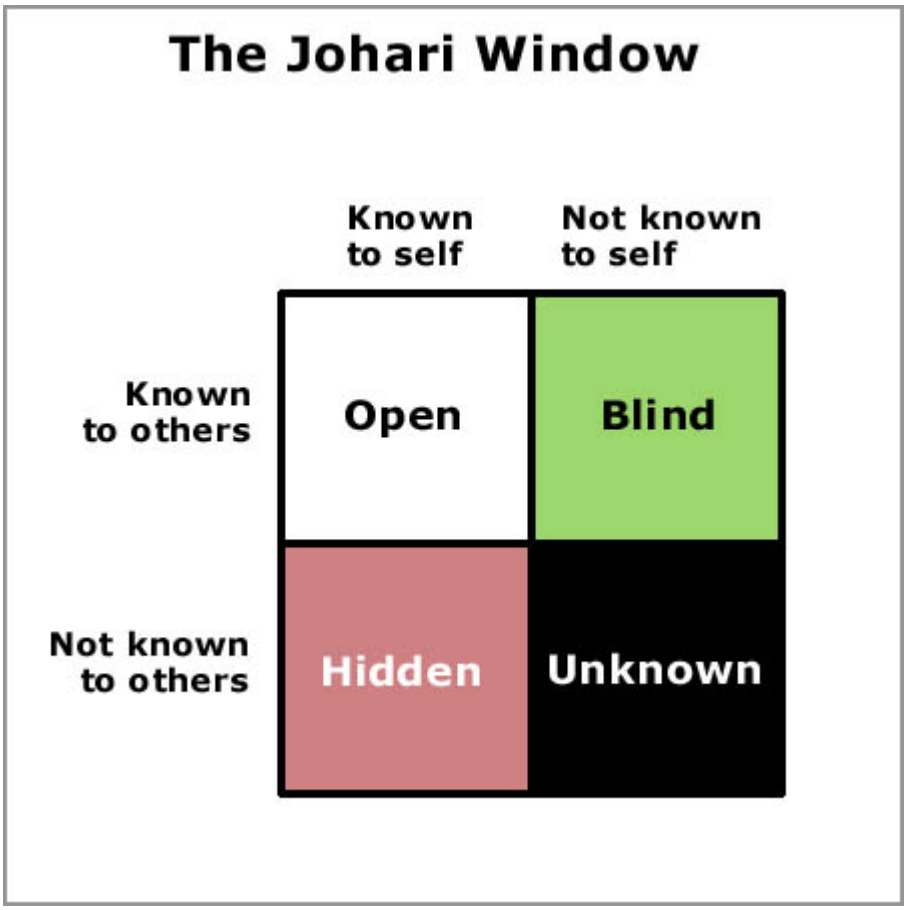
Active Listening

Do not give solution!

Feedback

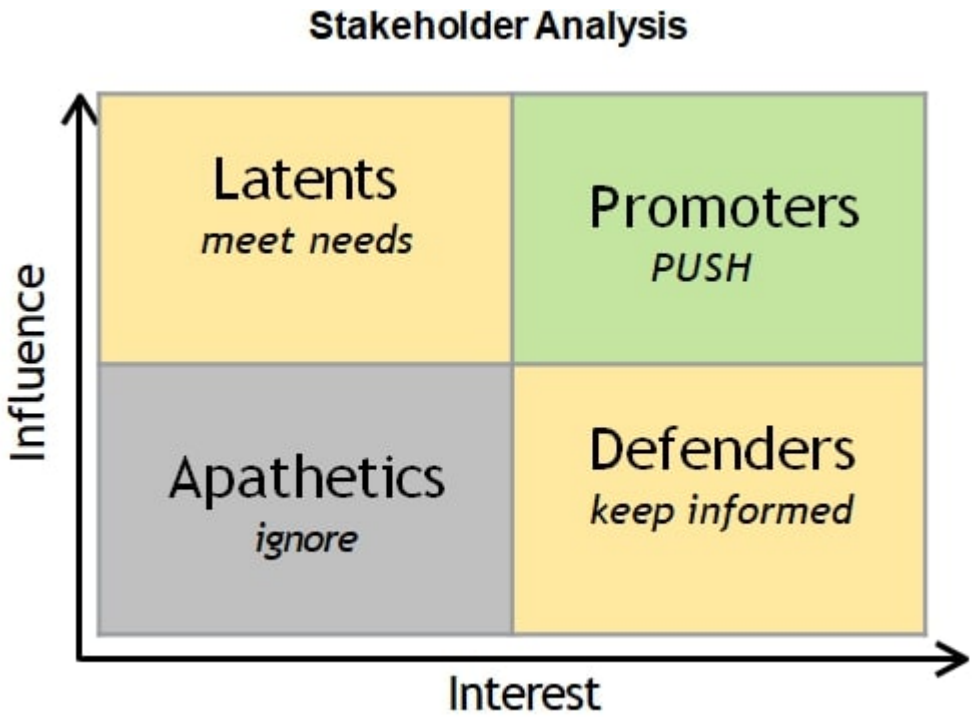


You can use feedback for blind spots.



Stakeholders

Who's buy-in you need?



Situational Leadership

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?

Willing

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

Able

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

Allowed

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

The Situational Leadership Model

Conflict Management

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

SIEMENS
Ingenuity for l

1 Go to the balcony	Turn off emotions Get time to think
2 Step to the other side	Temporarily turn off speculations Find common ground
3 Reframe the game	Perception change through questions (Socratic method) Provide options
4 Build the golden bridge	Involve them to craft an agreement Address fear of losing face, unmet interests
5 Make it hard to say "no"	Probe for consequences of not reaching agreement Show them the way out: the golden bridge Final commitment

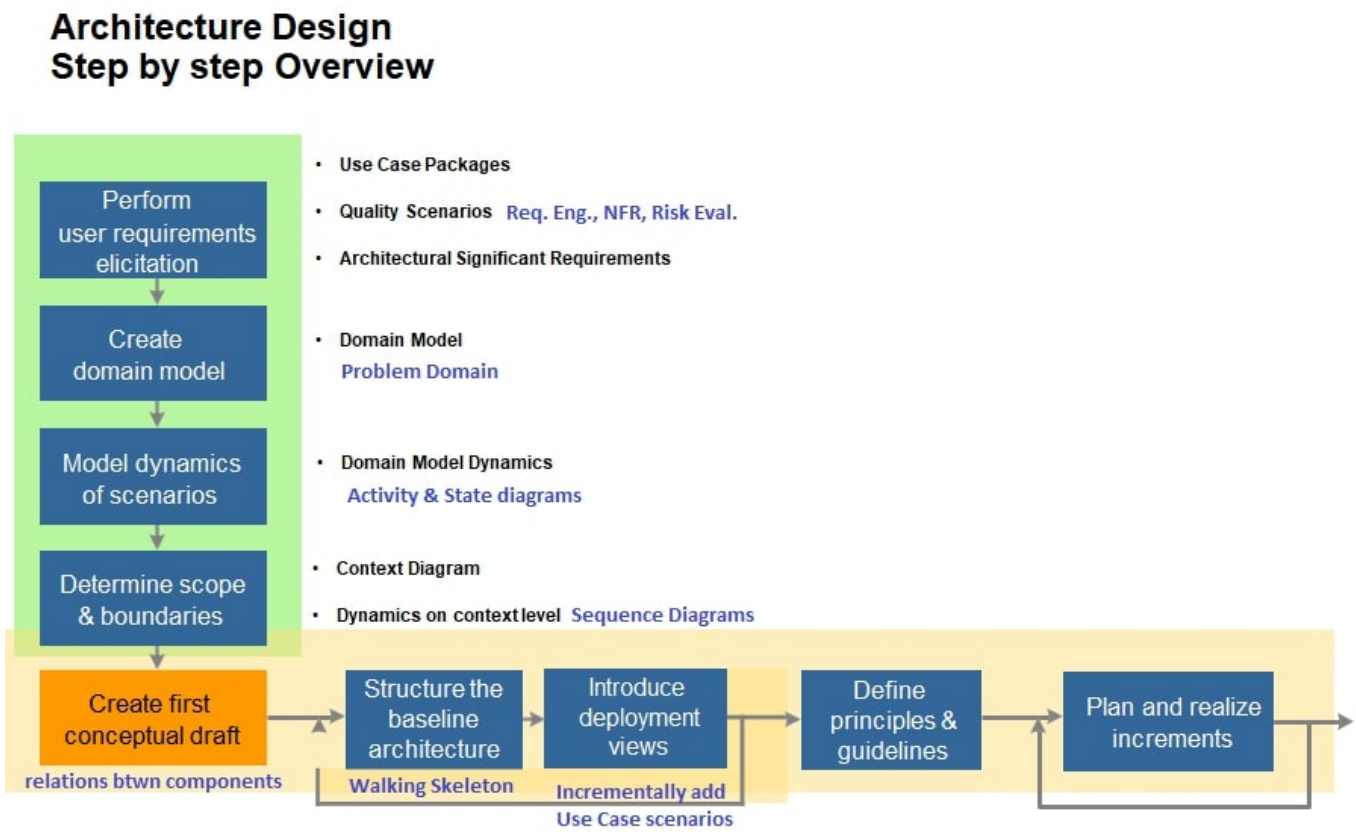
SYSTEMATIC ARCHITECTURE

What is architecture?

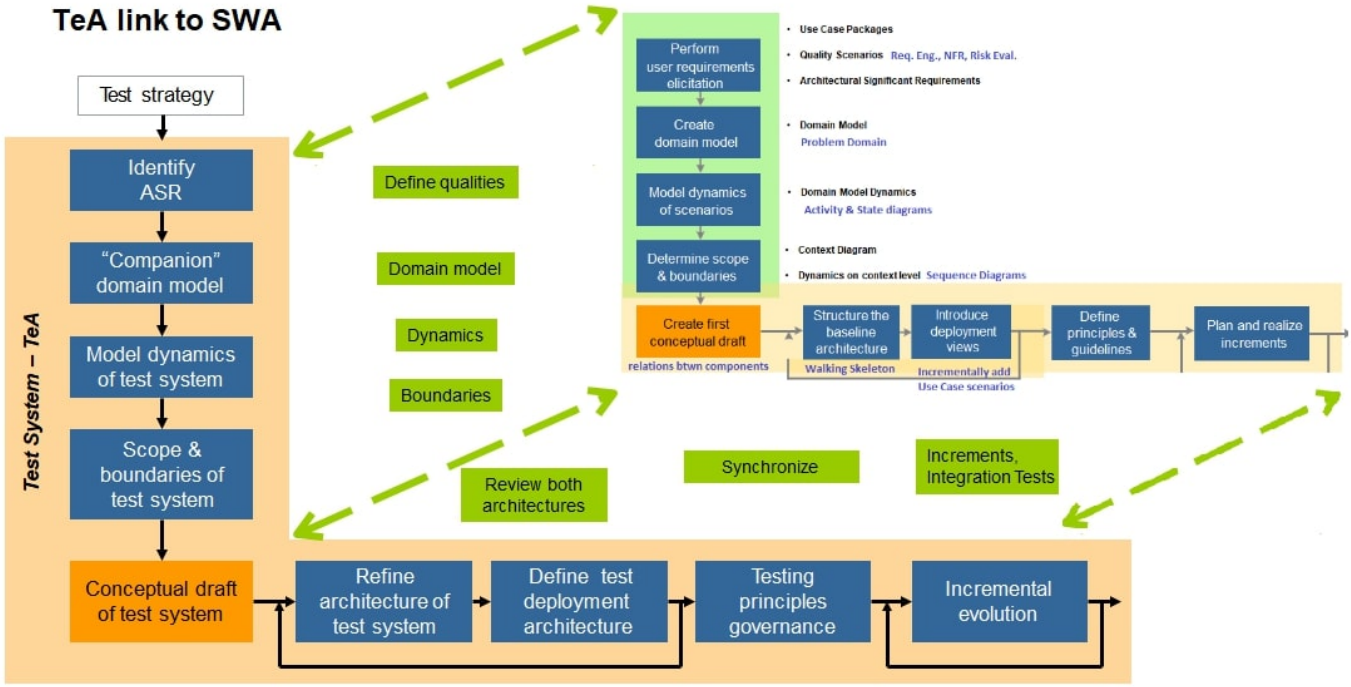
- Abstraction of reality
- Focus on fundamental & critical concepts
- Rationels / Why

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

Architecture Design - step by step



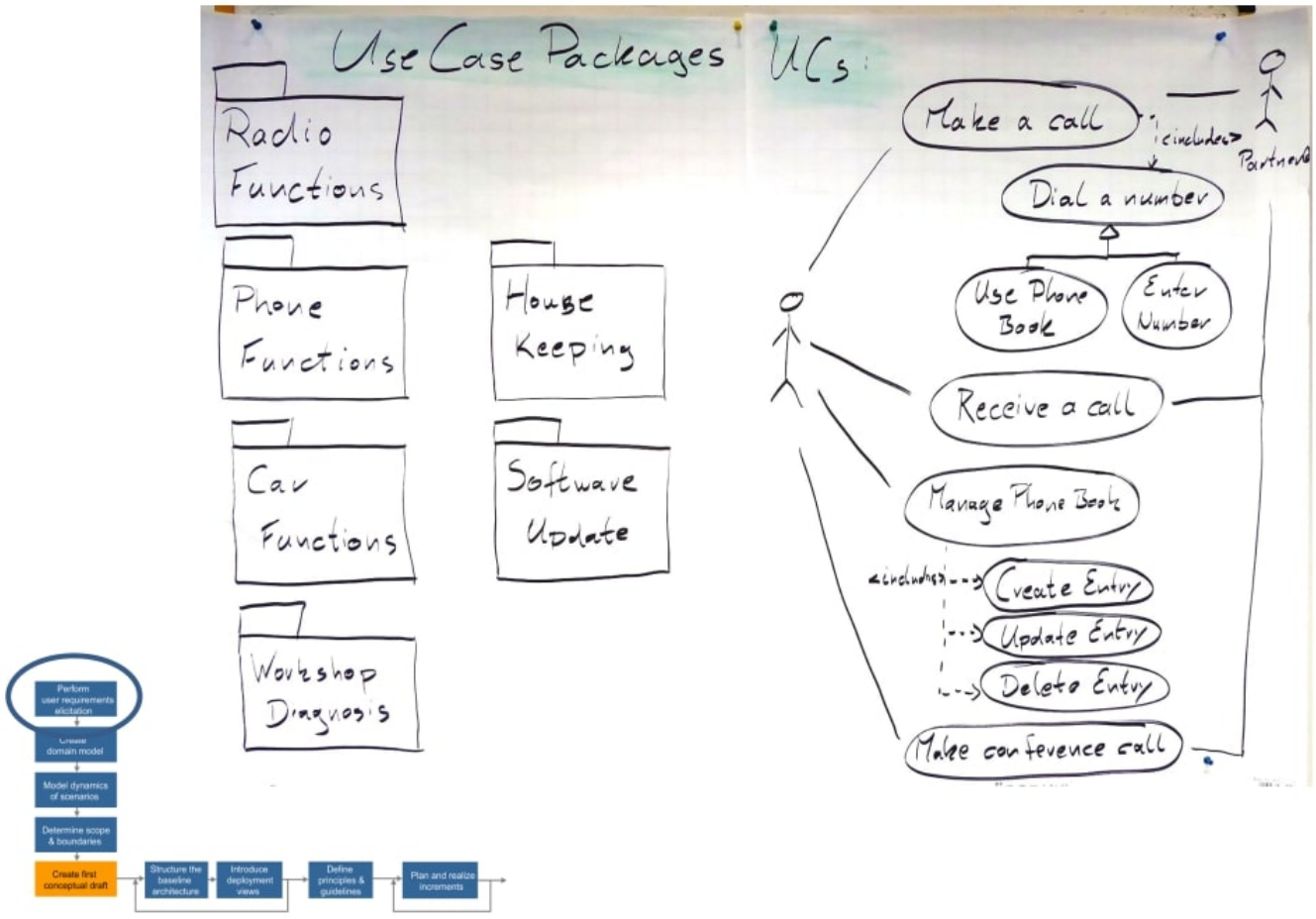
Link between TeA and SWA



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

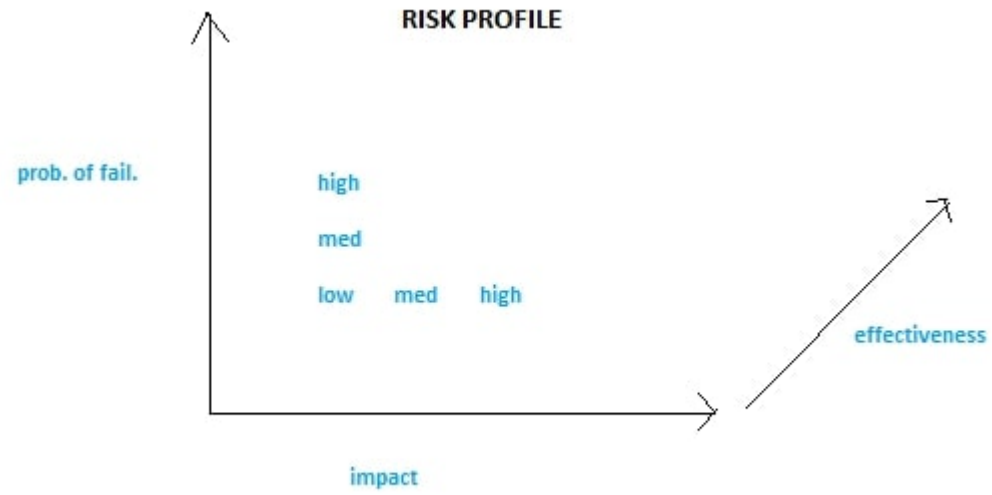
Use Case Packages

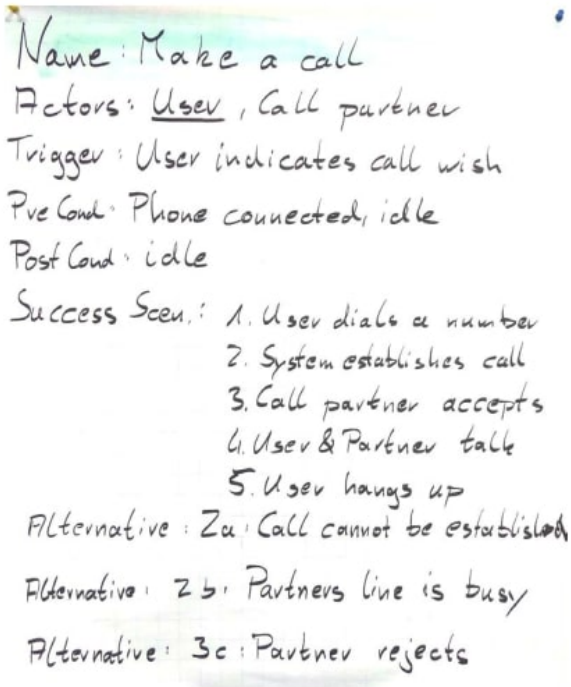
Each use case is a requirement.

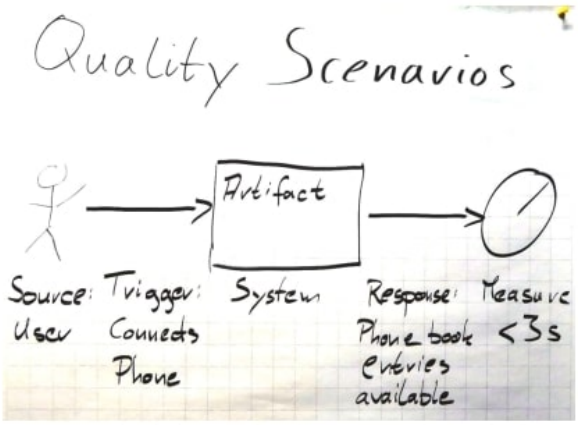


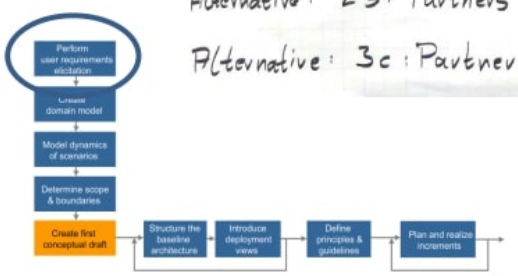
Quality Scenarios

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

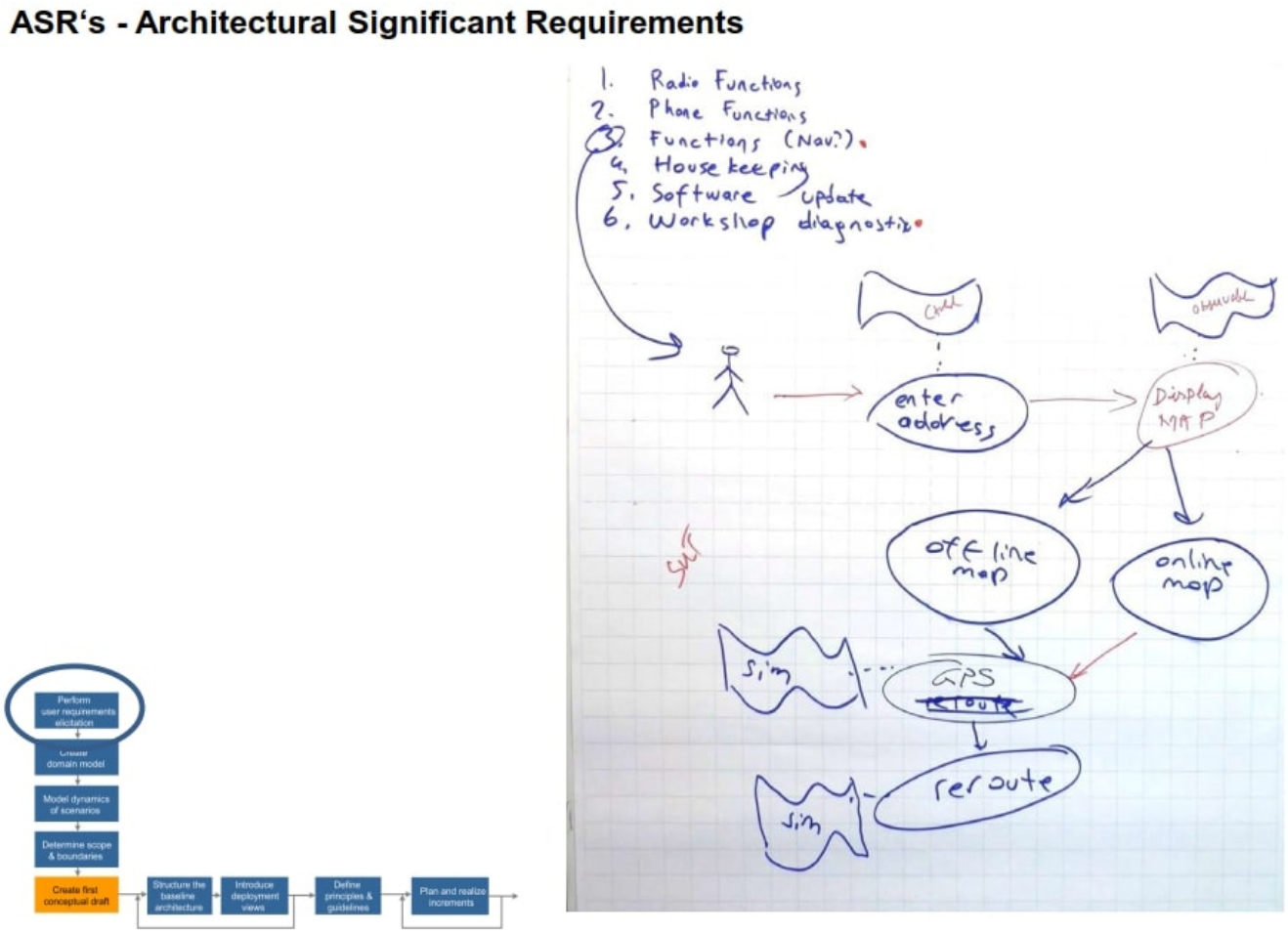








Architectural Significant Requirements (ASR's)



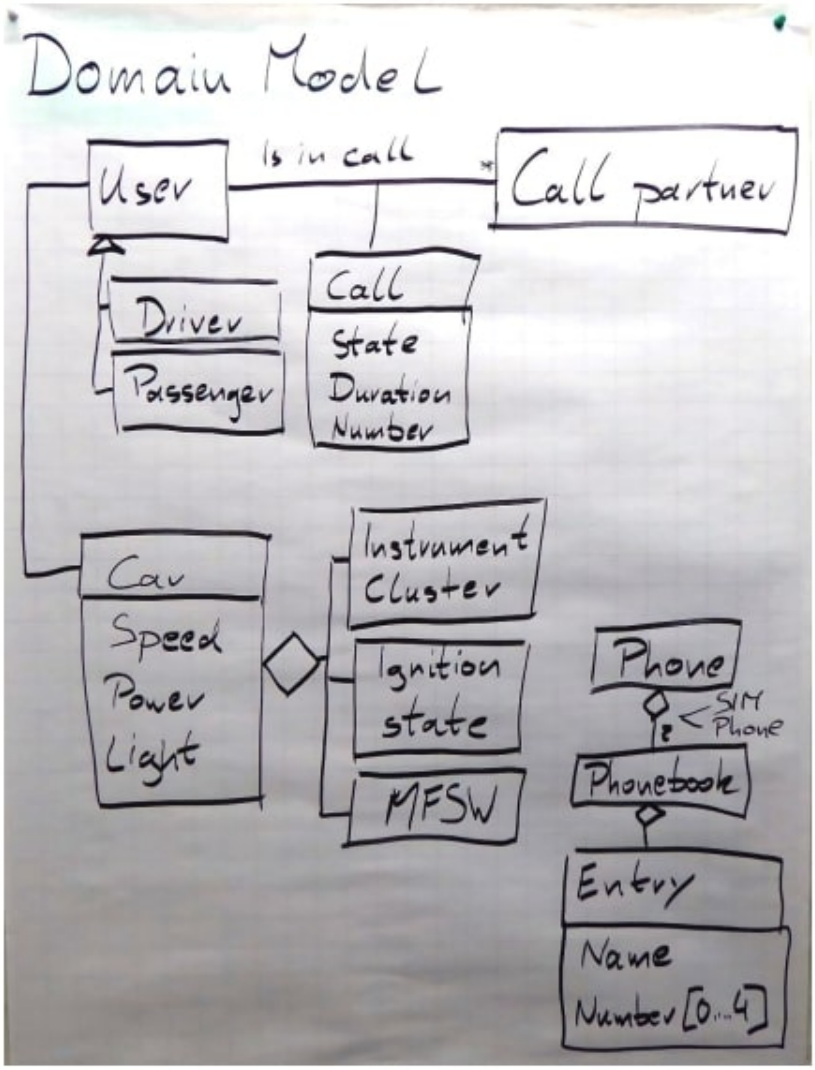
2) SWA: Domain Model - TeA: "Companion" Domain Model

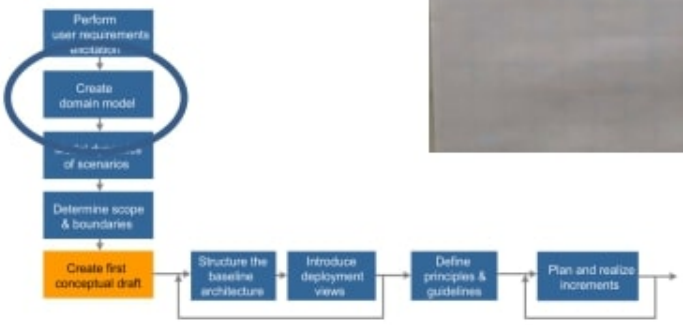
This is not a class diagram, but the **Problem Domain**. **It is a conceptual model that incorporates behavior and data**.

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.

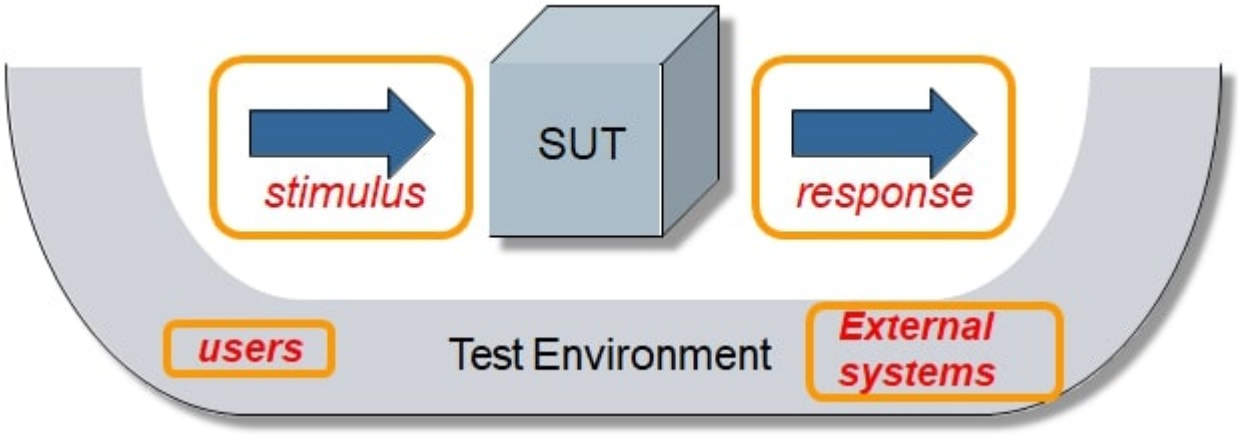
> - *Requirement* : capability or condition needed by a stakeholder
> - *Domain Model* : conceptual model that incorporates behavior and data
> - *Context diagram* : focuses on what is in/out of scope, boundary between system and environment





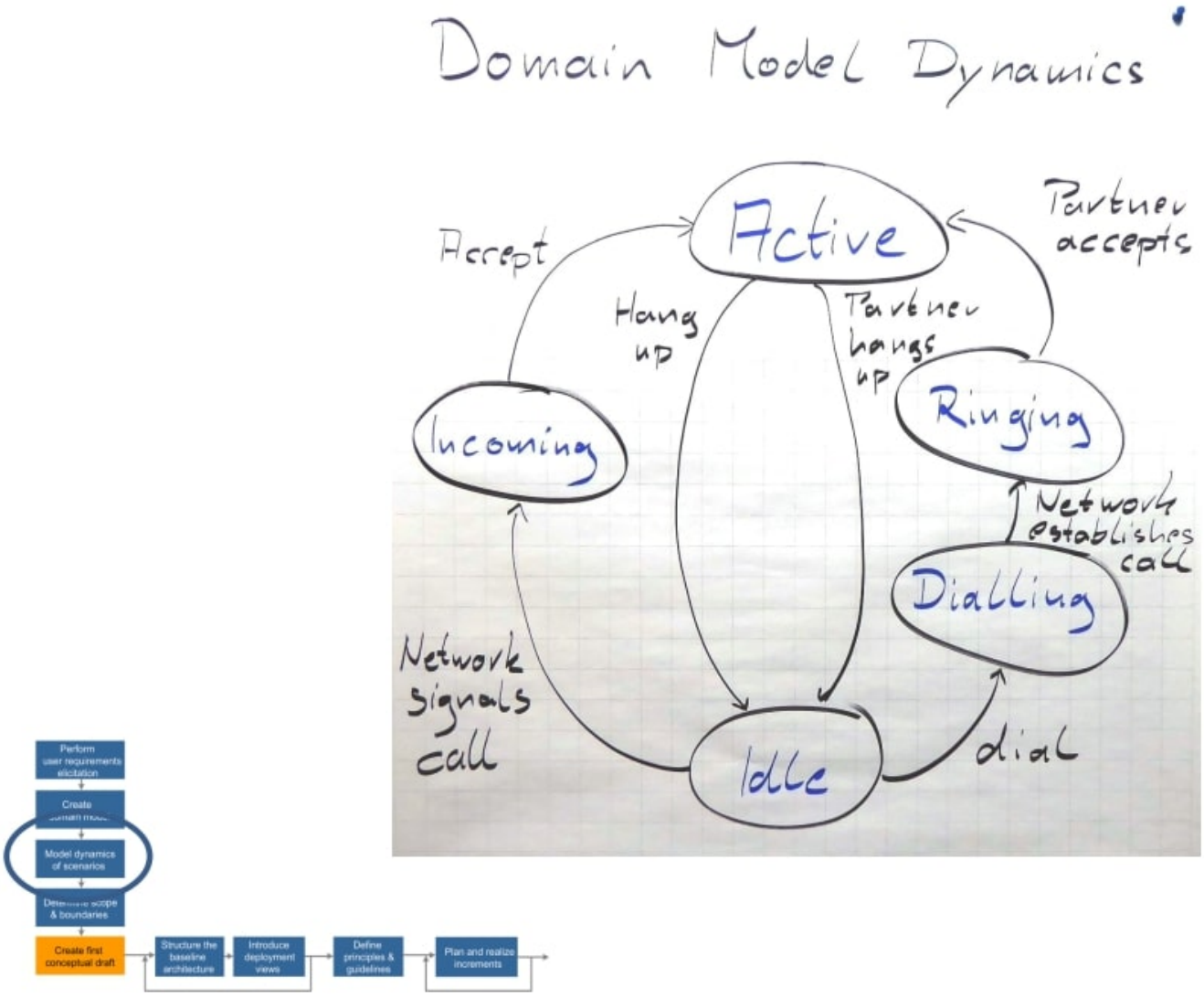
Your Test System is the "companion" Domain Model, you have to cover all the other elements:

- 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

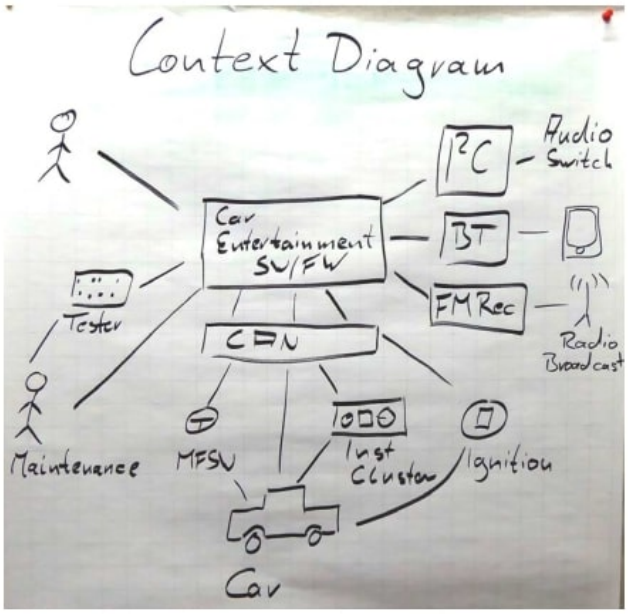
Context Diagram

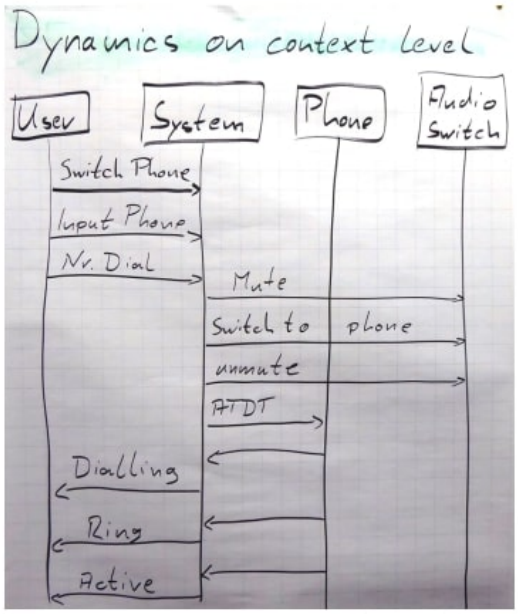
For the product/ system, clearly define the boundaries: **What is IN and what is OUT of scope, boundary between system and environment.**

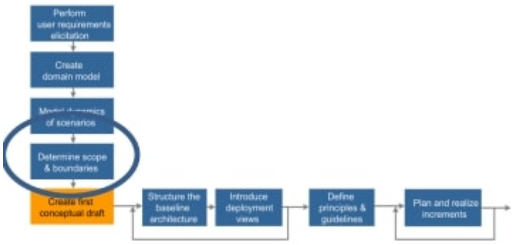
> Context diagram may look like a use case package diagram but has different intent:
>
> - context diagram: describes what is in/out of scope, the so-called “context”, the interaction between system and the environment around.
> - use case package diagram: shows packages and relationships between them,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

Dynamics on Context level

Sequence Diagrams

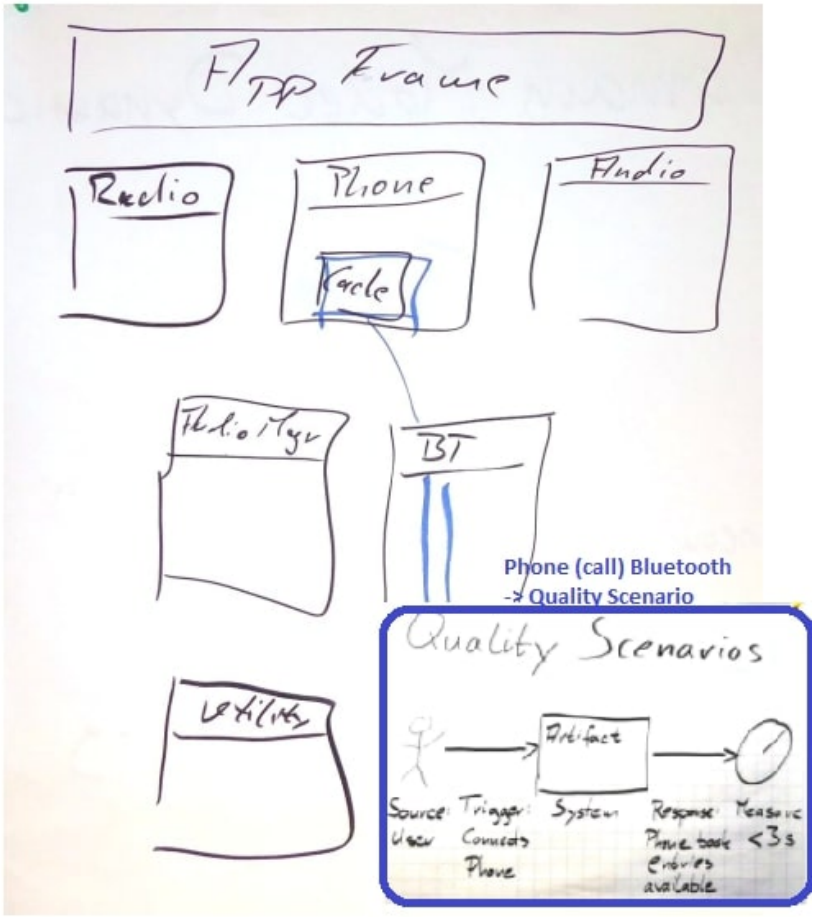


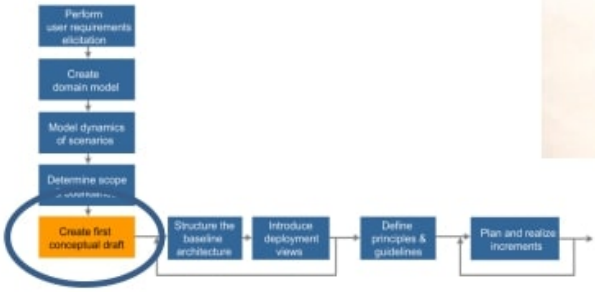




5) SWA: Conceptual Draft - TeA: Test System's Conceptual Draft

Relations between components.





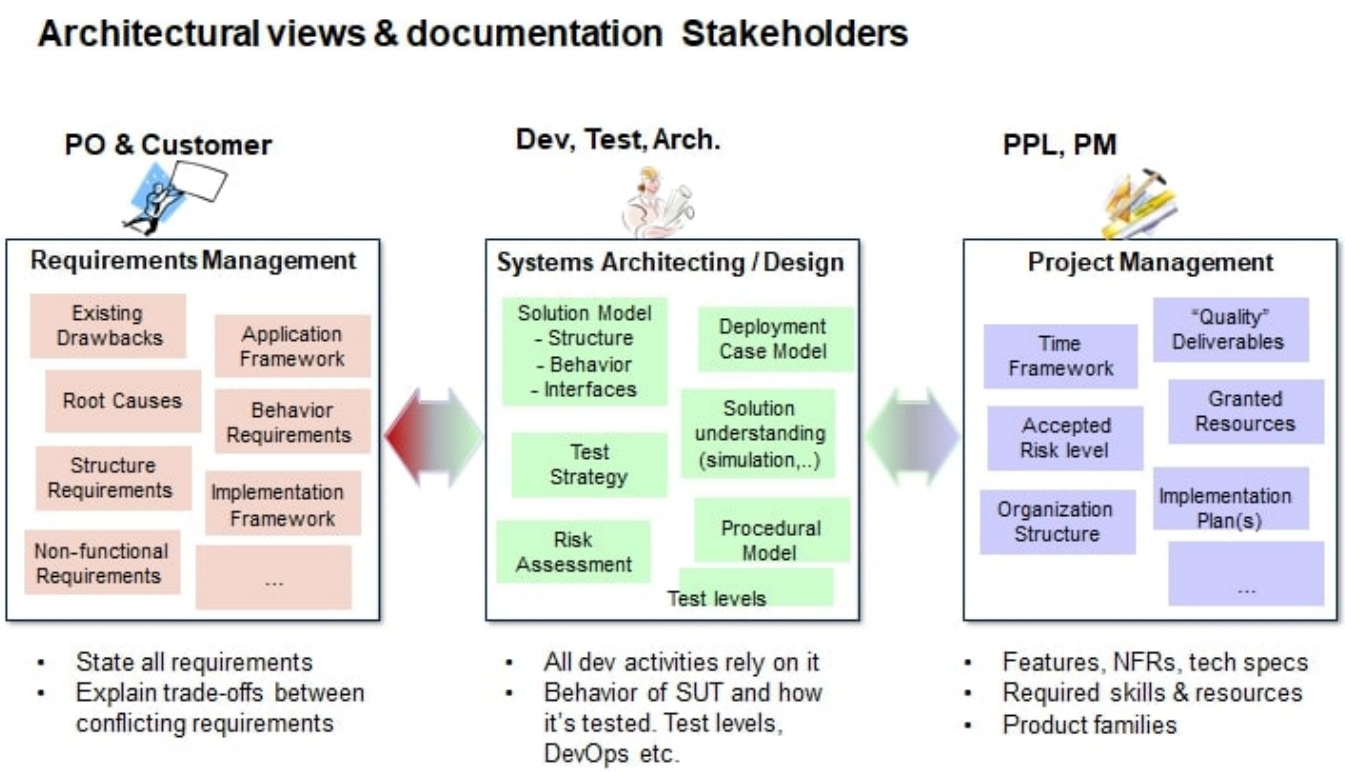
6+) SWA: Structure the baseline Architecture & Introduce Deployment Views - TeA: Refine test architecture & Define test deployment architecture

- 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



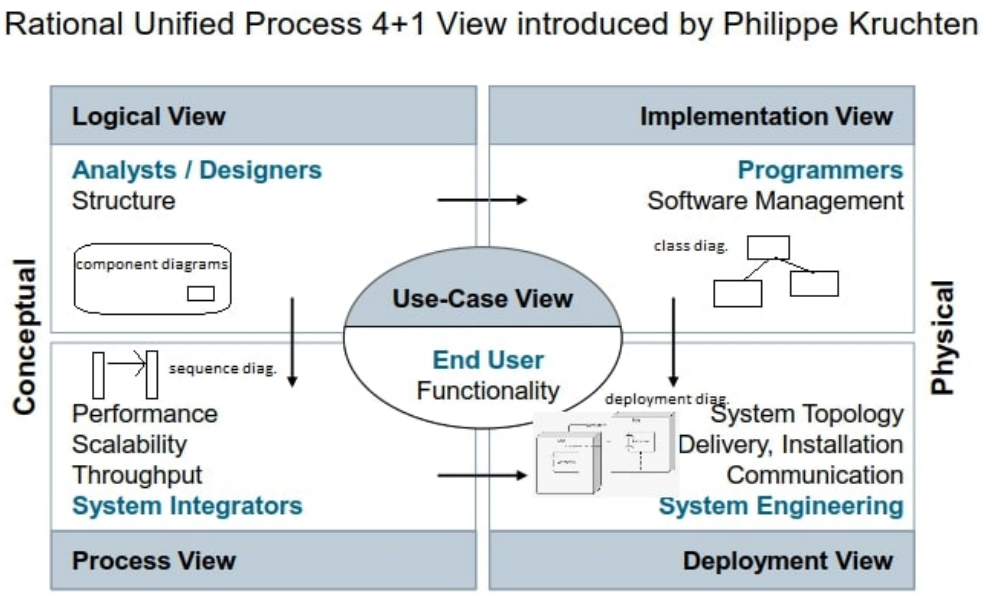
Architectural Views: Kruchten, Zachmann...

4+1 View explained:

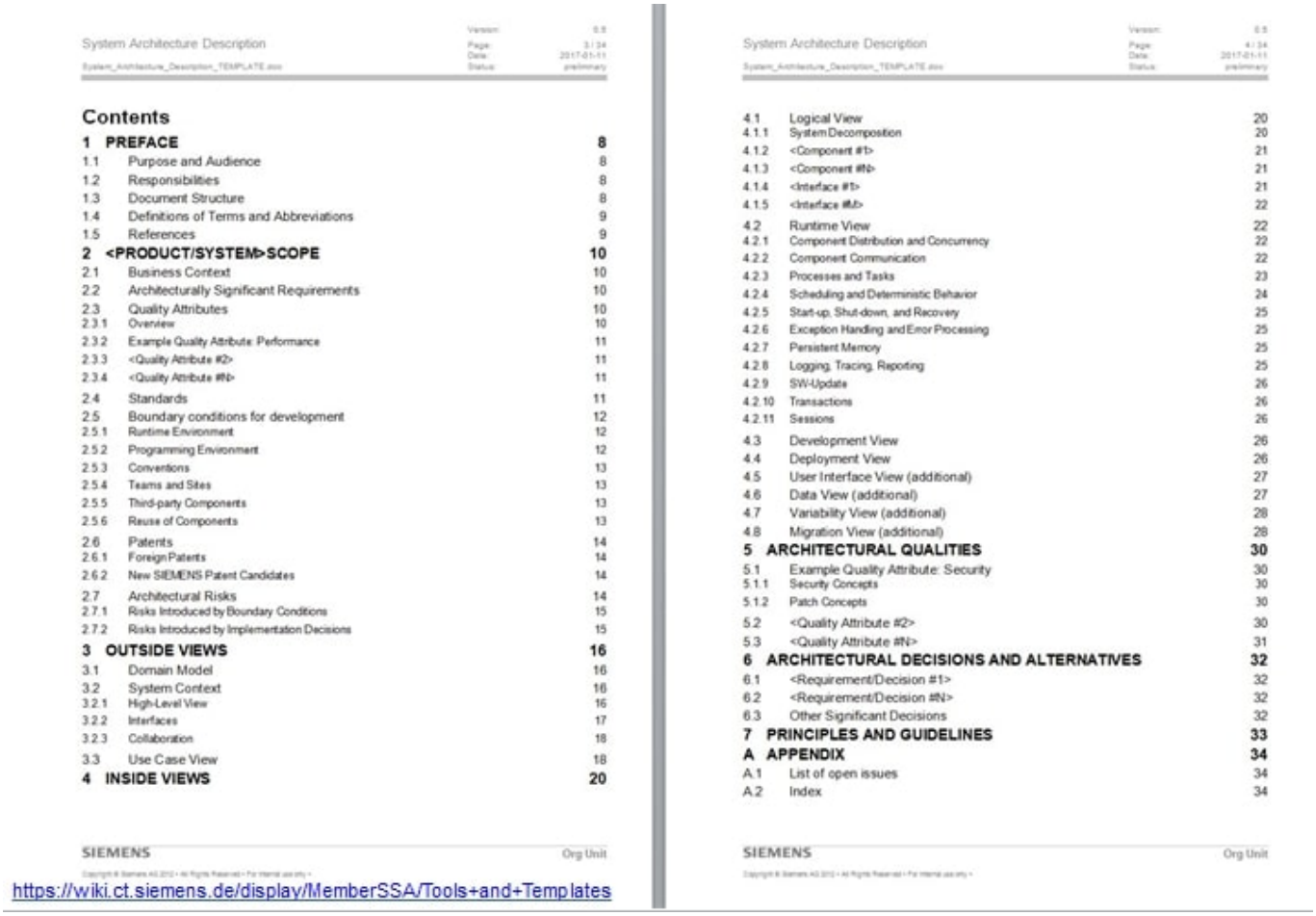
- User view: all posibble scenarios the user expects from SUT
- Functional aspects:
  - Logical View: how the functionality use cases are modeled. *Component diagrams*
  - Implementation/Development View: how the functionality is implemented (source code, libs, executables etc.) *Class diagrams*
- Non-functional 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)



Siemens SW Architecture Doc Template



System Architecture Description

Contents

SIEMENS Org Unit

https://wiki.ct.siemens.de/display/MemberSSA/Tools+and+Templates

Test Architecture Documentation

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

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

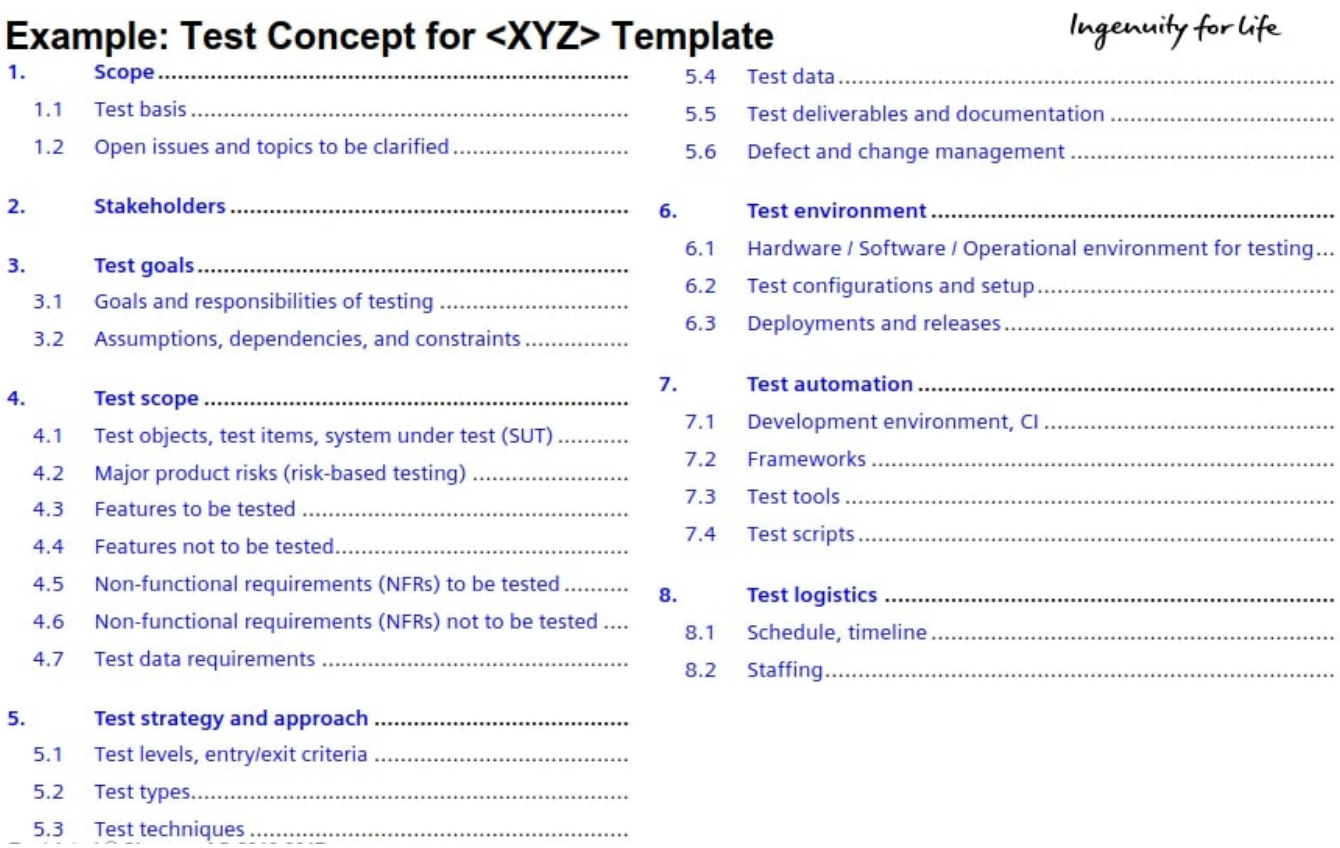
- Part 1: Concepts and Definitions
- Part 2: Test Process
- Part 3: **Test Documentation**
- Part 4: Test Techniques

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



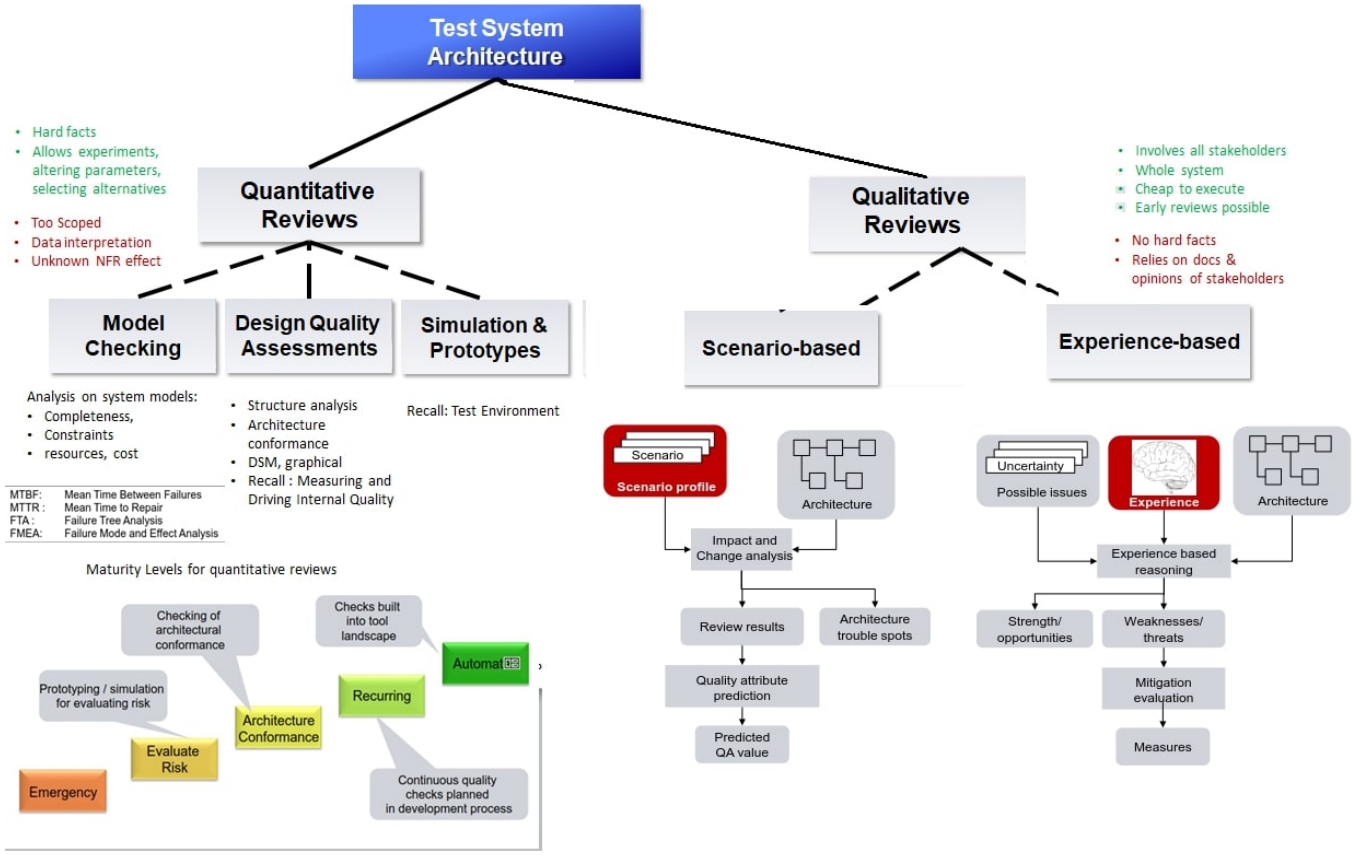
Example: Test Concept for <XYZ> Template

Ingenuity for life

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	▪ Places where the design will not function properly ▪ Contradicting behavior
Inefficiencies	▪ Unnecessary complexity ▪ Inefficient use of resources, channels, etc
Ambiguities	▪ Unclear requirements ▪ Undocumented assumptions ▪ Elements of the design specification which can be understood in different ways
Inflexibility	▪ Elements making change requests difficult ▪ Obstacles for dealing with lifecycle issues (e.g. configuration, service, upgrade, etc.)

SIEMENS
Ingenuity for life

Comparison of qualitative reviews

	Active Design Review	Industry Practice	Architecture Tradeoff Analysis Method	System Architecture Analysis for the Field
Interaction	Designer, reviewer	Interviews	Workshop	Workshop
Phase	Detailed component / module design ready	After architecture has been designed	Architecture design complete enough for walkthroughs	Detailed component / module design ready
Strength	Focused on finding defects in design	Concrete measures	Bring stakeholders together, requirement prioritization	Concrete measures
Key restriction	Small scale	No common understanding of requirement priorities	No measures	Focus on system in operational conditions
Duration	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
Intention	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
Interaction	Workshop	Workshop	Designer, reviewer	Interviews
Phase	Architecture design complete enough for walkthroughs	Architecture design complete enough for walkthroughs	Detailed component / module design ready	After architecture has been designed
Strength	Bring stakeholders together, requirement prioritization	Like SAAM, but deeper architectural evaluation	Focused on finding defects in design	Concrete measures
Key restriction	No risks, no measures	No measures	Small scale	No common understanding of requirement priorities
Duration	2–3 days	Two weeks	2 days / reviewer	Four weeks regular 1 day flash

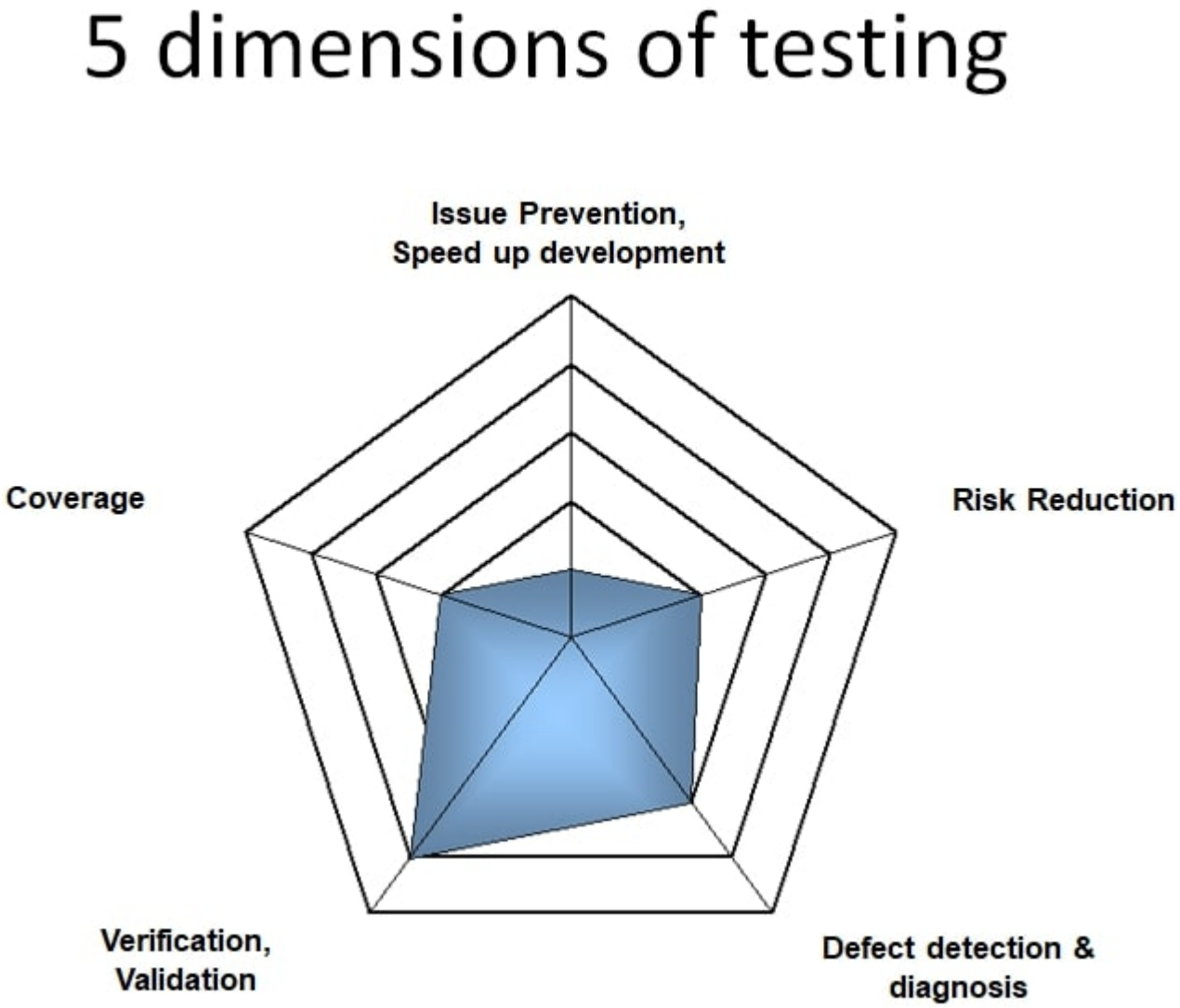
TESTING & QUALITY

Value of Testing

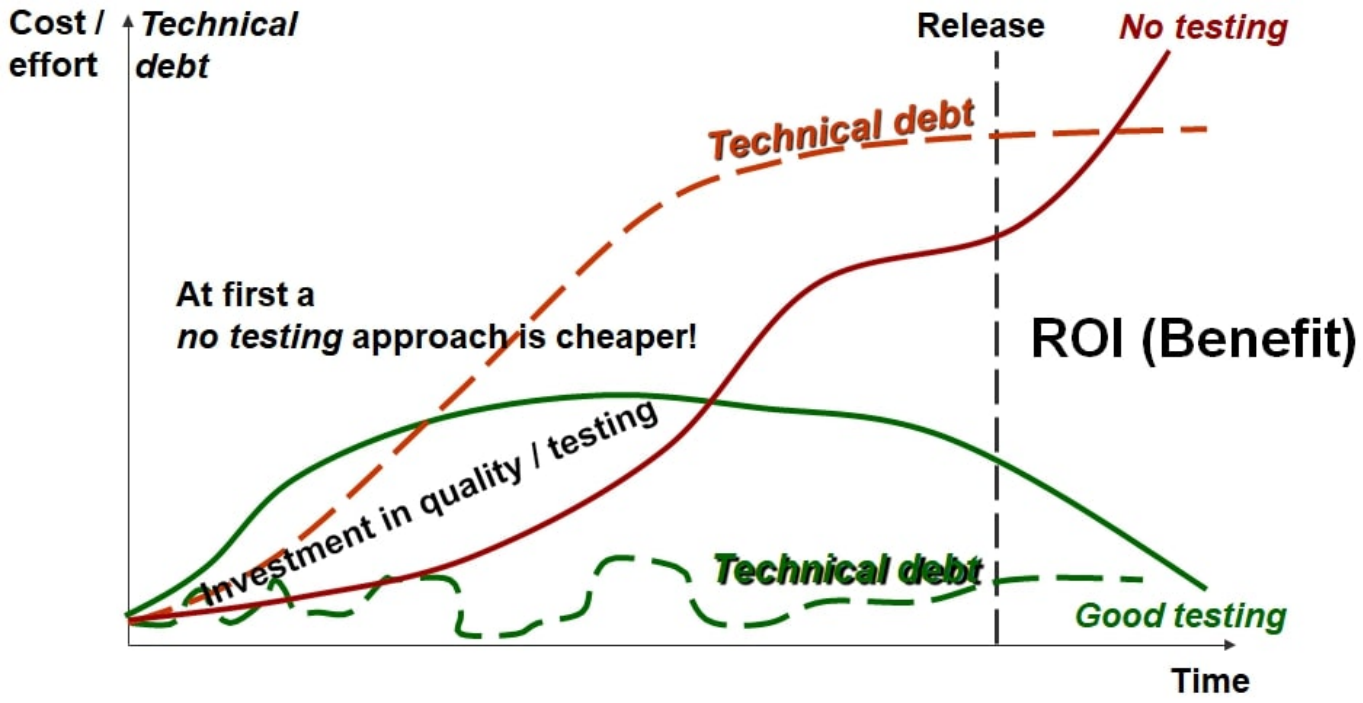
What is Testing? : investigation of the SUT to provide **information** that results in **improvements**.

The 5 Dimensions of Testing

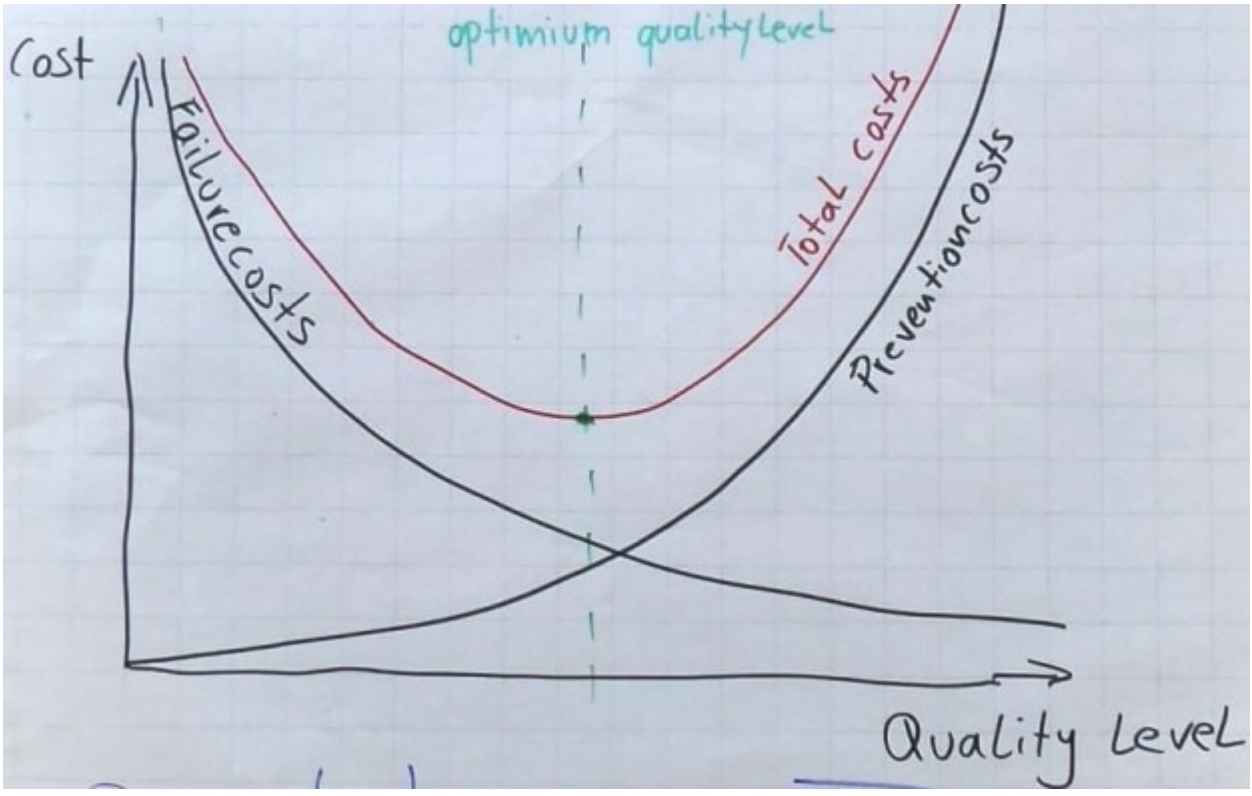
> Coverage: an assessment for the thoroughness or completeness of testing with respect to our test model - *Paul Gerrard*



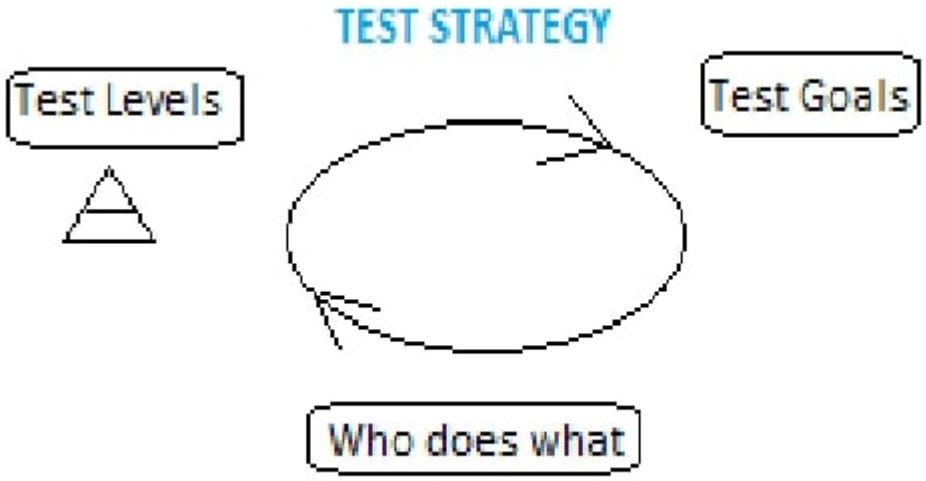
ROI of Testing



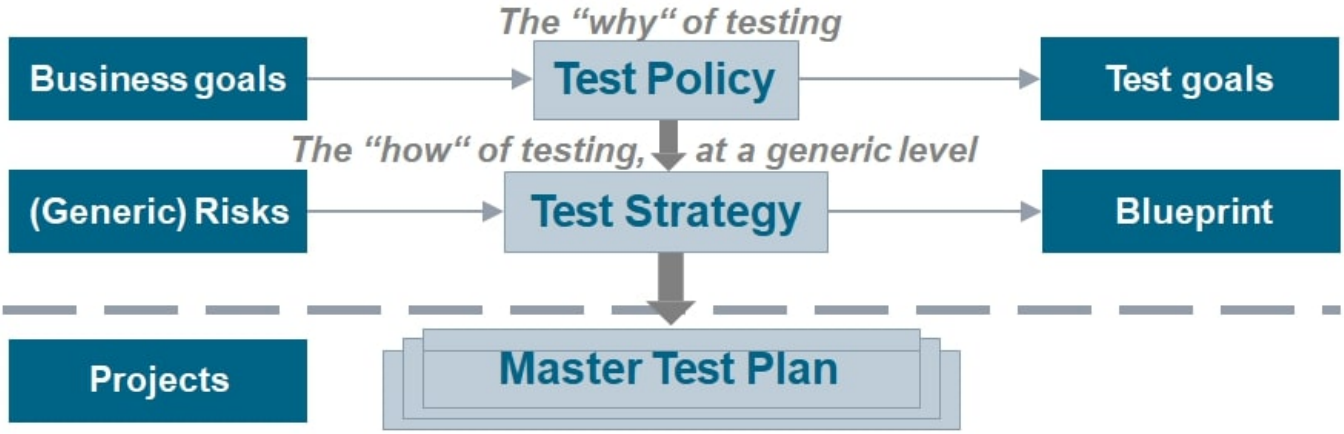
Cost of Quality



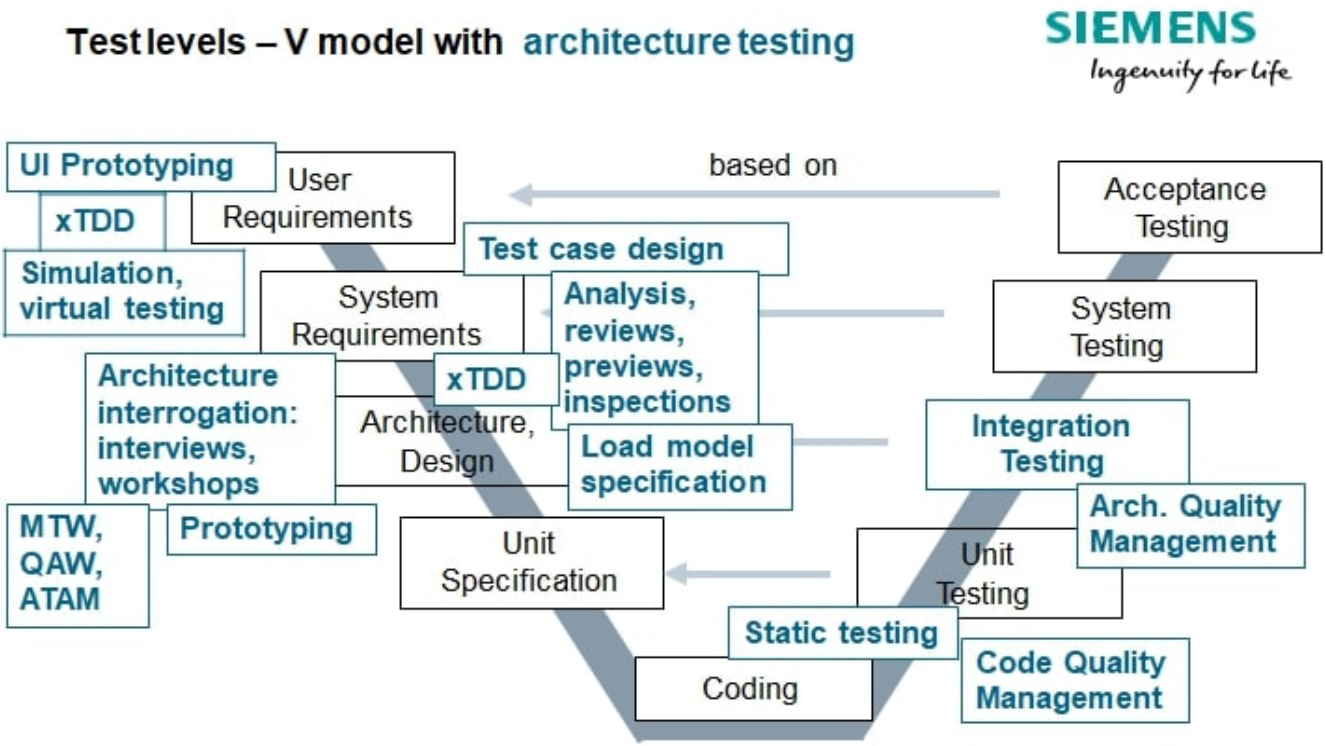
Test Strategy



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

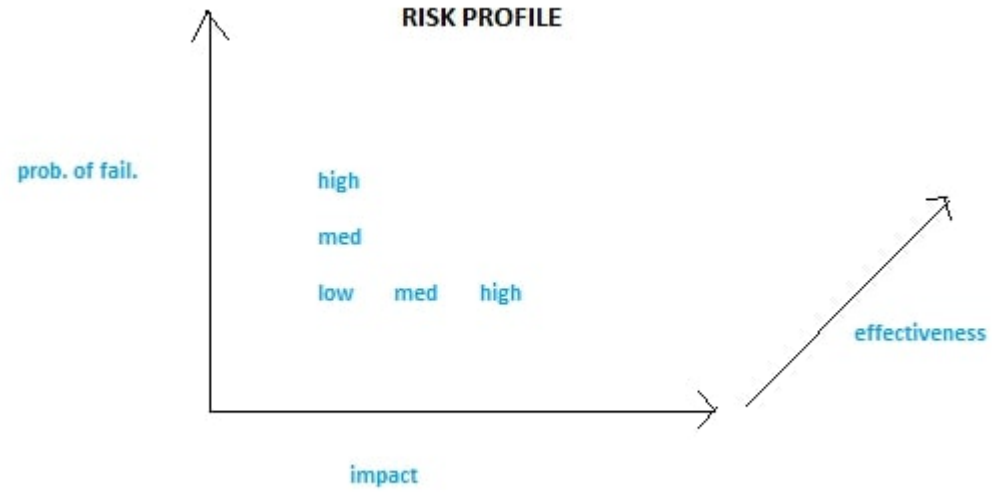


Test levels – V model with architecture testing



Risk Based Testing

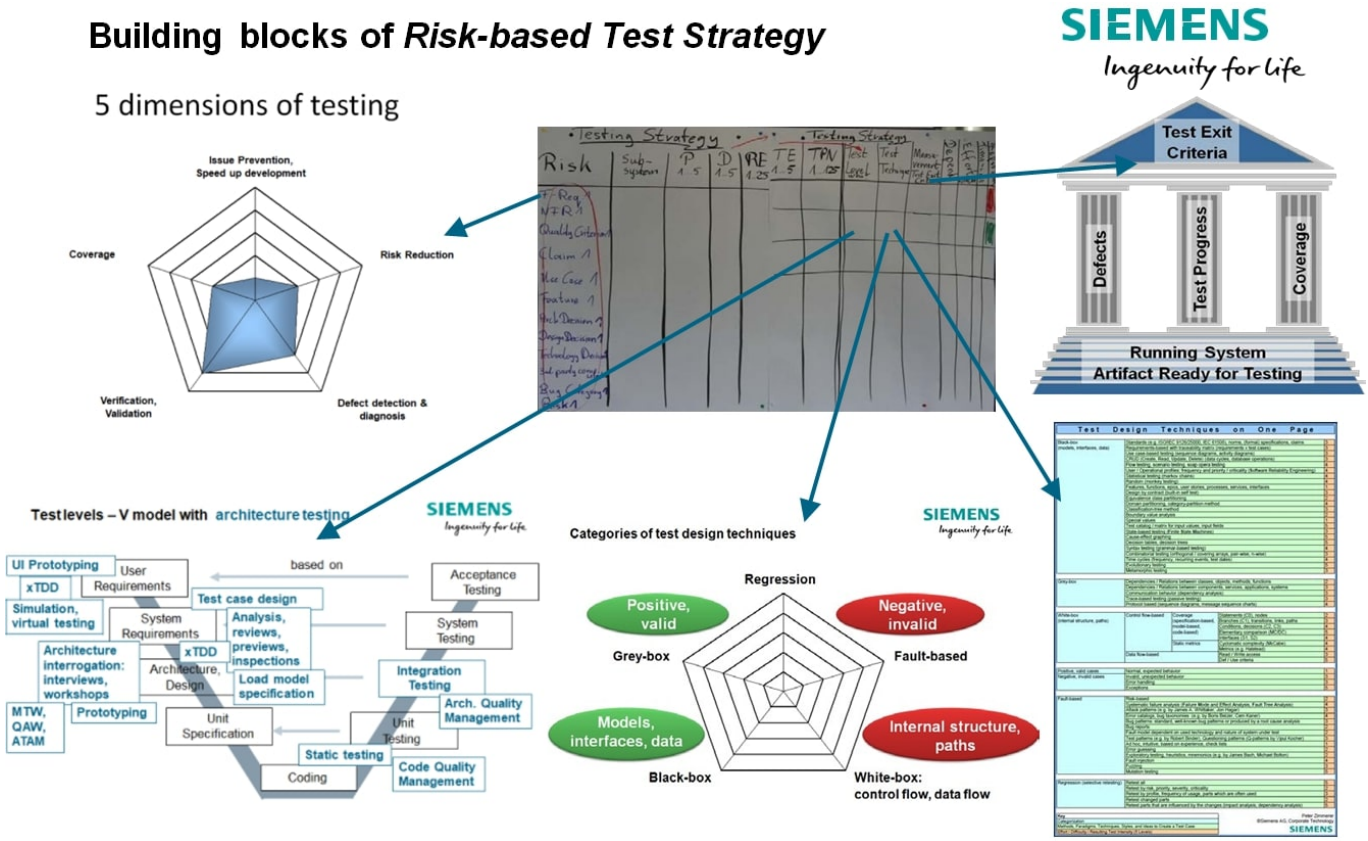
Risk Profile



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

	Identifier	Risk (Failure Mode)	Objective/Benefits Threatened	Subsystem	Probability	Damage (Consequence & Cost)	Exposure	Test Effectiveness	Test Priority Number	Test Objective(s)	Test Level	Test Technique	Measurement	Dependencies	Effort	Timescale	Reporting
Description	ID of entry	Brief description of the (product) risk, that is mode of failure	Which cardinal objective or business benefit(s) is threatened?	Which part of the system is concerned (how much) with this objective/benefit, that is risk?	What is the likelihood of the system being prone to this mode of failure (that is risk)? - Frequency of use - Chance of failure: criticality & complexity at implementation, criticality & complexity at usage, lack of quality	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?	Product of **Probability**, **Consequence**, and **Test Effectiveness** that is product of **Exposure** and **Test Effectiveness**	What test objective will be used to address this risk?	In which test level is this testing performed? By whom (person or group)?	What method or technique is to be used in testing?	How can the attainment of the threatened objective/benefit, that is the risk reduction or elimination be	What do the testers assume or depend on?	How much effort is required to do this testing?	How much elapsed time is required to do this testing?	Objective/Ben If attained, th is risk reduced eliminated
Scores, 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 1-20%: Highly unlikely, chances are slight 2 21-40%: Unlikely, probably not 3 41-60%: We doubt, improbable, better than even 4 61-80%: Probable, likely, we believe 5 81-99%: Almost certainly, highly likely	Scores from 1 to 5: 1 Negligible: no noticeable effect 2 Low: business will be affected slightly 3 Moderate: business objectives will be affected 4 High: business objectives will be undermined 5 Critical: business objectives cannot be accomplished	Range between 1 and 25	Score from 1 to 5: 1 Testing is not the way to address this risk or an appropriate test objective would prove to be unachievable ... 5 High confidence that testing will find faults and provide evidence that the risk has been addressed	Range between 1 and 125	For example: demonstrate that... verify that... validate that... check that...	For example: unit testing integration testing system testing acceptance testing ... developers integration test group system test group	For example: black-box testing white-box testing	For example: a measurement for a quality criterion a test exit criterion	For example: a test entry criterion	For example: high medium low	For example: days weeks months	Objective/Ben If not attained, risk is not red eliminated
	1	Functional requirement															
	2	Non-functional requirement (NFR)															
	3	Quality criterion															
	4	Claim															
	5	Use case															
	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	3rd party component selection (frameworks, open source, external partnering)															
	17	Core asset in PLE															
	18	Open variant space in software ecosystems															
	19	Bug category															
	20	Risk															
	21																

Relations in RBT worksheet



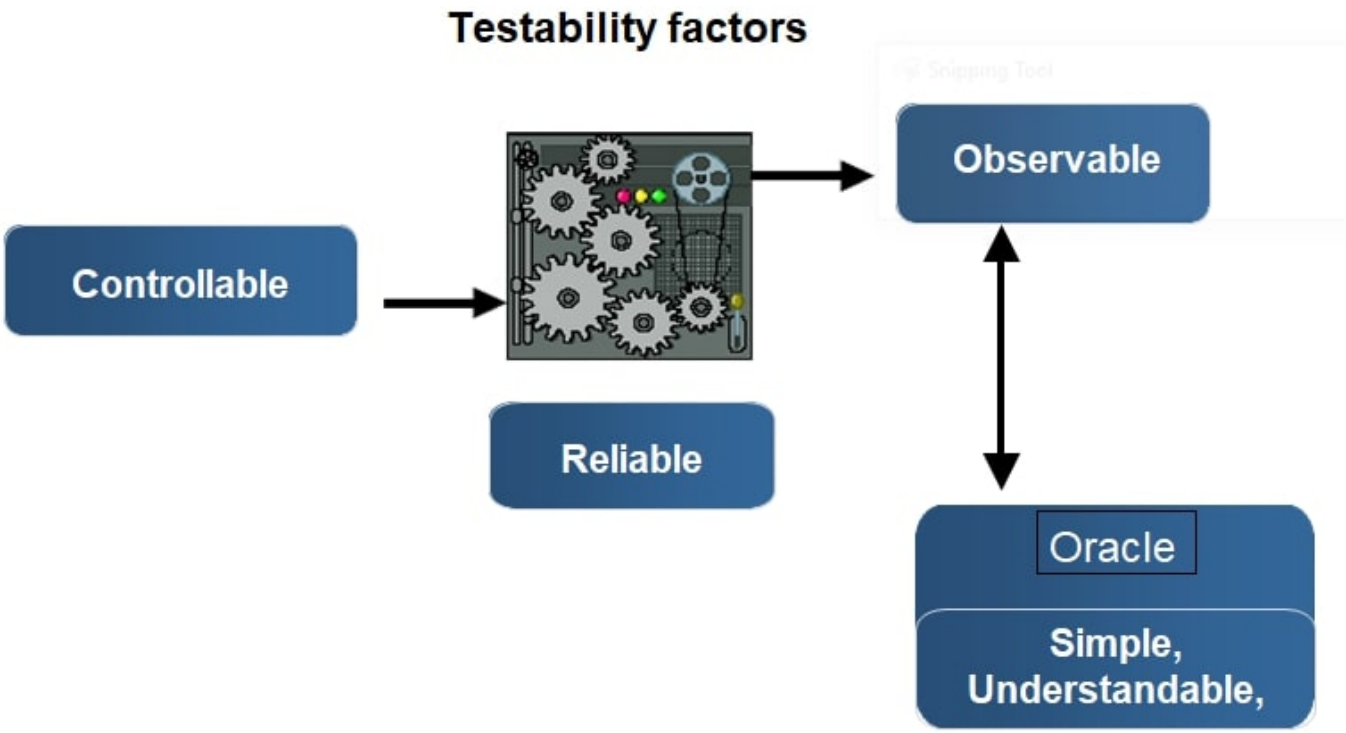
Design for Testability

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

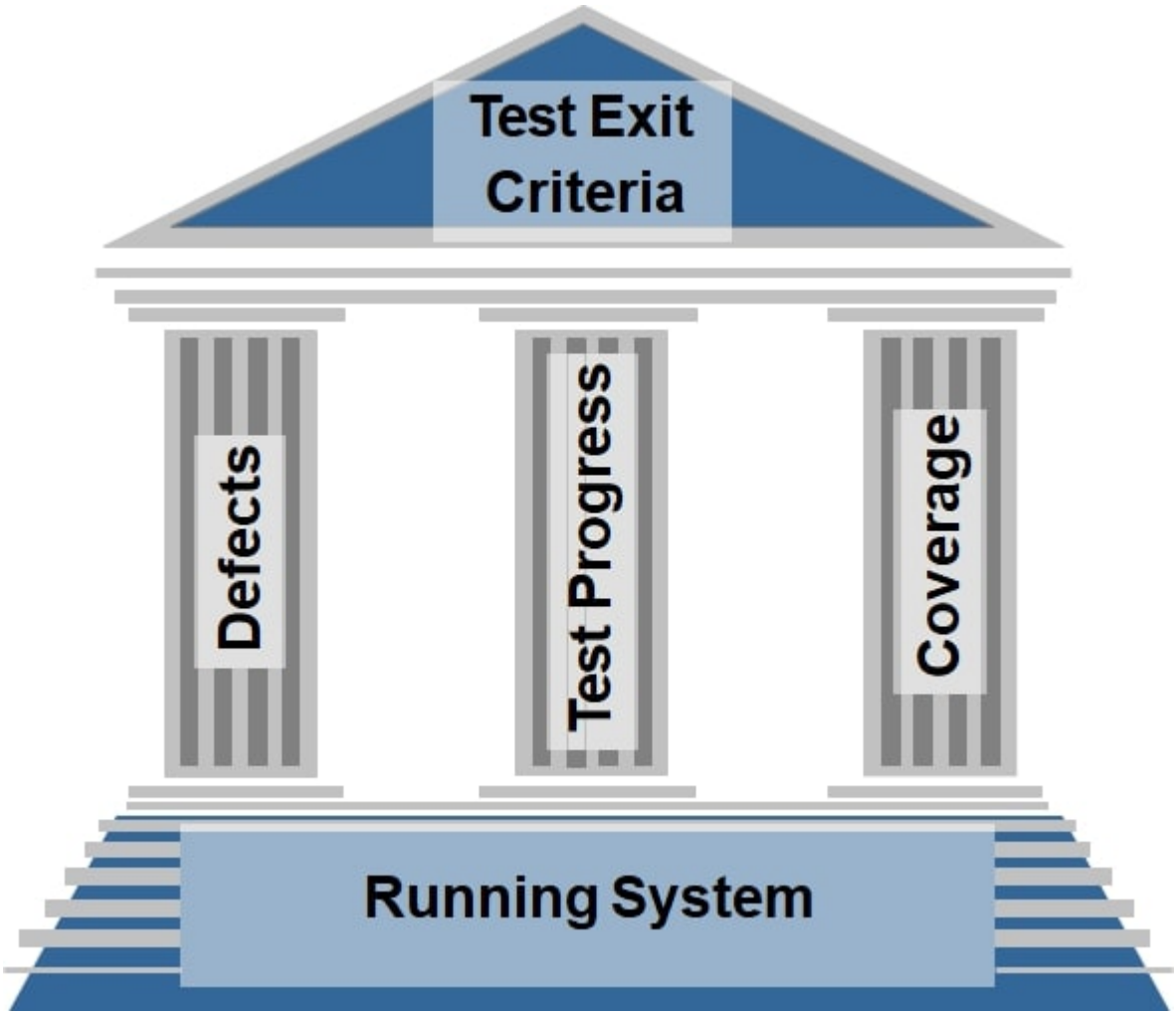
Why : reduce the cost of testing, diagnosis, maintenance.

Who : system, software and test architects

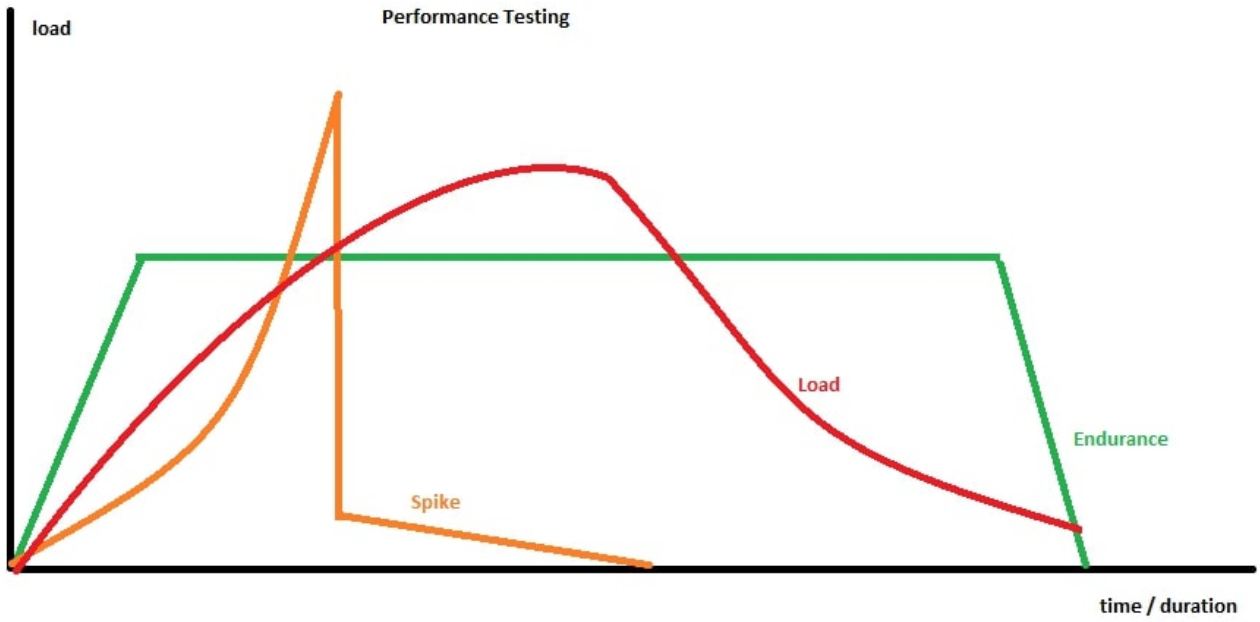
How : TDD, Loose Coupling, Inversion of control, SOLID, follow the best practices of clean code & architecture.



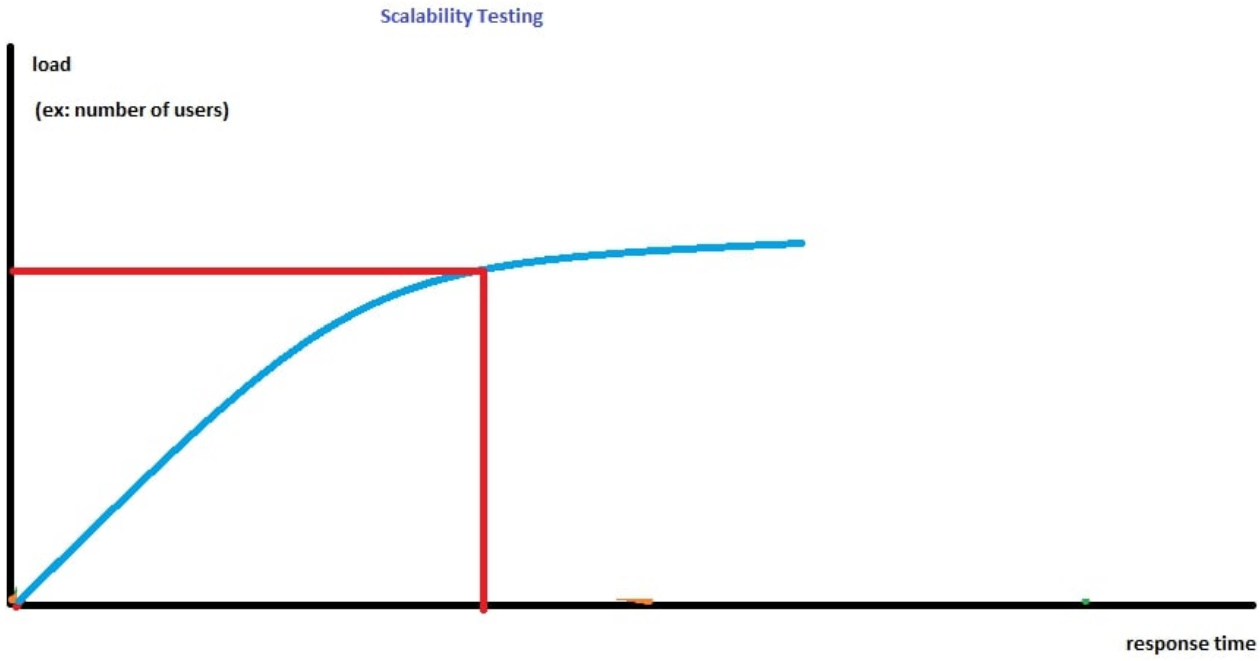
Test Exit Criteria



Performance Testing and Scalability



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



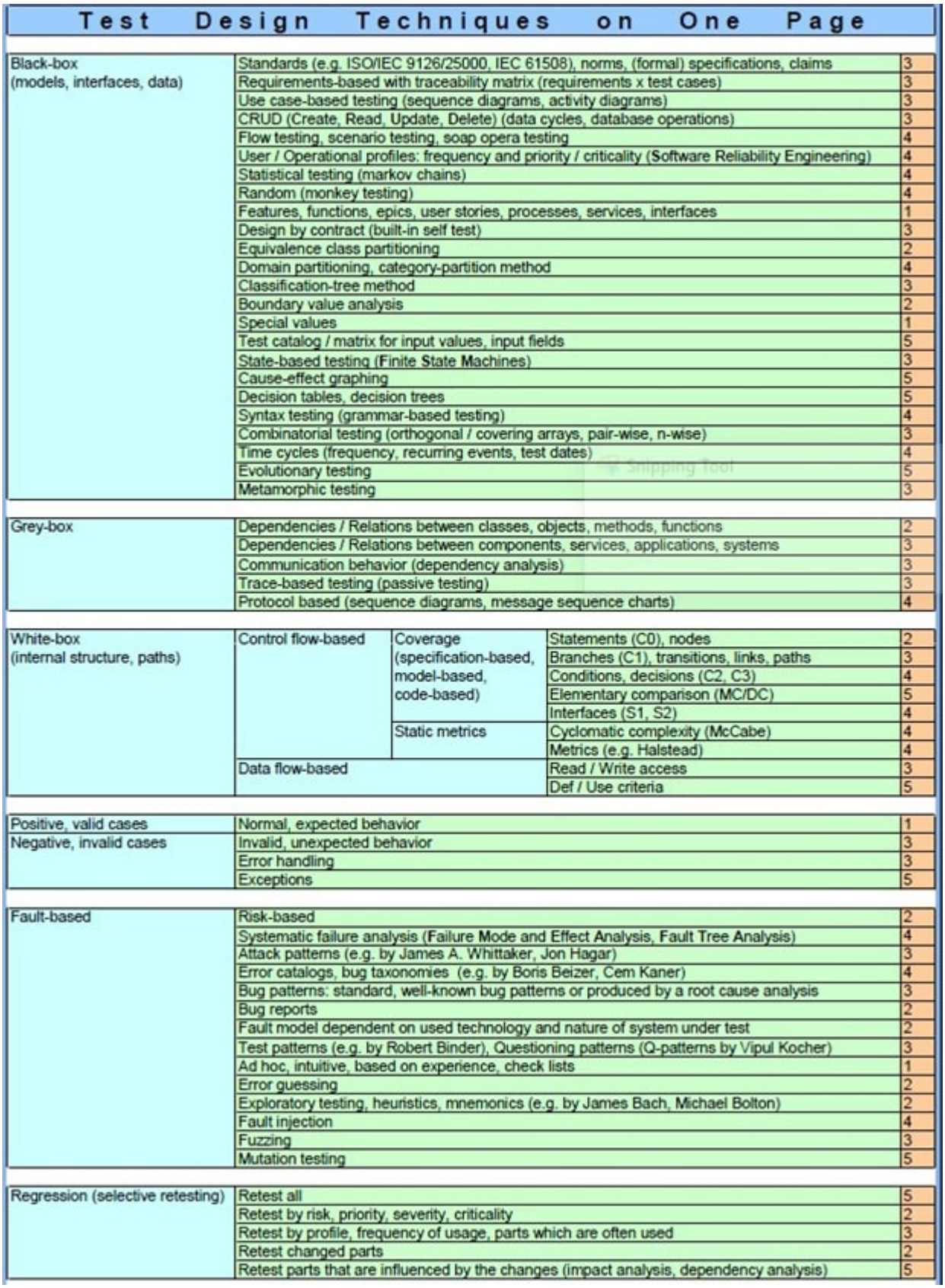
TDD

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

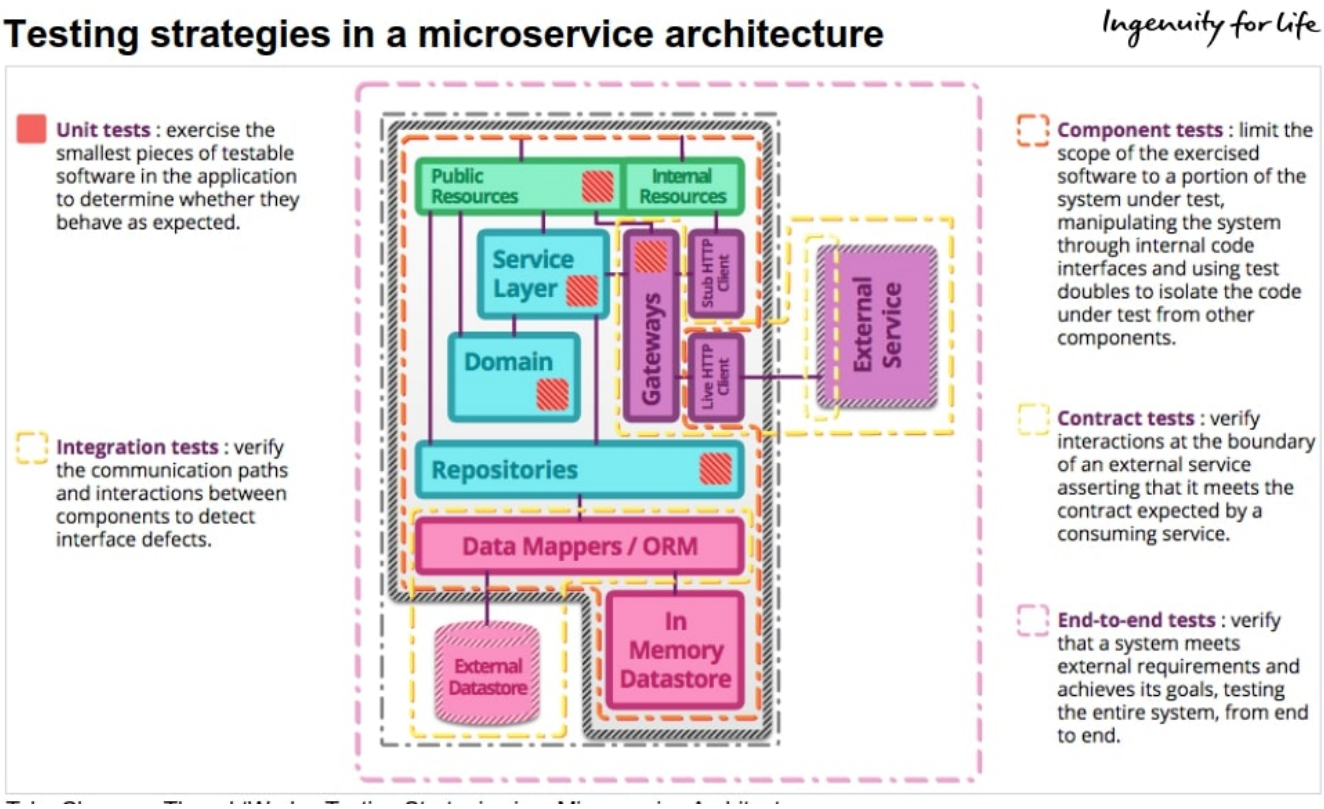
Test Automation Design Patterns paper

Test Environment

Test environment: test rig

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

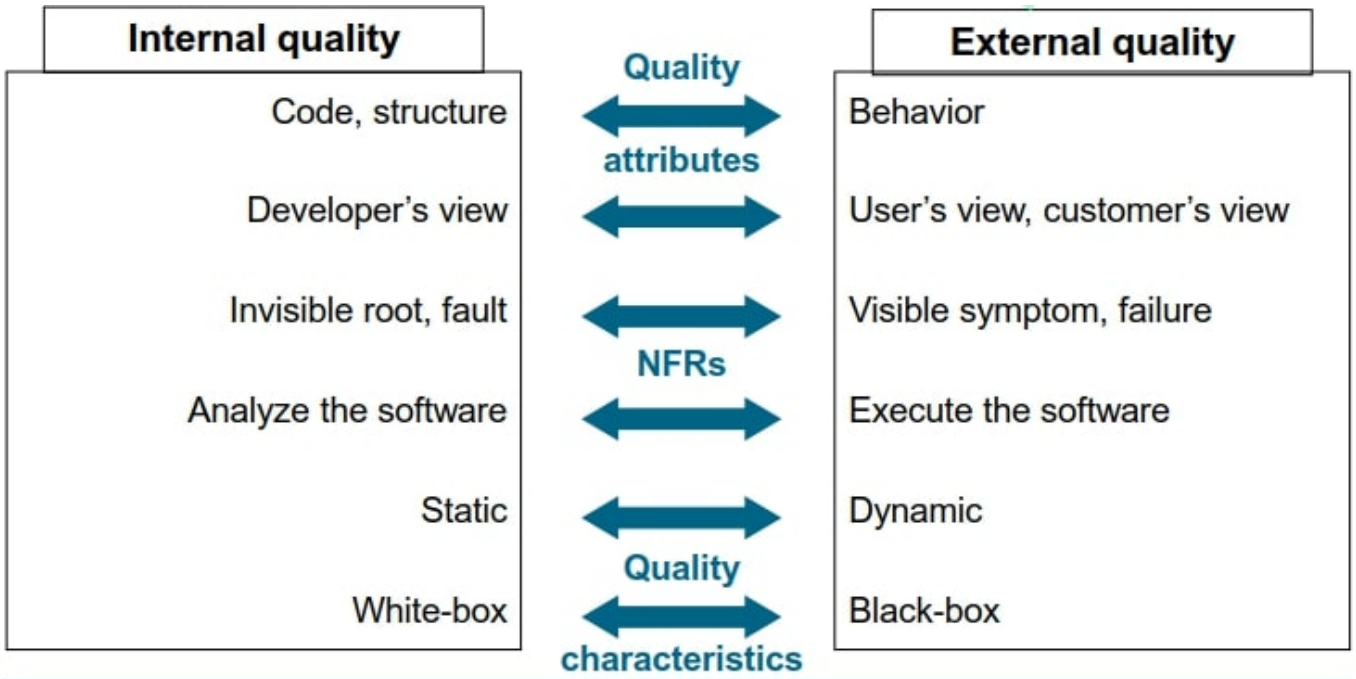
Test suite architecture: test levels



Internal Quality

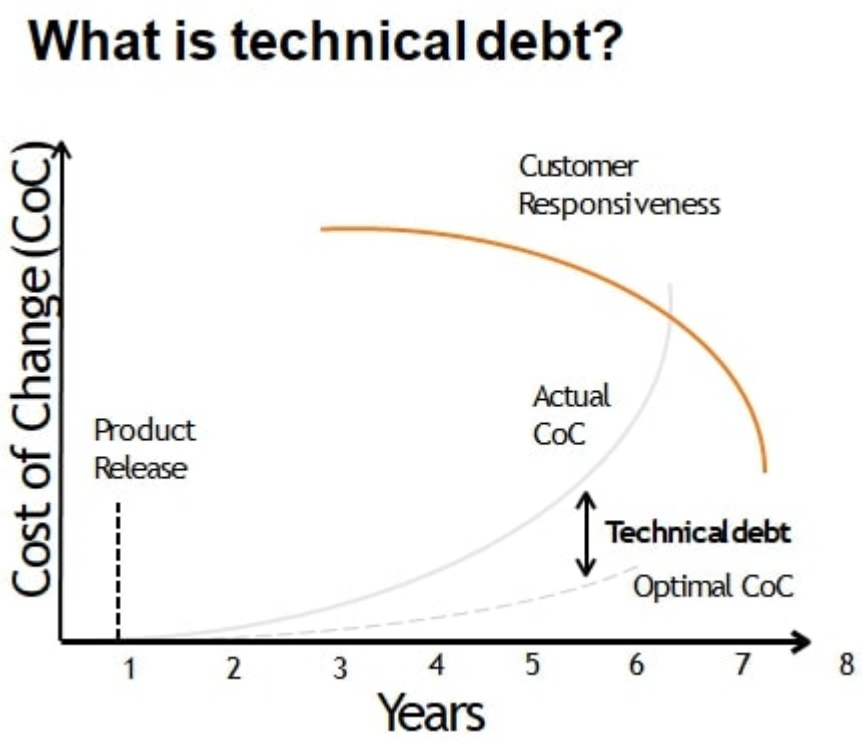
Negative efects:

- 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



Technical Debt

Lack of internal quality reseults in technical debt.

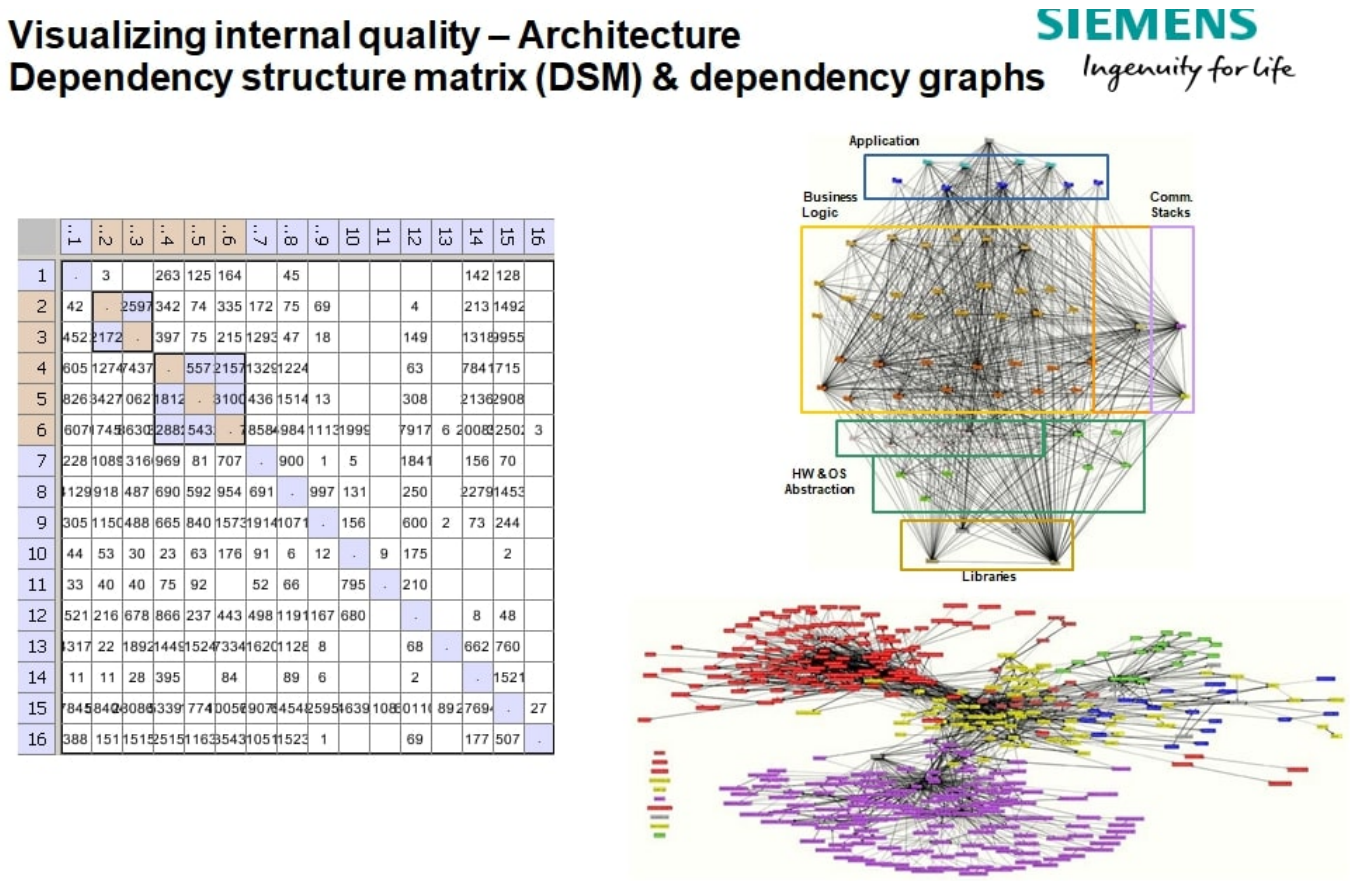


Measuring and Driving Internal Quality

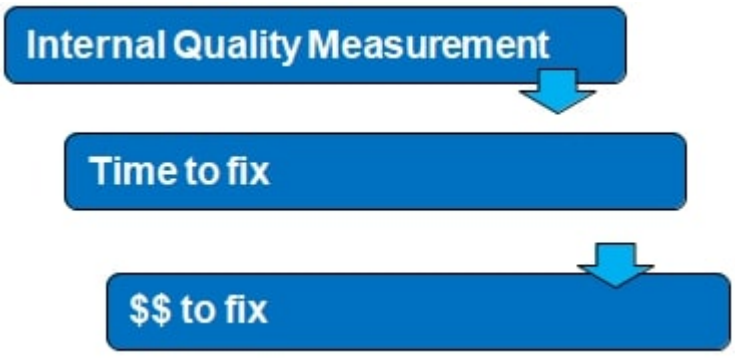
To measure internal quality

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

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



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