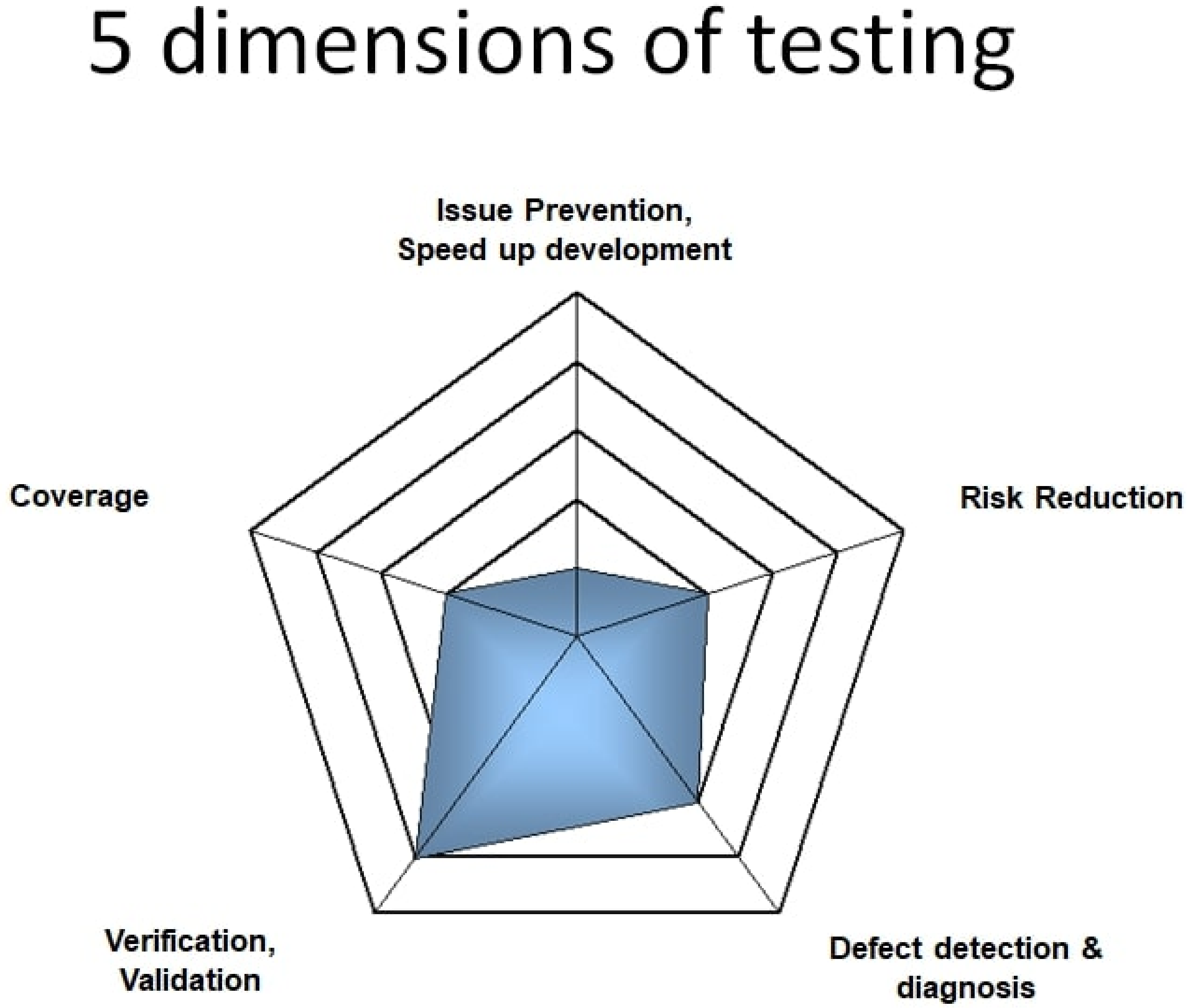
TESTING & QUALITY

Value of Testing

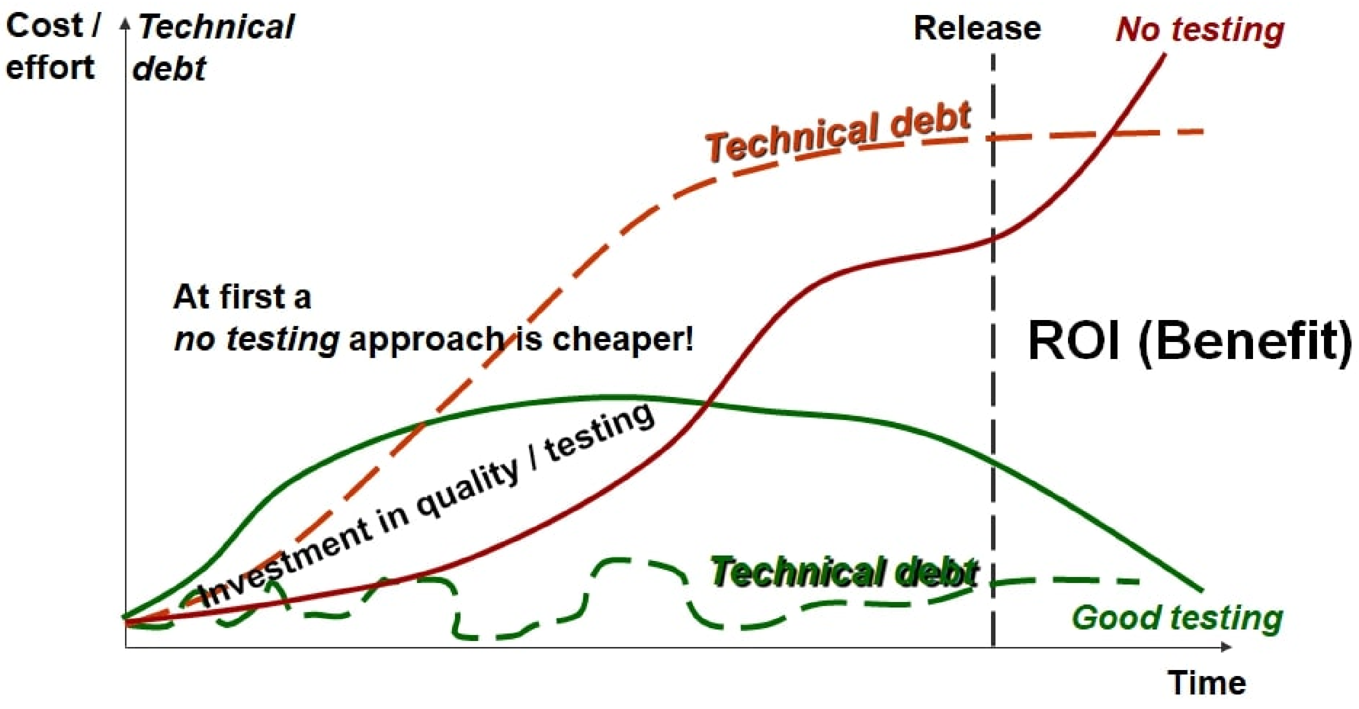
What is Testing? : investigation of the SUT to provide **information** that results in **improvements**.

The 5 Dimensions of Testing

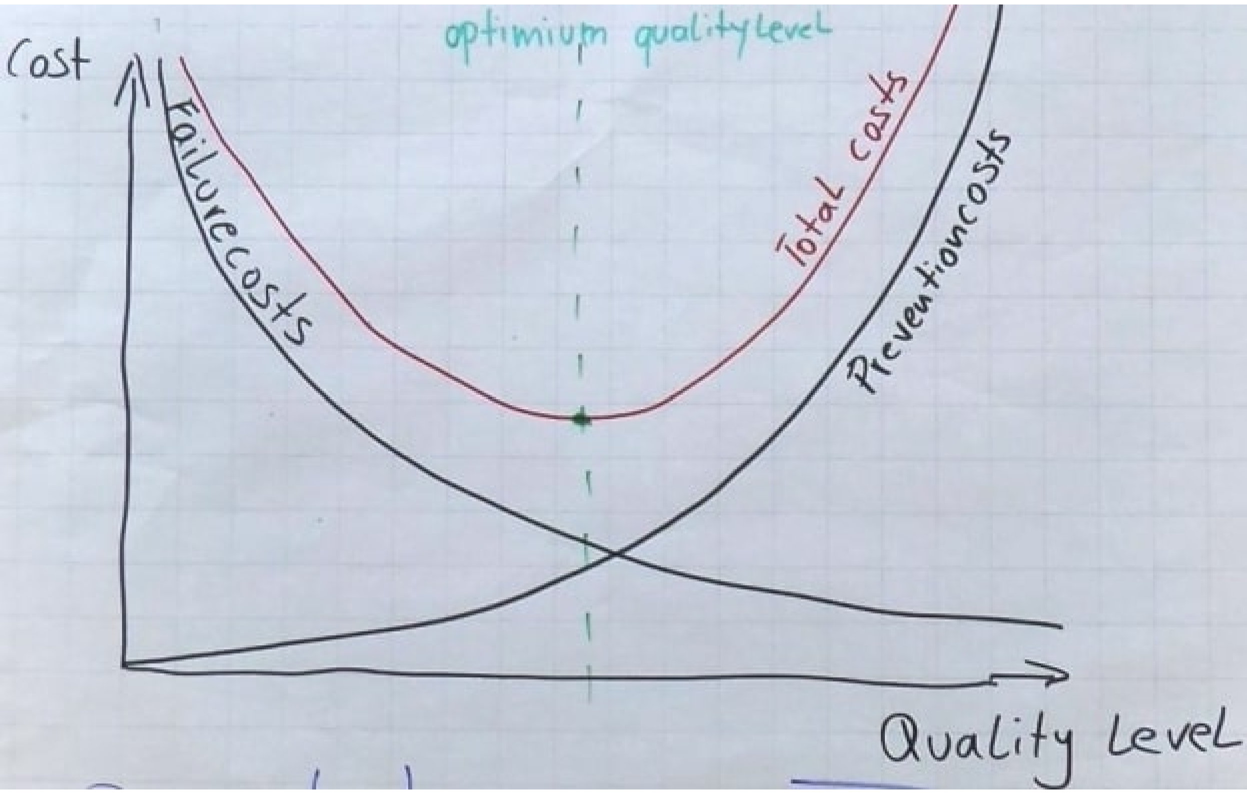
> Coverage: an assessment for the thoroughness or completeness of testing with respect to our test model - *Paul Gerrard*



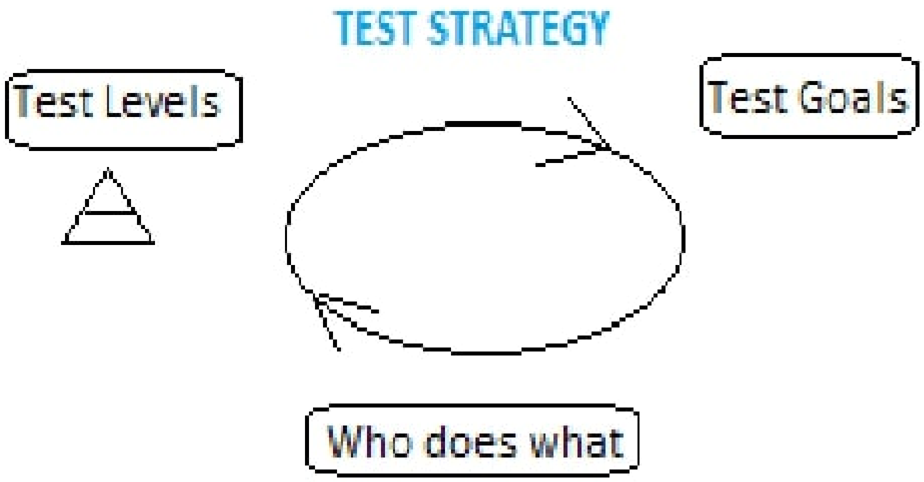
ROI of Testing



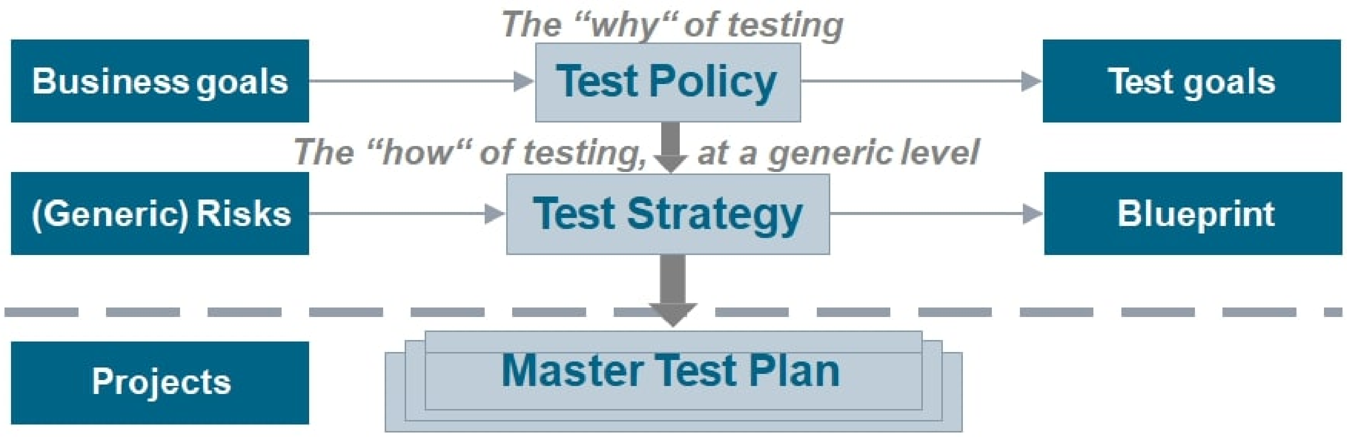
Cost of Quality



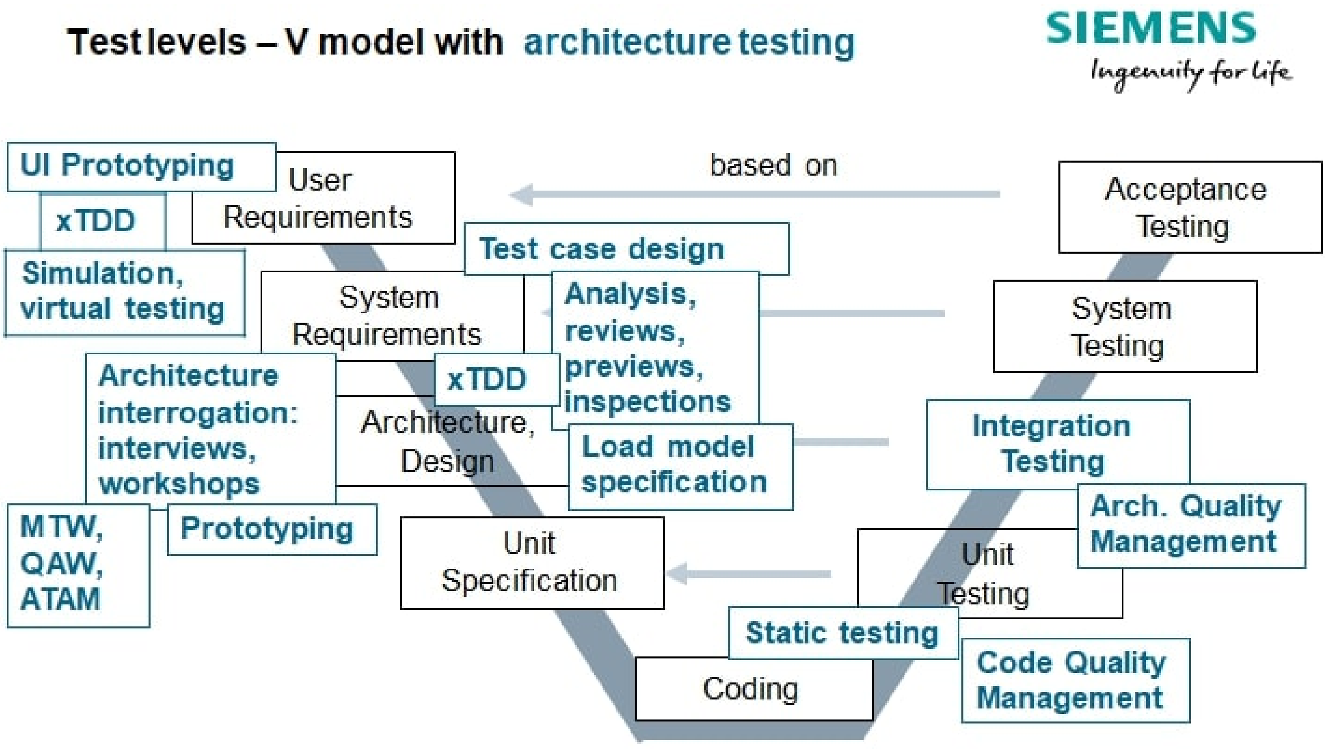
Test Strategy



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

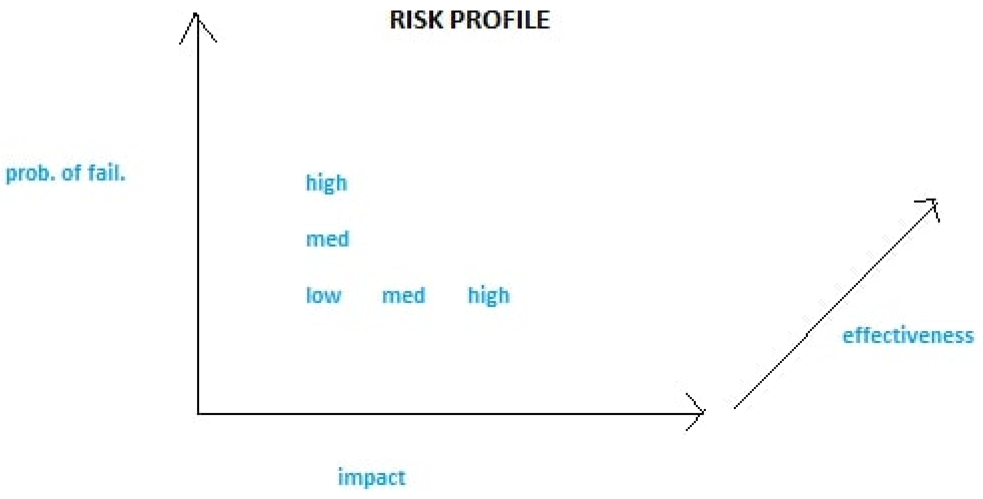


Test levels – V model with architecture testing

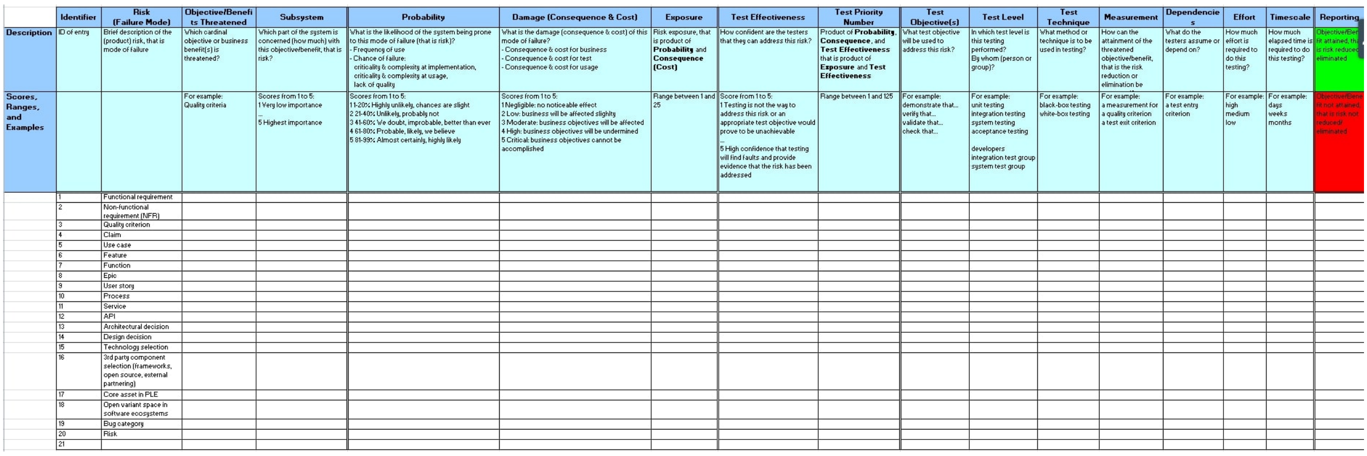


Risk Based Testing

Risk Profile

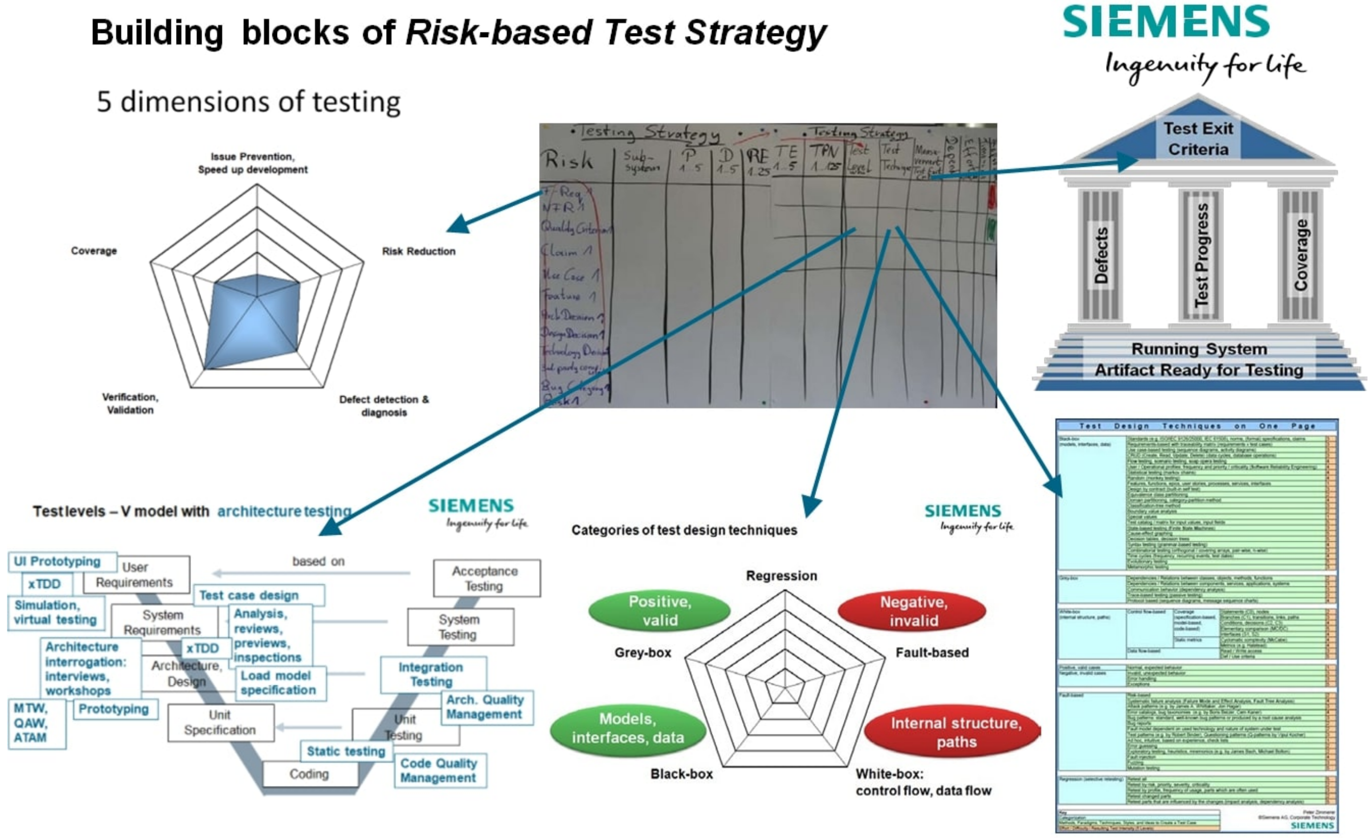


Risk Based Testing Worksheet. You can download the worksheet in xls .



	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/Bene fit attained, tha 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 ever 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 slighlty 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/Benefit not attained, that is risk not reduced/ 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



Building blocks of *Risk-based Test Strategy*

5 dimensions of testing

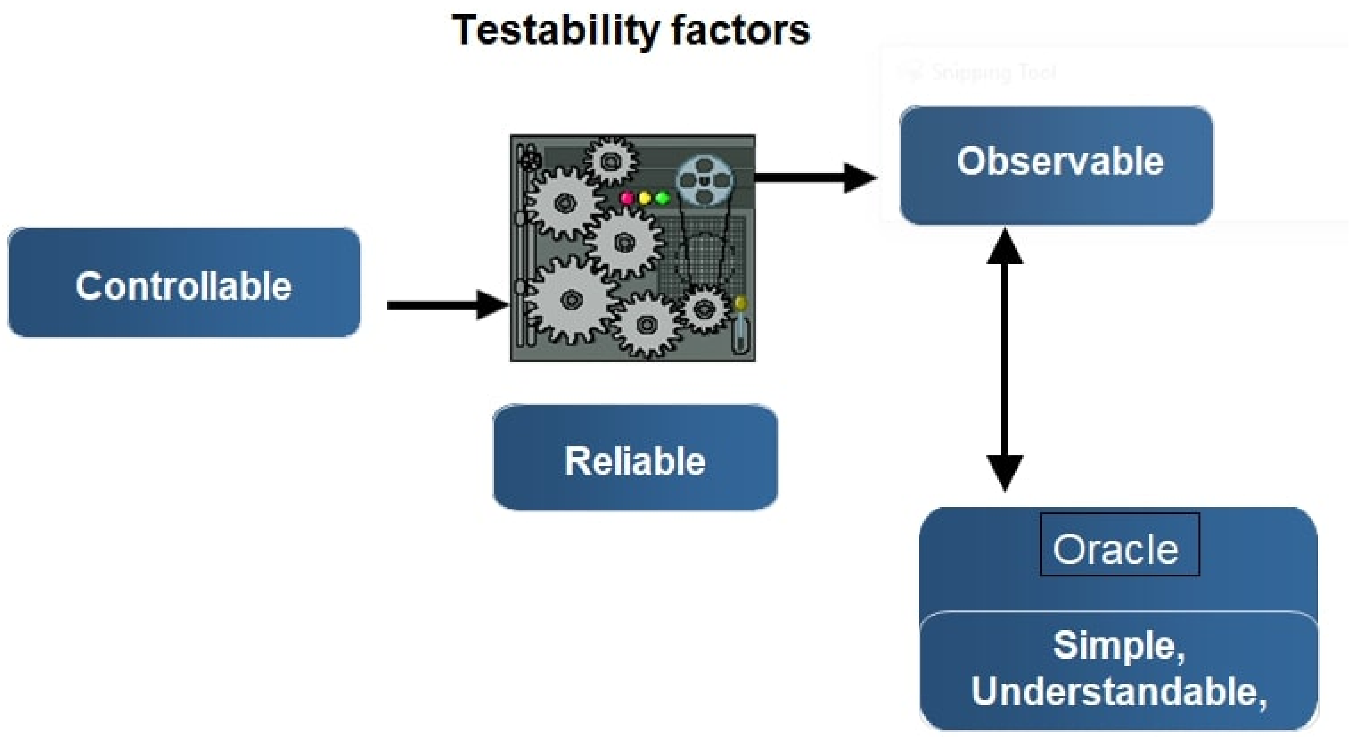
Design for Testability

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.



Testability factors

Controllable

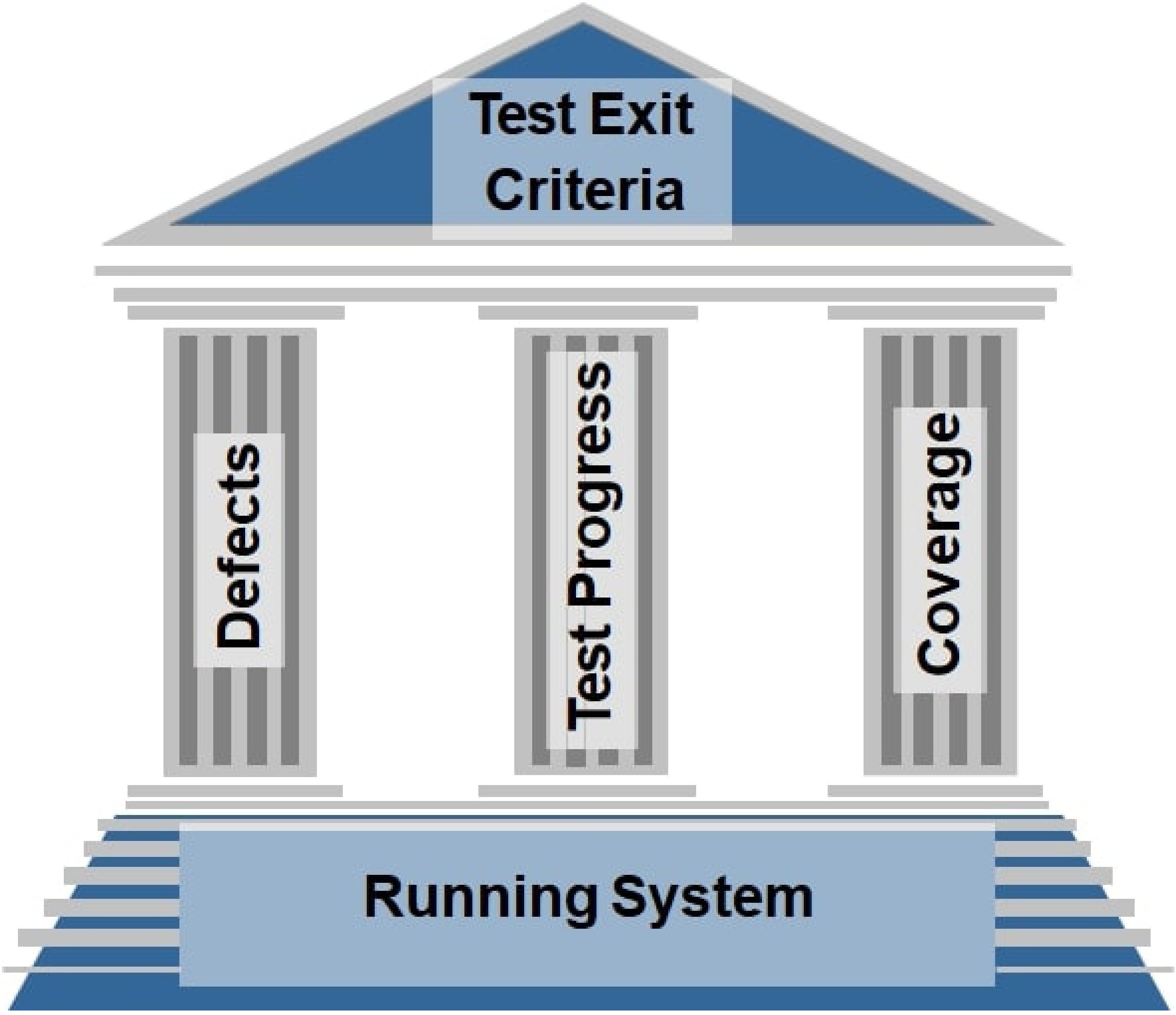
Observable

Reliable

Oracle

Simple, Understandable,

Test Exit Criteria



Test Exit Criteria

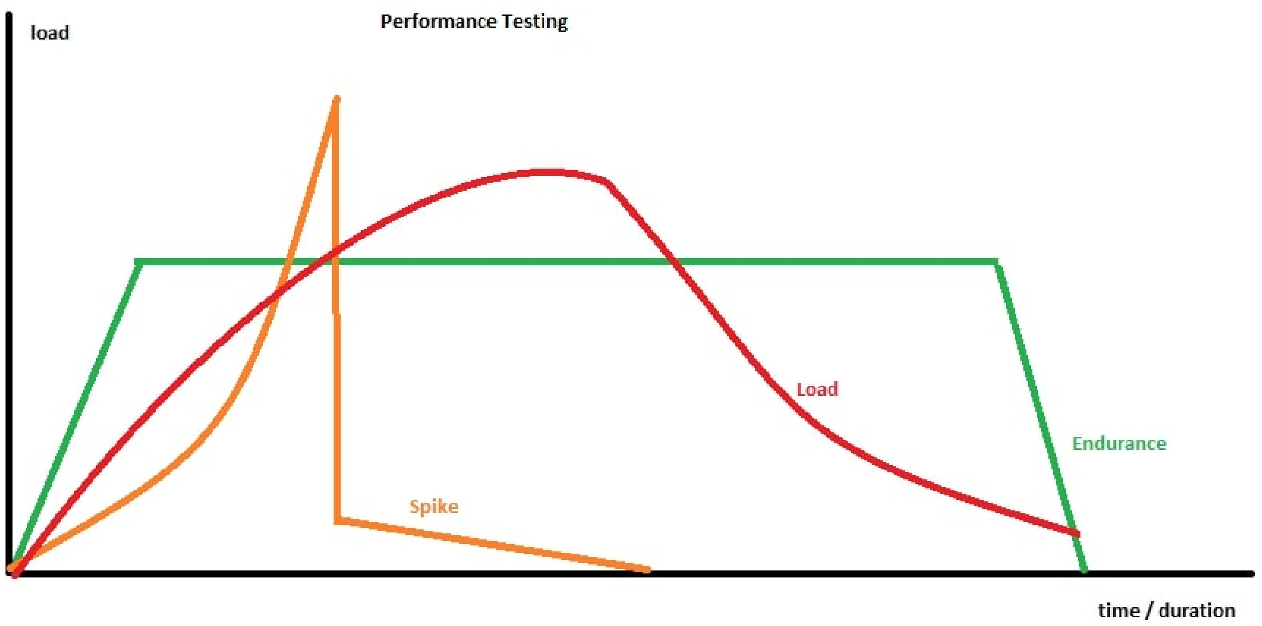
Defects

Test Progress

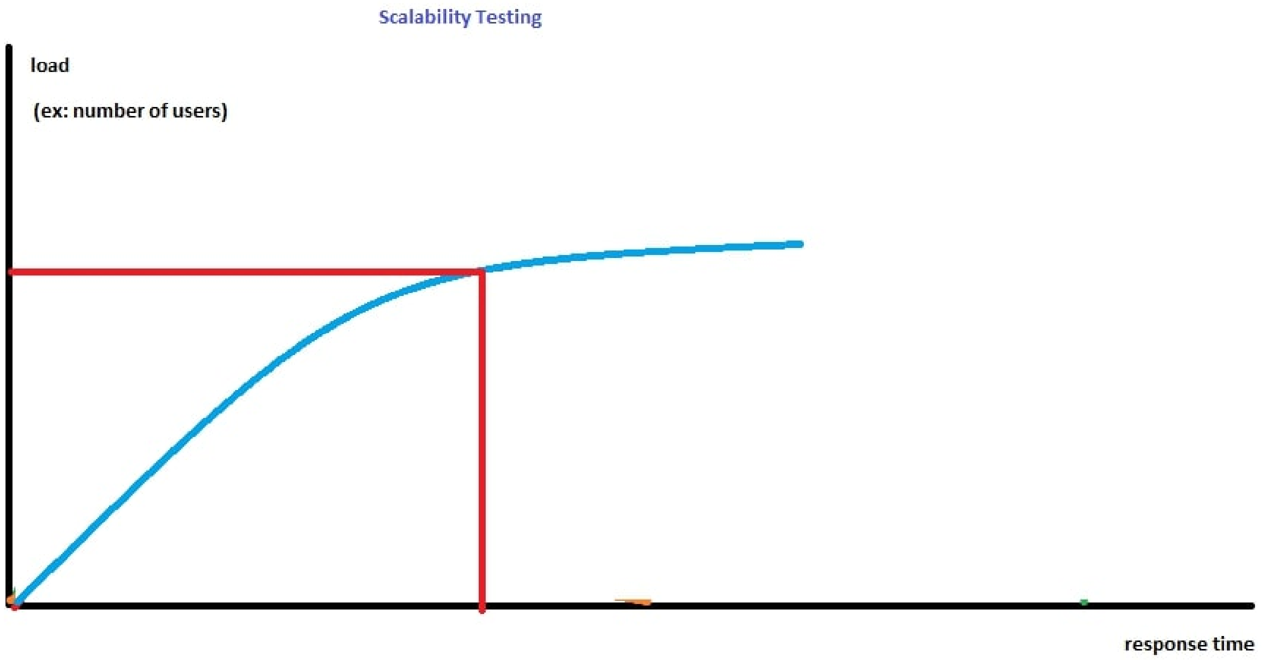
Coverage

Running System

Performance Testing and Scalability



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



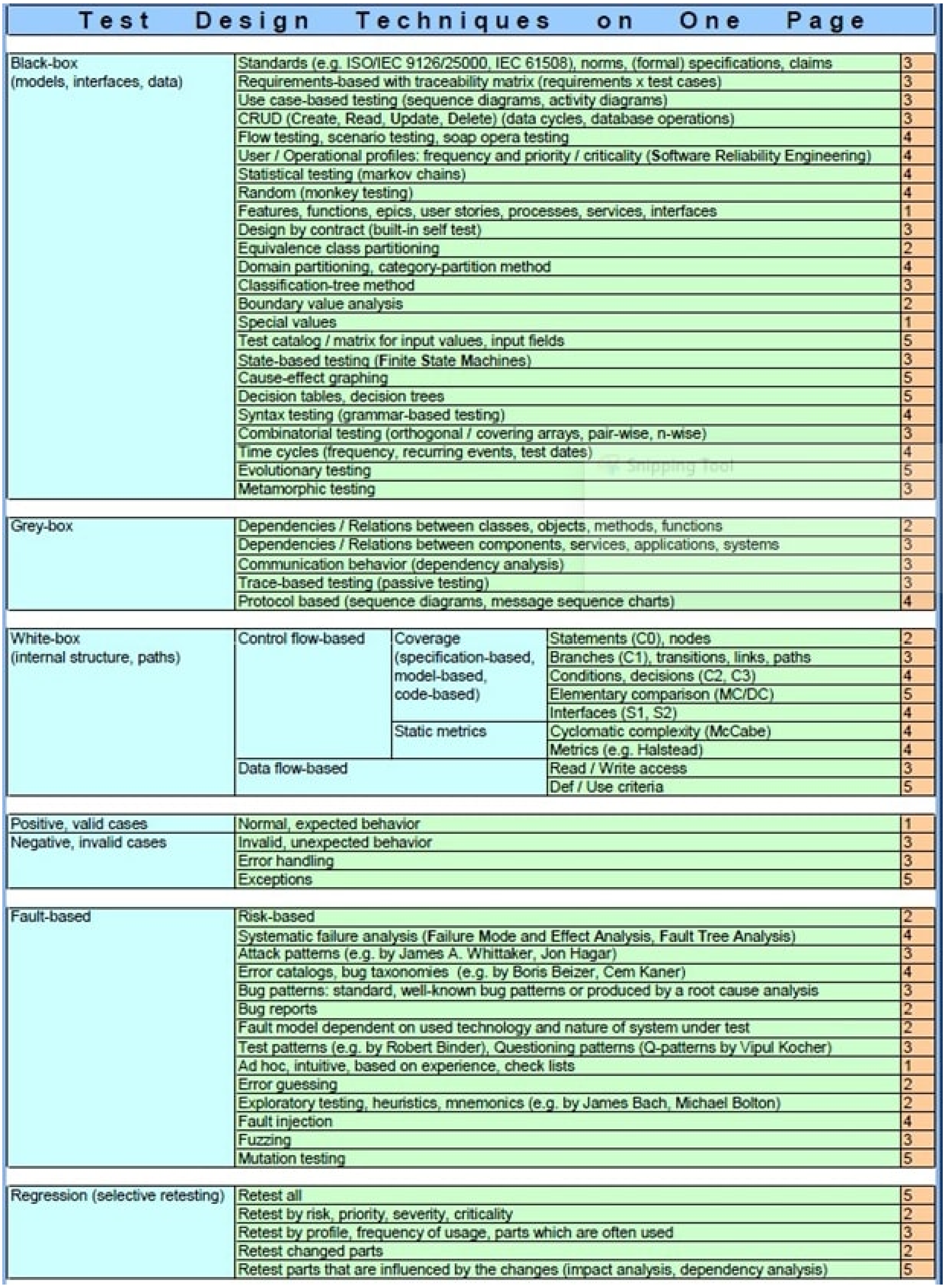
TDD

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

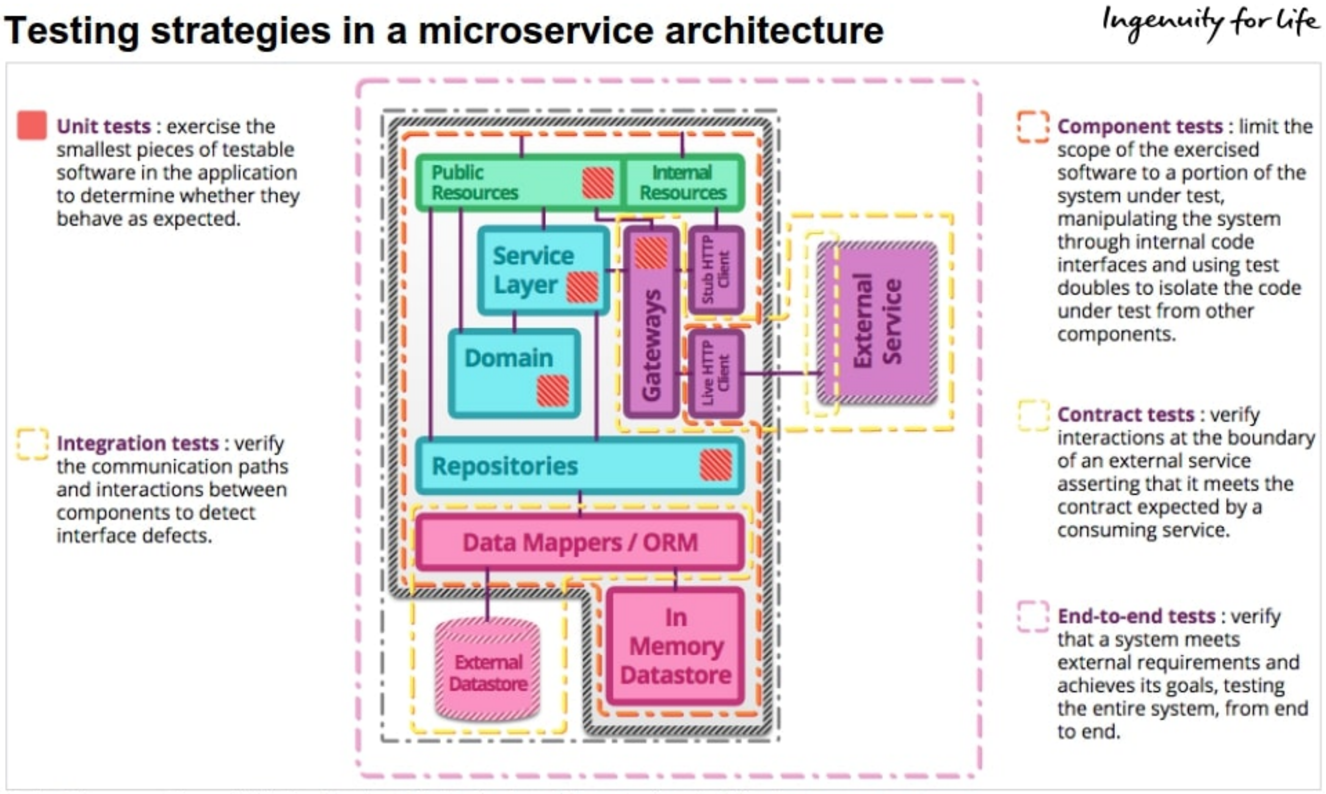
Test Automation Design Patterns paper

Test Environment

Test environment: test rig

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

Test suite architecture: test levels



Testing strategies in a microservice architecture

Ingenuity for life

🟥 **Unit tests** : exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

Integration tests : verify the communication paths and interactions between components to detect interface defects.

Component tests : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

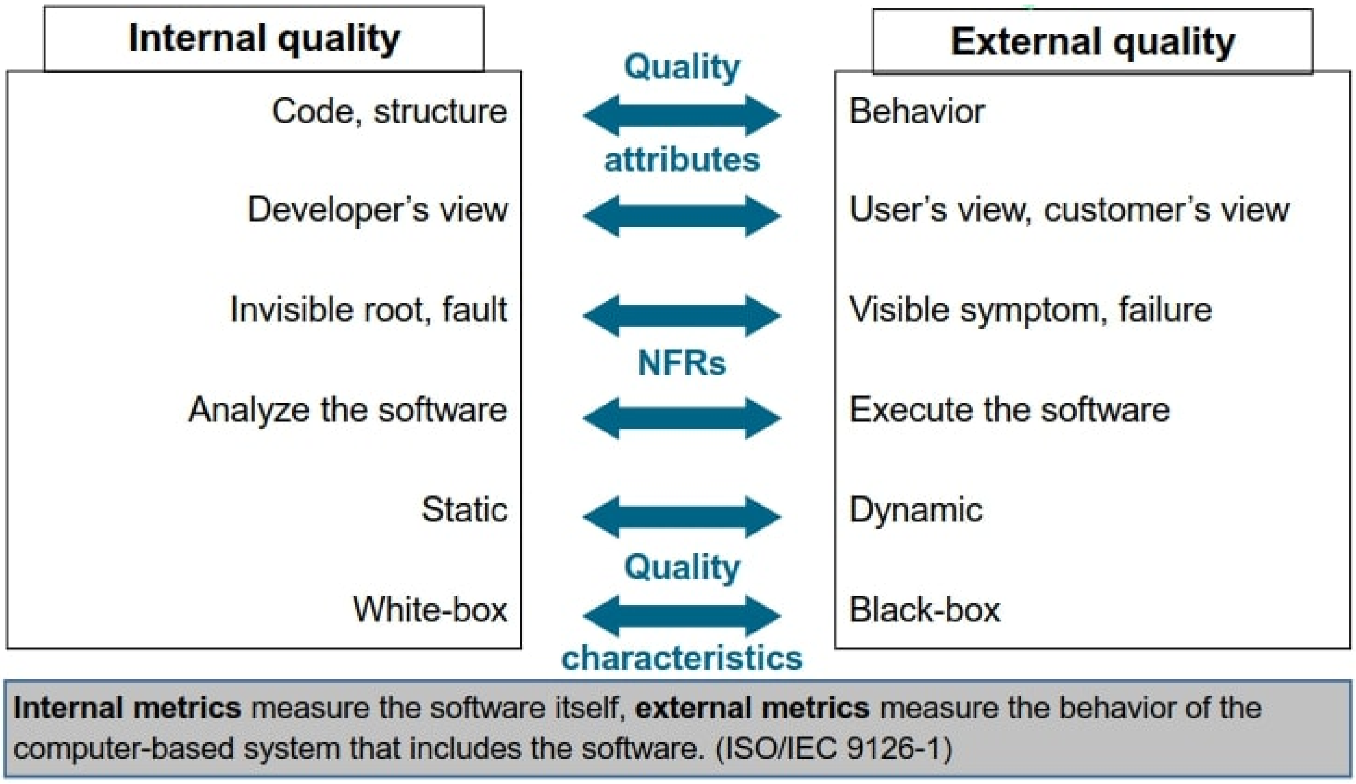
Contract tests : verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.

End-to-end tests : verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.

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

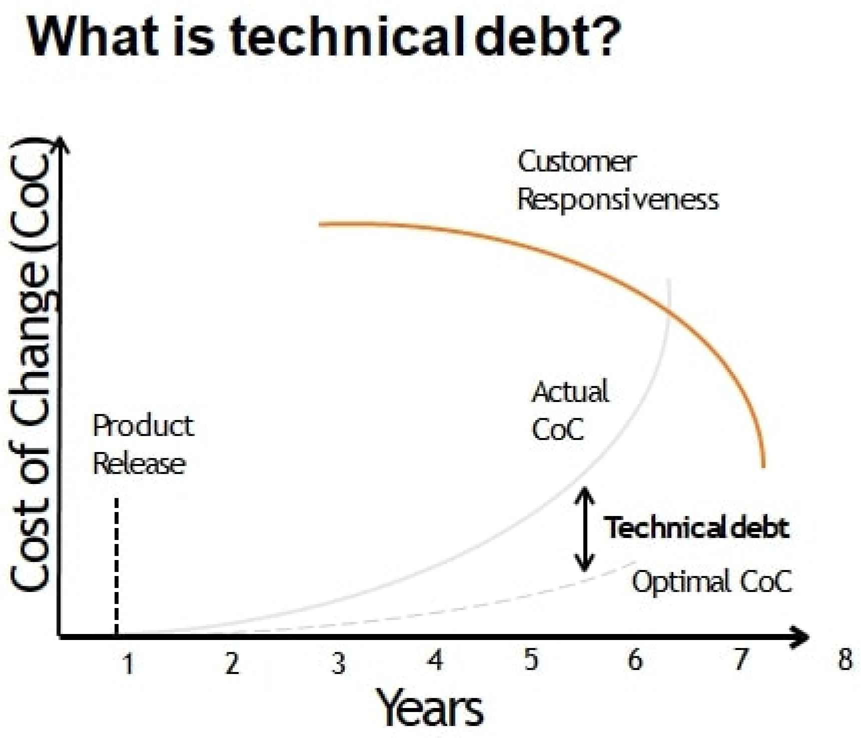


Internal quality	External quality
Code, structure	Behavior
Developer's view	User's view, customer's view
Invisible root, fault	Visible symptom, failure
Analyze the software	Execute the software
Static	Dynamic
White-box	Black-box

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 reseults in technical debt.

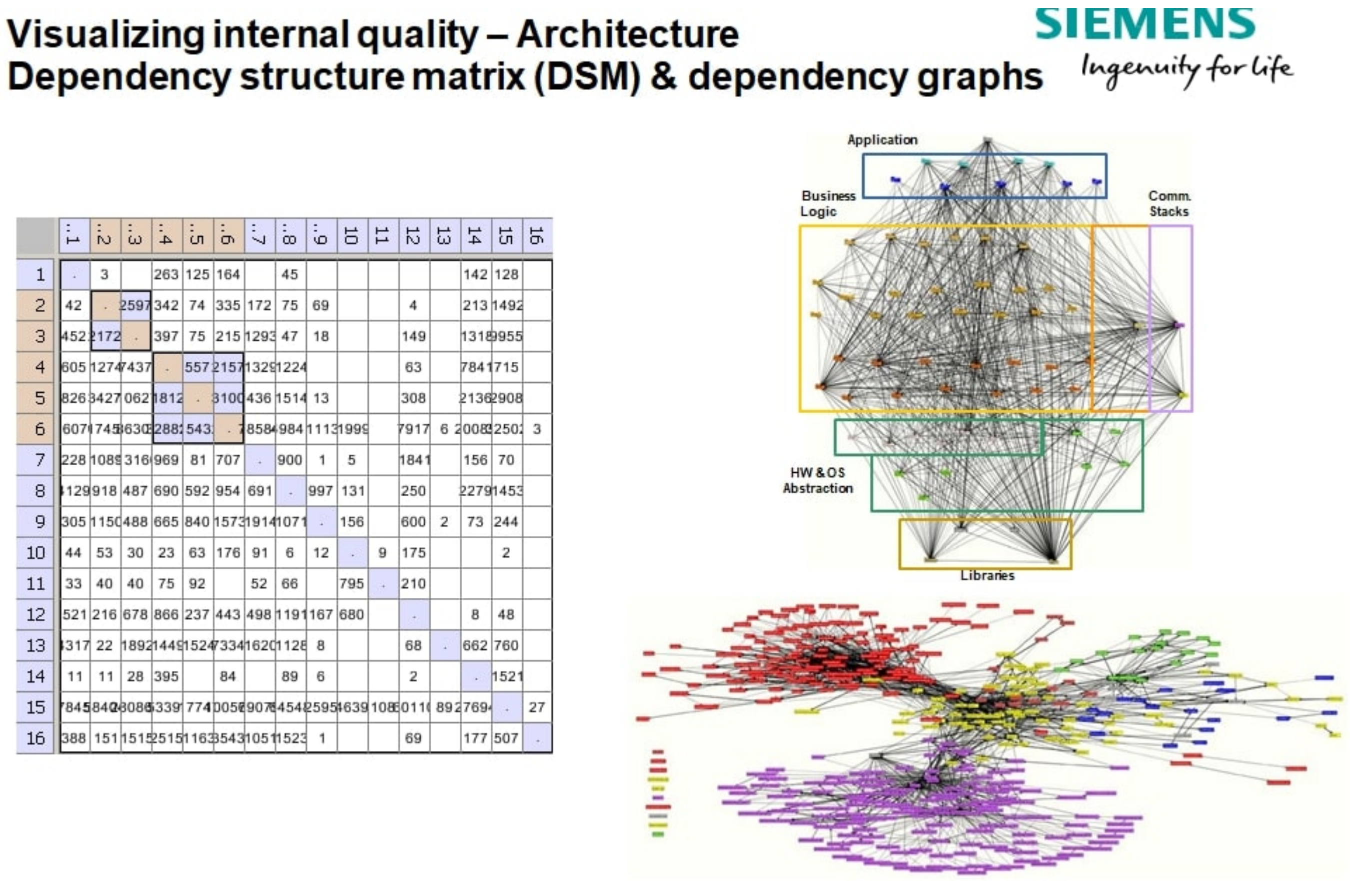


Measuring and Driving Internal Quality

To measure internal quality

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

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

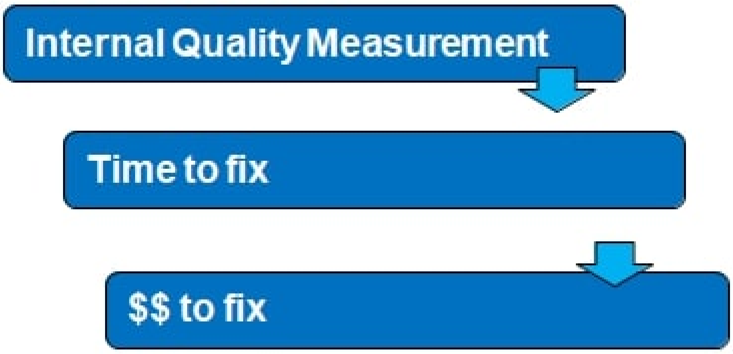


Visualizing internal quality – Architecture
Dependency structure matrix (DSM) & dependency graphs

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	.	2597	342	74	335	172	75	69			4		213	1492	
3	452	2172	.	397	75	215	1293	47	18			149		1318	955	
4	605	1274	7437	.	557	2157	1329	1224				63		784	1715	
5	826	3427	062	1812	.	3100	436	1514	13			308		2136	2908	
6	607	1745	3630	2288	543	.	7858	4984	1113	1999		7917	6	2008	2250	3
7	228	1089	316	969	81	707	.	900	1	5		1841		156	70	
8	129	918	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	1449	1524	7334	1620	1128	8			68	.	662	760	
14	11	11	28	395		84		89	6			2		.	1521	
15	7845	5840	2808	6339	774	0056	9078	4454	2595	4639	108	0110	892	7694	.	27
16	388	151	1515	2515	1163	3543	1051	1523	1			69		177	507	.

To drive internal quality, you must monetize it:



Internal Quality Measurement

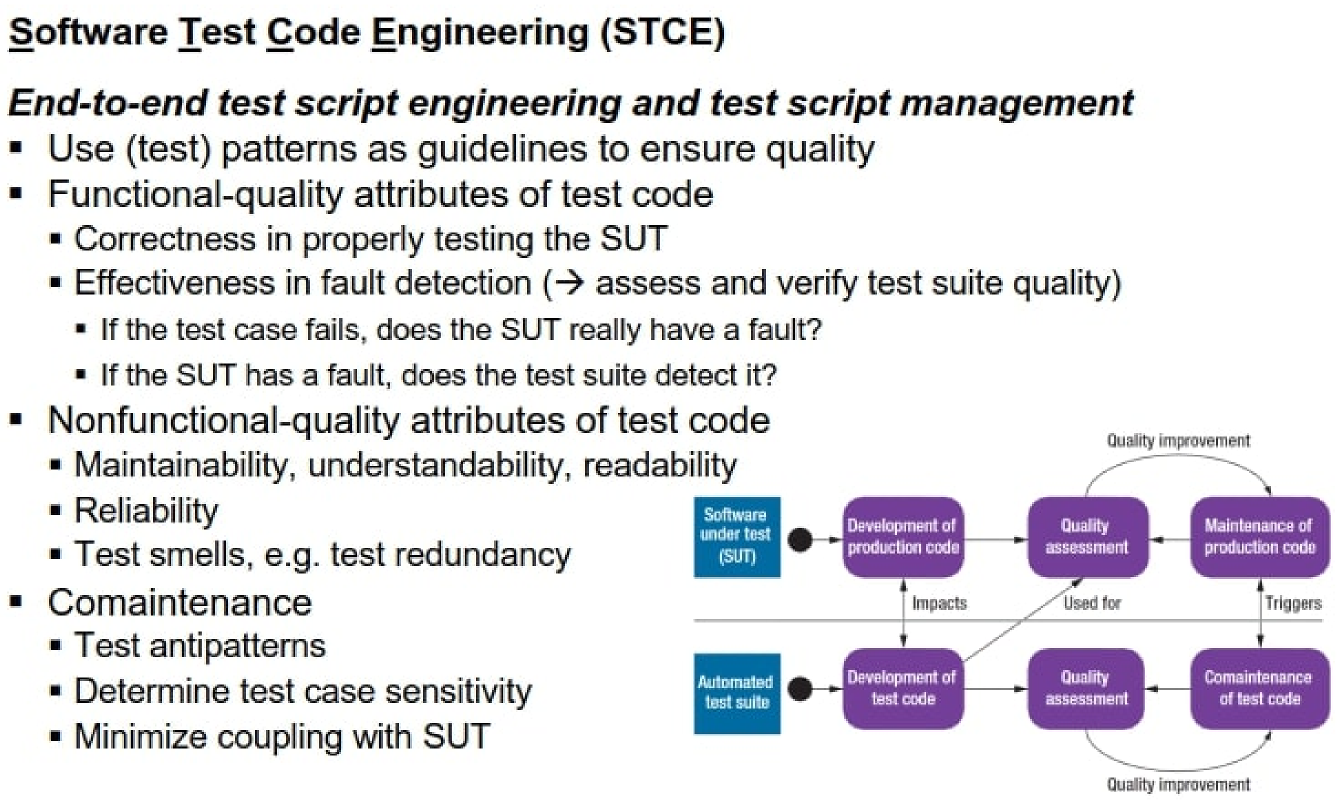
Time to fix

$$ to fix

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