Fundamentals of Software Architecture
DIT344

Sam Jobara Ph.D.
jobara@chalmers.se
Software Engineering Division
Chalmers | GU

Dr. Sam Jobara
Chalmers | University of Gothenburg

I have a Ph.D. in Computer Science and Engineering, USF, USA

I have research interests in testing and fault modeling, computer architecture, information security, and product line engineering. I am also interested in learning and cognitive theories.

My teaching covers a multitude of courses in Computer architecture, Information security, TDA594 Software engineering principles for complex systems, & DIT824 Software engineering for data-intensive AI applications.

My industrial experience spans over 18 years as an IT Consultant in Telecommunication, Information Security, and Supply Chain Management.

jobara@chalmers.se

Software Architecture
DIT344

Requirements, & Quality Attributes

Dr. Sam Jobara
Chalmers | University of Gothenburg

References

- *Object-Oriented Software Engineering,* Chapter 4

  by Lethbridge & Laganiere

- Software Architecture in Practice, Chapter 3 & 4

  by Len Bass; Paul Clements; Rick KazmanPublished

- Other literatures (see reference at slides)

Objectives

Learning experience for this lecture:

- Problem statement and Systems Requirements
- How to capture and document system requirement
- Introduce System quality attributes
- How to define, and measure quality attributes

Outline

Concepts alignment

Software Architecture Contexts

System Requirements

System Components & Layers

Use Cases Brief

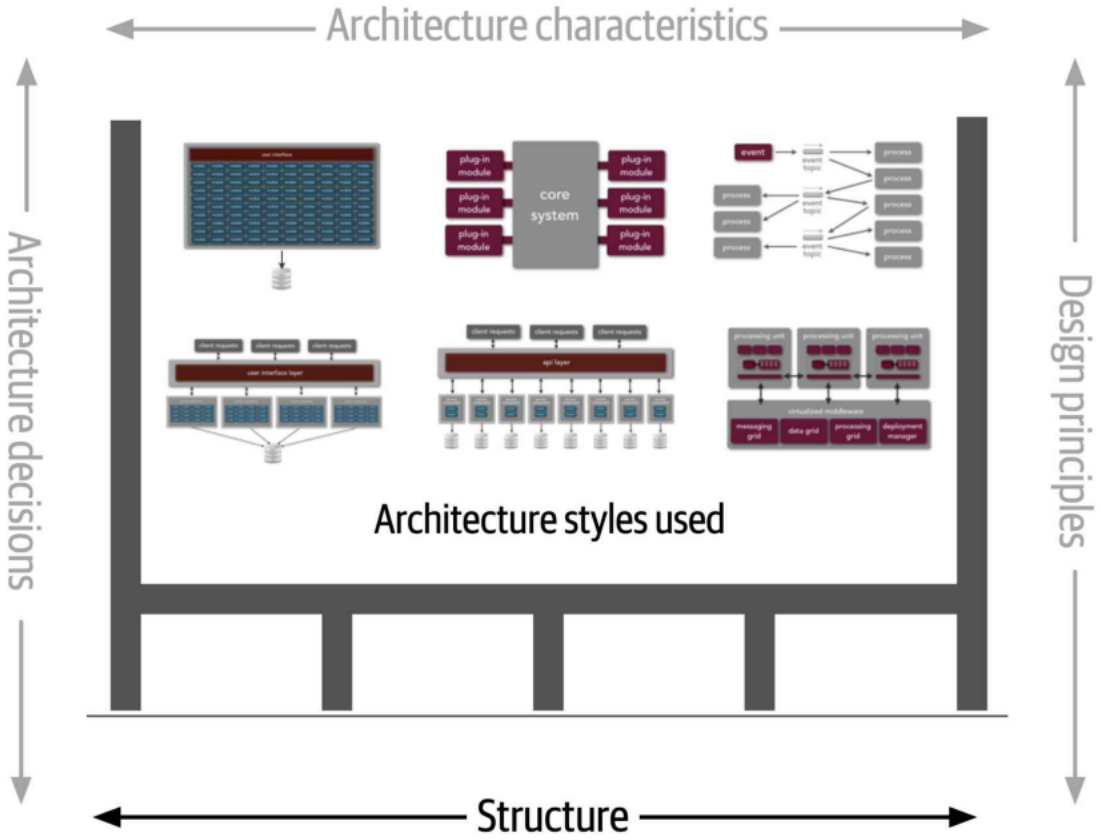
Quality Attributes

QA Scenarios

Concepts Alignment

System Architecture

It consists of the Structure (Style) of the system , driven by architecture characteristics, design principles, and architecture decisions

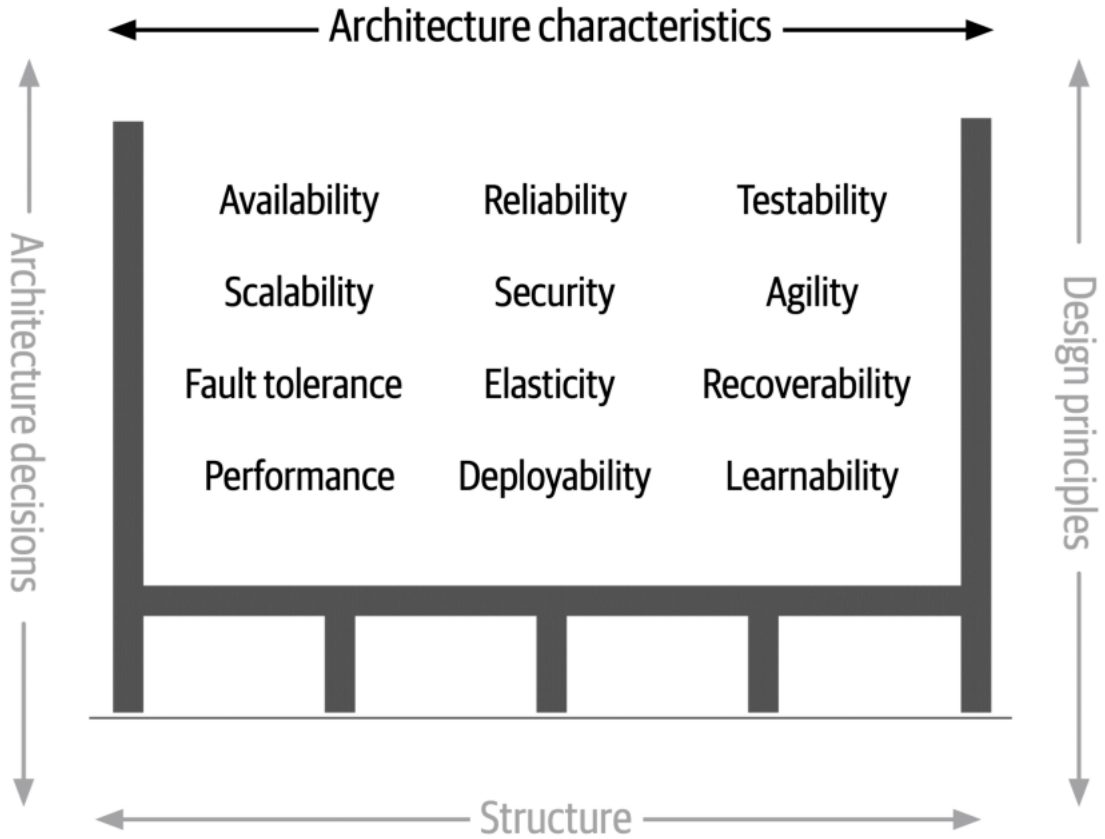


Style, Structure or pattern (such as client-server,P2P microservices, layered)

Concepts Alignment

Architecture characteristics (Attributes)

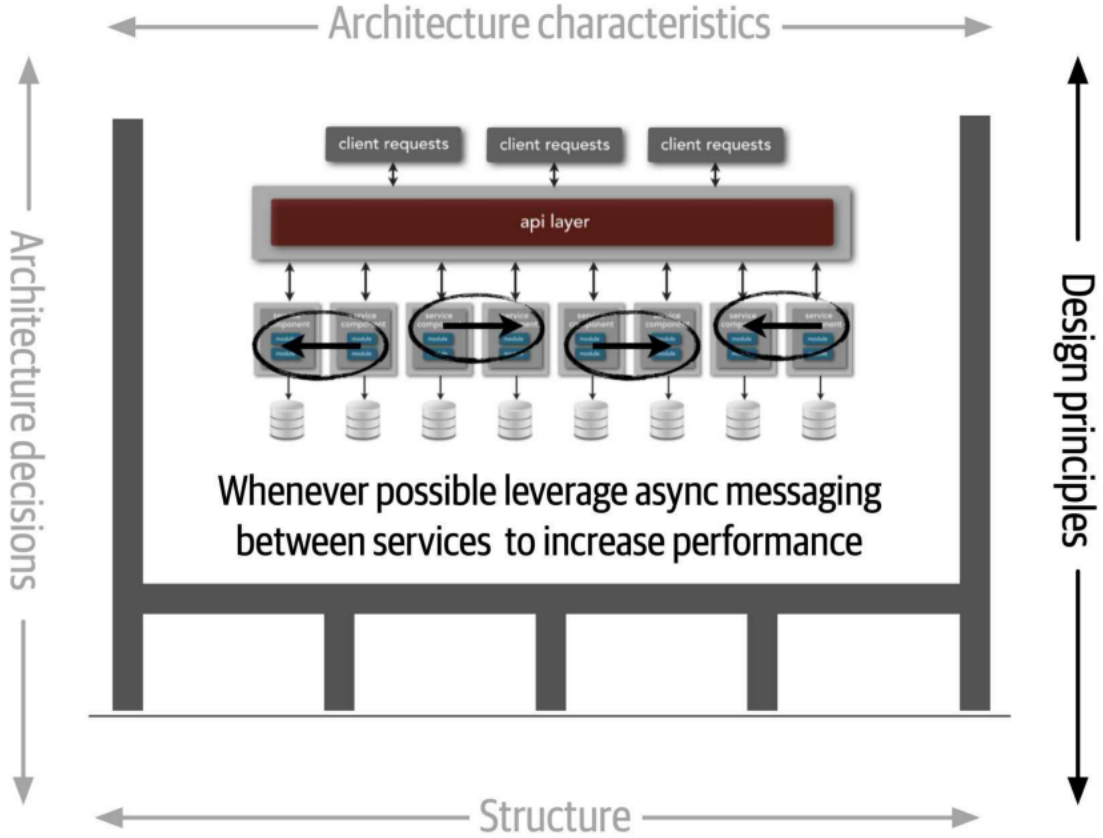
System quality attributes which encompasses all -ilties



Concepts Alignment

Design Principles

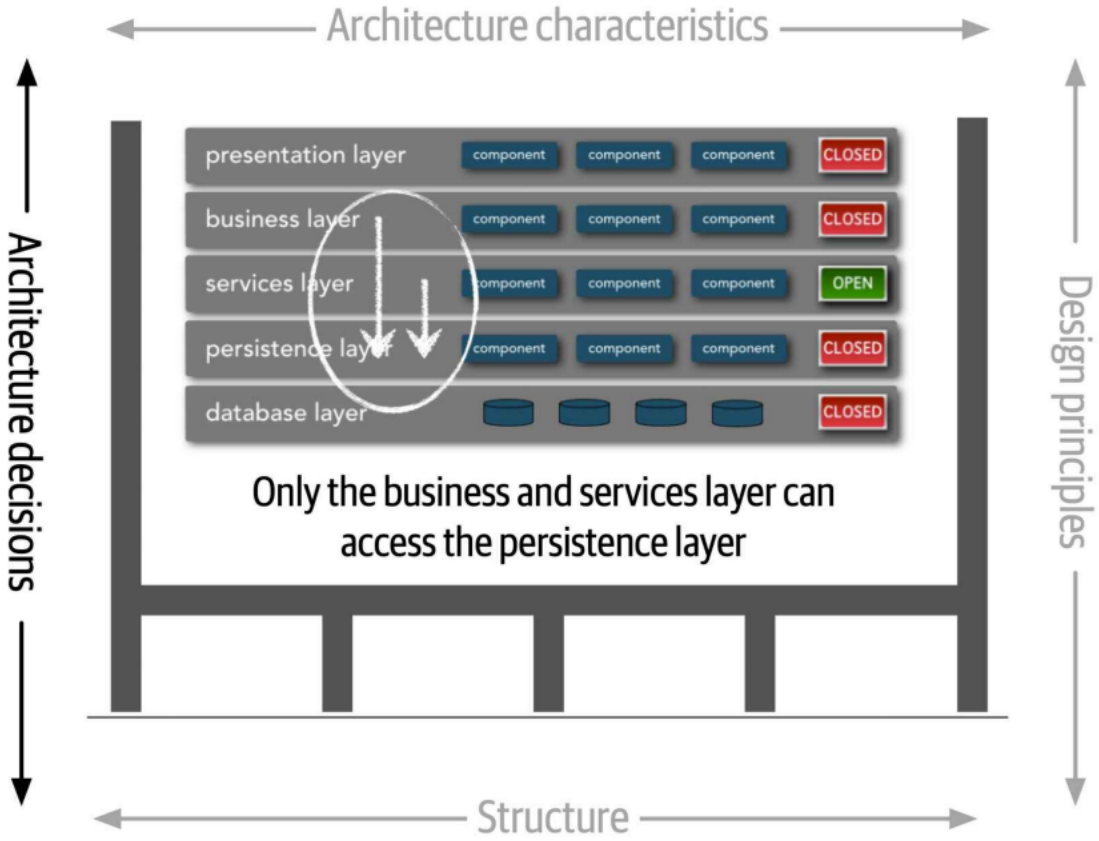
A design principle is a guideline rather than a hard-and-fast rule



Concepts Alignment

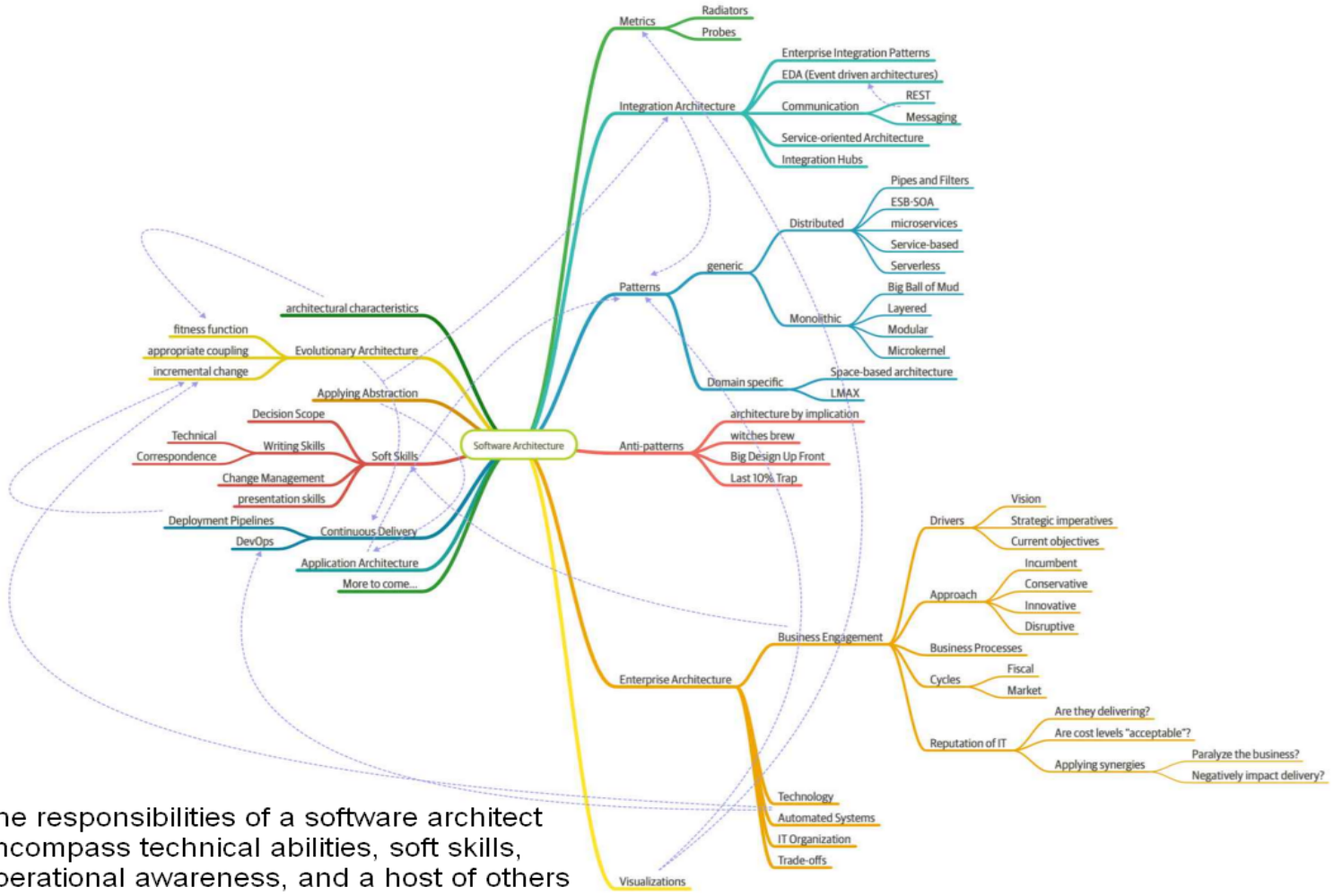
Architecture Decisions

Define the rules for how a system should be constructed



Mind Map for Software Architecture disciplines*

Architects need some knowledge and expertise as shown in the mind map, and this is not an inclusive map!

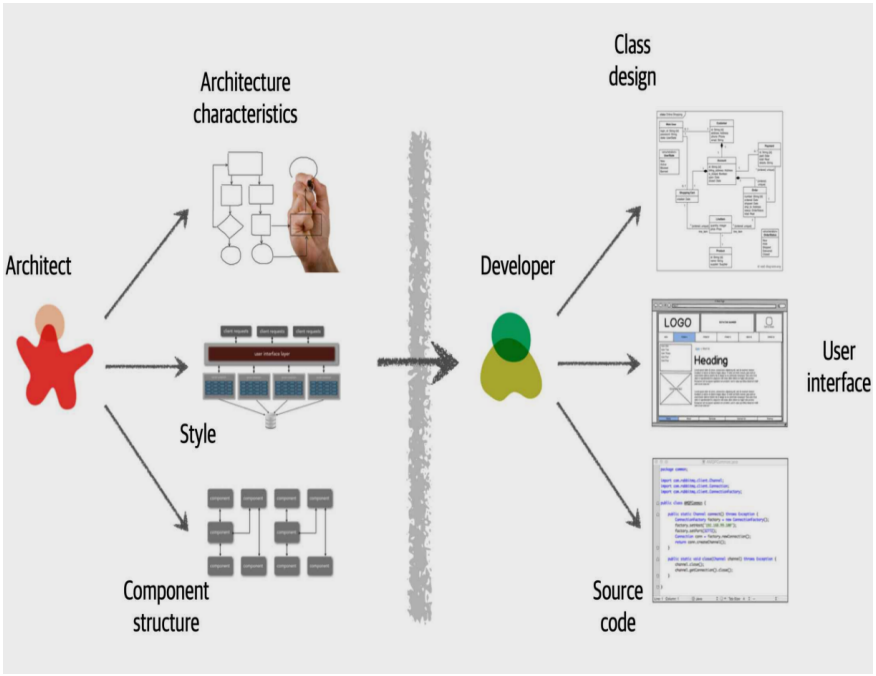


The responsibilities of a software architect encompass technical abilities, soft skills, operational awareness, and a host of others

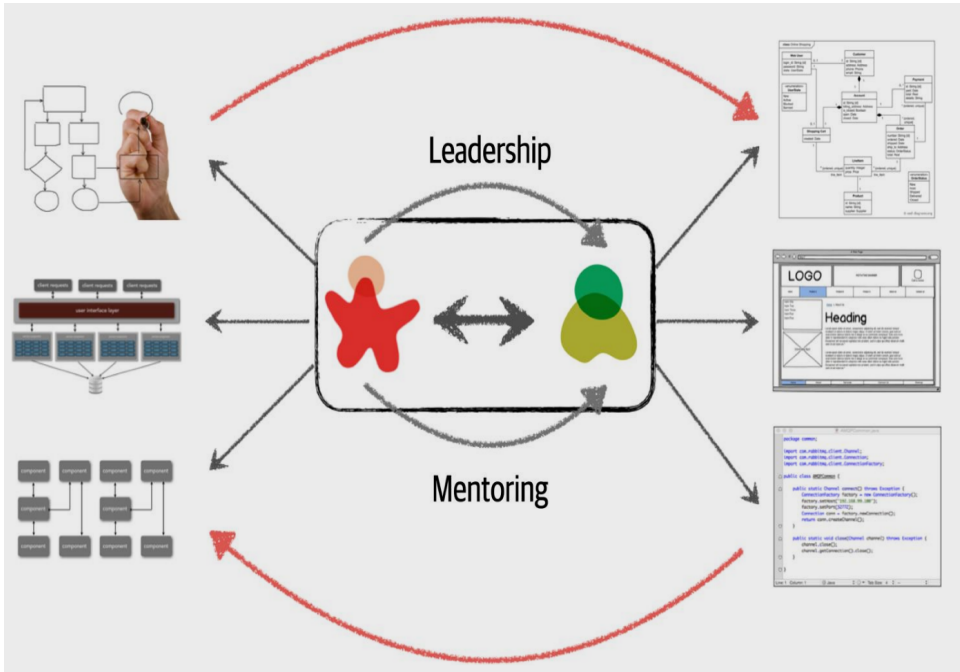
* Richards, Mark; Ford, Neal (2020). Fundamentals of Software Architecture

Architect vs. Developer

- A tight collaboration between the architect and the development team is essential.
- See details in the DevOps guest lecture



Old way



New way

Learning Unit Obligation

- Understand the 4 component of System Architecture and their relationship to each others.
- Realise the need for a close collaboration between architects and system developers.

Outline

Concept alignment

Software Architecture Contexts

System Requirements

System Components & Layers

Use Cases Brief

Quality Attributes

QA Scenarios

Software Architecture Contexts

ARCHITECTURE IN A TECHNICAL CONTEXT

Inhibit or enable the achievement of a system's technical quality attributes.

ARCHITECTURE IN A PROJECT LIFE-CYCLE CONTEXT

1. Waterfall: Plan Do Check Act
2. Iterative: Short cycles of PDCA
3. Agile: It includes Lean, Scrum, XP, Kanban, DevOps.

These frameworks are all incremental and iterative.

They emphasize close collaboration between stakeholders.

ARCHITECTURE IN A BUSINESS CONTEXT

Many business goals will be manifested as quality attribute requirements as Business needs and wants, budget, market and more.

Software Architecture Contexts

ARCHITECTURE IN A PROFESSIONAL CONTEXT

- Diplomacy with stakeholders
- Communication (RACI)
- Consultation & Negotiation
- Economy & Budget sense
- Legal and regulatory issues
- Market knowledge
- Up to date technology
- Learning organization enabler

Learning Focus

- Understand the architecture contexts and how they impact system development

- Understand the nature of agile project life-cycle context of system development

- Realize the obligation of system architects professional career development

Outline


Concept alignment


Software Architecture Contexts


System Requirements


System Components & Layers


Use Cases Brief


Quality Attributes


QA Scenarios

System Requirement

What is a requirement?

Definition:

a requirement is a statement (Scope of Work SOW) describing:

1. *an aspect of what the system functions must do,*
2. *a constraint on the system's development*
3. *adequately solving the stakeholders' problem*

System Requirement

Types of requirements

Functional requirements: describe **what** the system should do

Non-Functional requirement: deal with **how** system attributes perform.

Requirements can be tricky!

Regulatory, liability, budgetary, timely, leverage, skills, etc

System Requirement

The functional requirements: WHAT as follows:

- What I/O the system should expect
- What data structure and medium to use.
- What computations the system should perform
- What is the User and system admin functions

Non-Functional Requirement

The attributes “HOW”, to be covered later

System Requirement

Constraints

Constraints a set restrictions on how the user requirements are to be implemented.

- Interface APIs Requirements.
- Communication (protocols) Interfaces.
- Hardware Interfaces.
- Software compatibility & Interfaces.
- User Interfaces & experience
- Language, code, and reusability
- Testing and maintenance

System Requirement

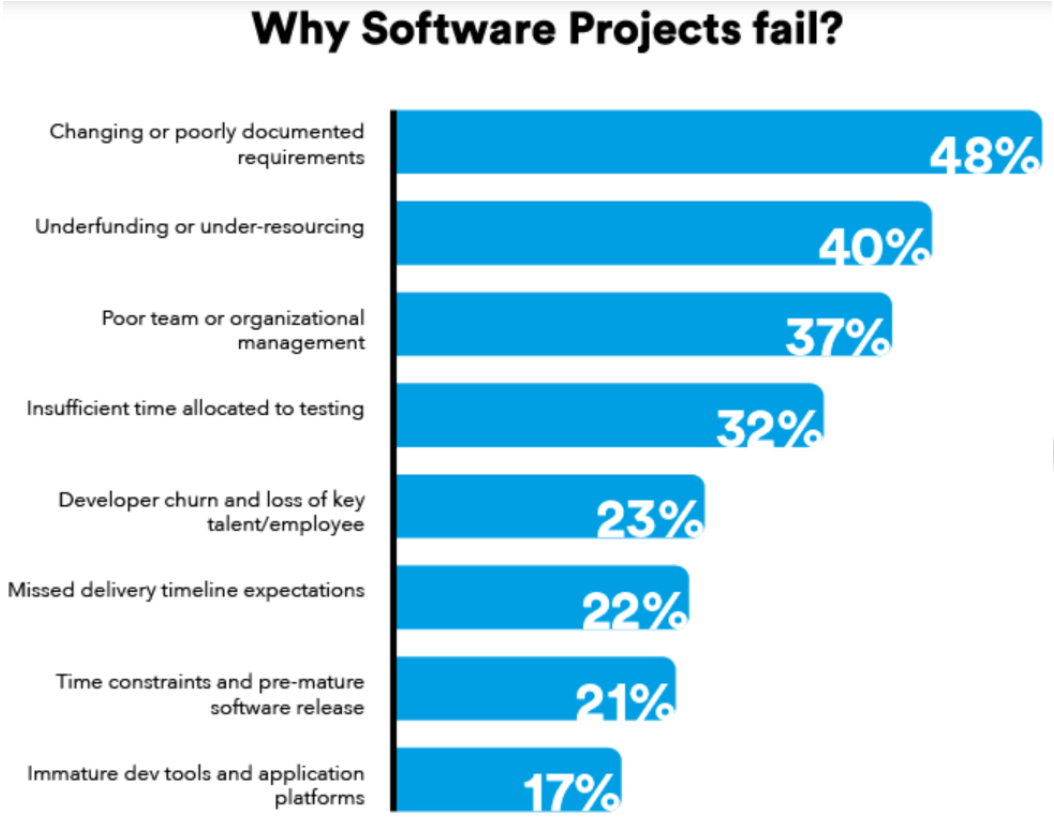
Do customers know what they want?

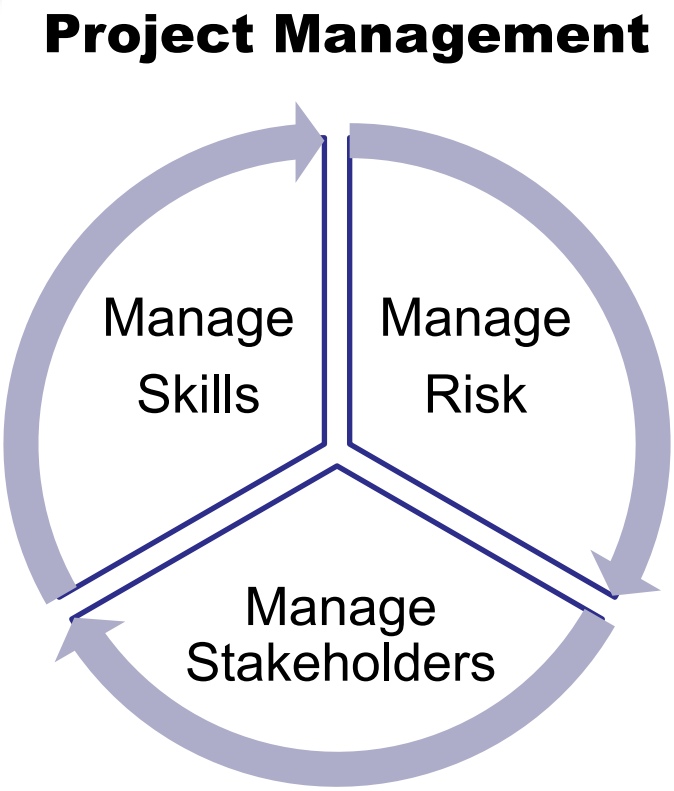
Stakeholders engagement issue

What about scope creep!

System Requirement

How they fail?





System Requirement

Understand problem and setting the scope

Start Domain analysis to understand the background of the project.

Careful attention to the problem statement.

are we adequately solving the problem? (BCDR project)

It is a good idea to define the problem and scope as early as possible

(but consider Agile cycles)

It is very important to define the stakeholders early with detailed RACI matrix.

Clear RACI helps allot in facilitating communication, collaboration and accountability

System Requirement

R: Responsible — Person assigned to do the task
A: Accountable — Person makes final decisions and own task
C: Consulted — Person that should (must) be consulted for the task
I : Informed — Person to be informed when a decision or action is made

RACI Matrix Template

	Project Leadership					Project Team Members					Project Sub-Teams					External Resources				
Project Deliverable (or Activity) / Role	Executive Sponsor	Project Sponsor	Steering Committee	Advisory Committee	Role #5	Project Manager	Tech Lead	Functional Lead	SME	Project Team Member	Developer	Administrative Support	Business Analyst	Role #4	Role #5	Consultant	PMO	Role #3	Role #4	Role #5
Initiate Phase Activities																				
- Submit Project Request	A/C	R/A				R/A	A/C	A/C	C											
- Request Review by PMO						R											A			
- Research Solution	I	C				R/A	A/C	A/C	C				C			C	A/C			
- Develop Business Case	I	A/C	I	I		R/A	C	C	C				C			C	C			
Plan Phase Activities																				
- Create Project Charter	C	C				R/A	C	C	C				C			C				
- Create Schedule	I	I	I	I		R/A	C	C	C	C	C	C	C			C	I			
- Create Additional Plans as required	I	I	I	I		R/A				I	I	I	I			C	I			
Execute Phase Activities																				
- Build Deliverables	C/I	C/I	C/I	C/I			R/A	R/A	R/A	R/A	R/A					A/C				
- Create Status Report	I	I	I	I		R/A	R/A	R/A	R/A							C	I			
Control Phase Activities																				
- Perform Change Management		C	C	C		R	A	A	A	A						C	I			
Close Phase Activities																				
- Create Lessons Learned	C	C	C	C		R/A	C	C	C	C	C	C	C			C	C			
- Create Project Closure Report	I	I	I	I		R/A	I	I	I	I	I	I	I				I			

System Requirement

Project requirements are highly dependent on scope.

- Is it greenfield or redesign, variation, upgrade, etc
- Is there a contract with scope/requirement
- Can we modify the scope/requirement, and at what stage

	Requirements must be determined	Clients have produced requirements
New development	A	B
Evolution of existing system	C	D

System Requirement

Example

A system that handles university degree requirements and registrations.
Then develop a requirements statement from this.
General statement: Automate all the functions of the registrar's office.

Narrower problem statement such as the following:

'Helping university administrators manage lists of courses, degree requirements, registration and academic results. Helping students choose and register in courses in which they are interested that will lead to their degree.'

What functions to include in the system.

The functions marked with a '++' will be included, while those marked with a '−−' will be excluded.
−− Applications for admission
++ Editing and querying the list of available courses.
++ Editing and querying the requirements for obtaining a degree
++ Editing and querying the list of courses to be taught in a given semester
−− Scheduling the times that courses will be offered
−− Allocating courses or exams to rooms
++ Helping students determine which courses they could take by analyzing their degree requirements.
++ Registering students
++ Recording marks
++ Printing transcripts

System Requirement

Agile approaches to requirements

You do not develop large requirements documents. Instead, two approaches are employed: user stories and test case.

User story: is similar to a use case, but has a looser structure; it describes some proposed software feature from the perspective of how the user will use it and should be limited to about three sentences. Development proceeds by choosing a very small number of user stories to implement in the next iteration. Ideally each iteration will take only a few days to develop.

Test case: The first stage of development in many agile approaches is to first develop test cases. The series of test cases becomes the detailed specification of how a user story should be implemented.

System Requirement

Managing changing requirements

- Requirements change.
- Many reasons for change, such as, business needs, budget, technology, market, skills, Schedule, testing, scope, legal, etc.
- Requirements analysis should therefore never really stop.
- The development team should continue to interact with the stakeholders
- Incremental & iterative deployment: Agile, Lean, Scrum, etc.

System Requirement

Requirements on Requirements

S **Specific**
To-the-point, precise (iterative!)

M **Measurable**
Quantifiable and verifiable

A **Acceptable**
to the stakeholders, but achievable

R **Realistic**
Deducible to the real business drivers

T **Testable**

"The client kept changing the requirements on a daily basis, so we decided to freeze them until the next release."

Let's consider

- "All communication between client and server is secure"
- "It is easy to extend"
- "The system should respond quickly"
- "The user should not have to wait more
- "Determine solution within 0.3 sec"
- "The system should be always available"
- "The system can
- "The system can handle 100 concurrent users"
- "The system should be state-of-the-art …"

not measurable

not precise

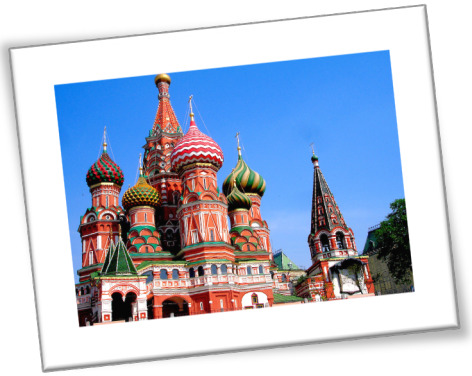
vague: to what?

attainable

doing what?

time-dependent;
means something else tomorrow

subjective



Prioritizing Requirements

MoSCoW Method:

M - MUST: be satisfied in the final solution.

S - SHOULD: high-priority item that should be included if it is possible.

C - COULD: is considered desirable but not necessary.

W - WON'T: stakeholders have agreed will not be implemented in this release.

Learning Focus

- Ability to describe requirement statement, and Scope of Work SOW
- Define types of requirement, and their classifications
- Realize the challenge of identifying functional vs. non-functional requirements
- Realise types of constrains and classify them based on their system impact
- Understand the reason behind most projects failure.
- Understand how RACI tool can help mitigating project risks at an early stage.
- Ability to define type of project based on greenfield, vs. upgrade or improvement
- Understand agile approach and user/test cases definitions.
- Understand requirement priorities and requirement on requirement

Outline


Concept alignment


Software Architecture Contexts


System Requirements


System Components & Layers


Use Cases Brief


Quality Attributes


QA Scenarios


Achieving QA using Tactics

System Components & Layers

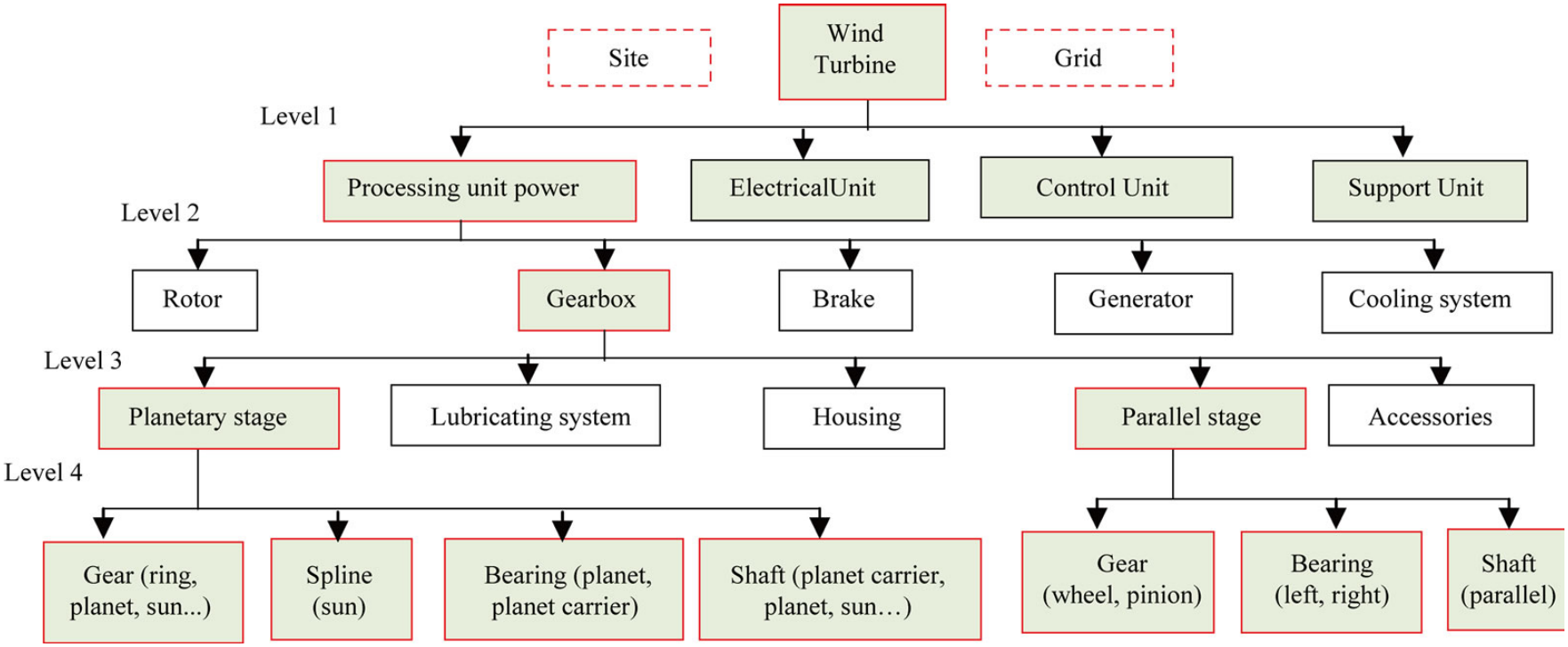
What is a subsystem/component?

Sub-system models is a logical grouping of functionality into a coherent subsystems:

- Operate on the same data
- Perform functions that belong to the same system
- Viewed at the same abstraction and details layer

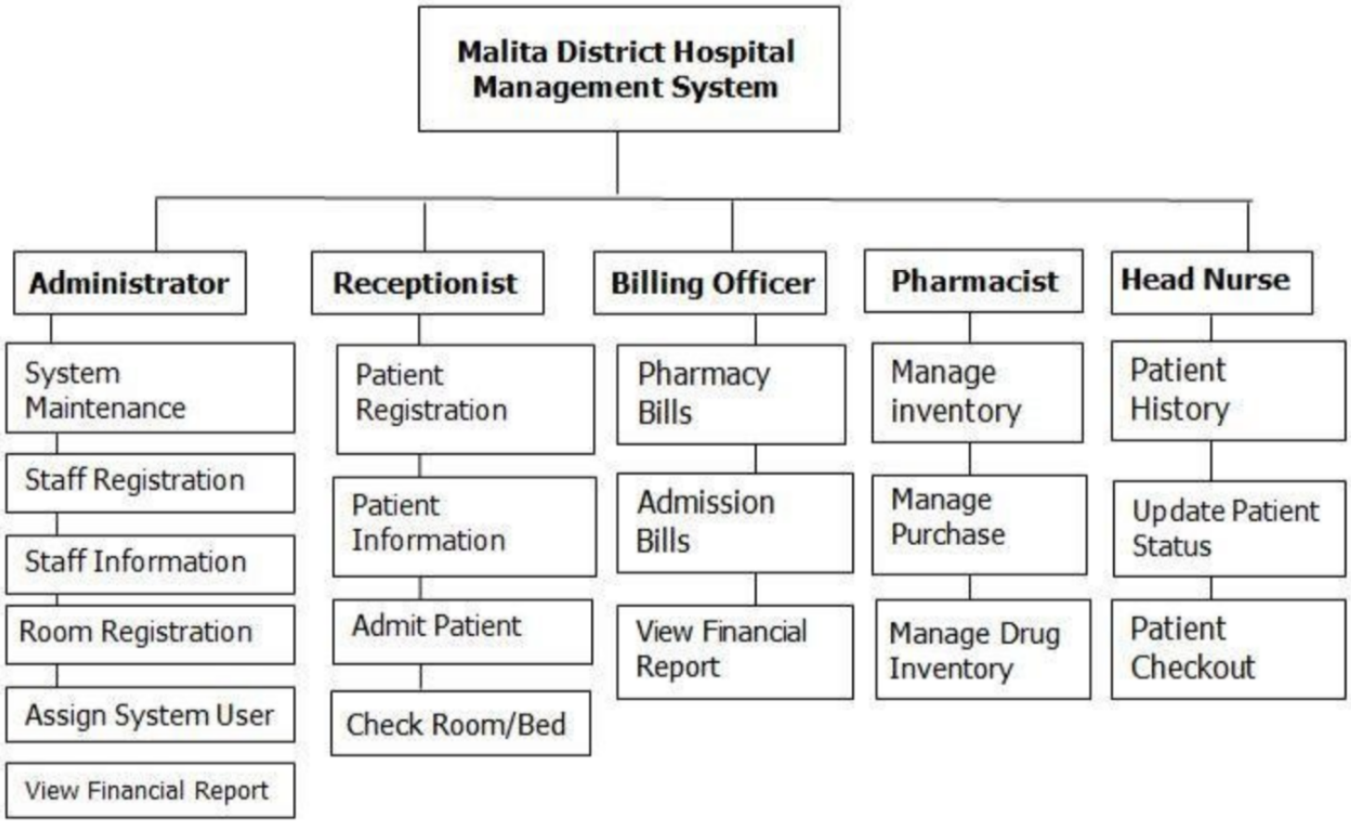
System Components & Layers

Subsystem Functions

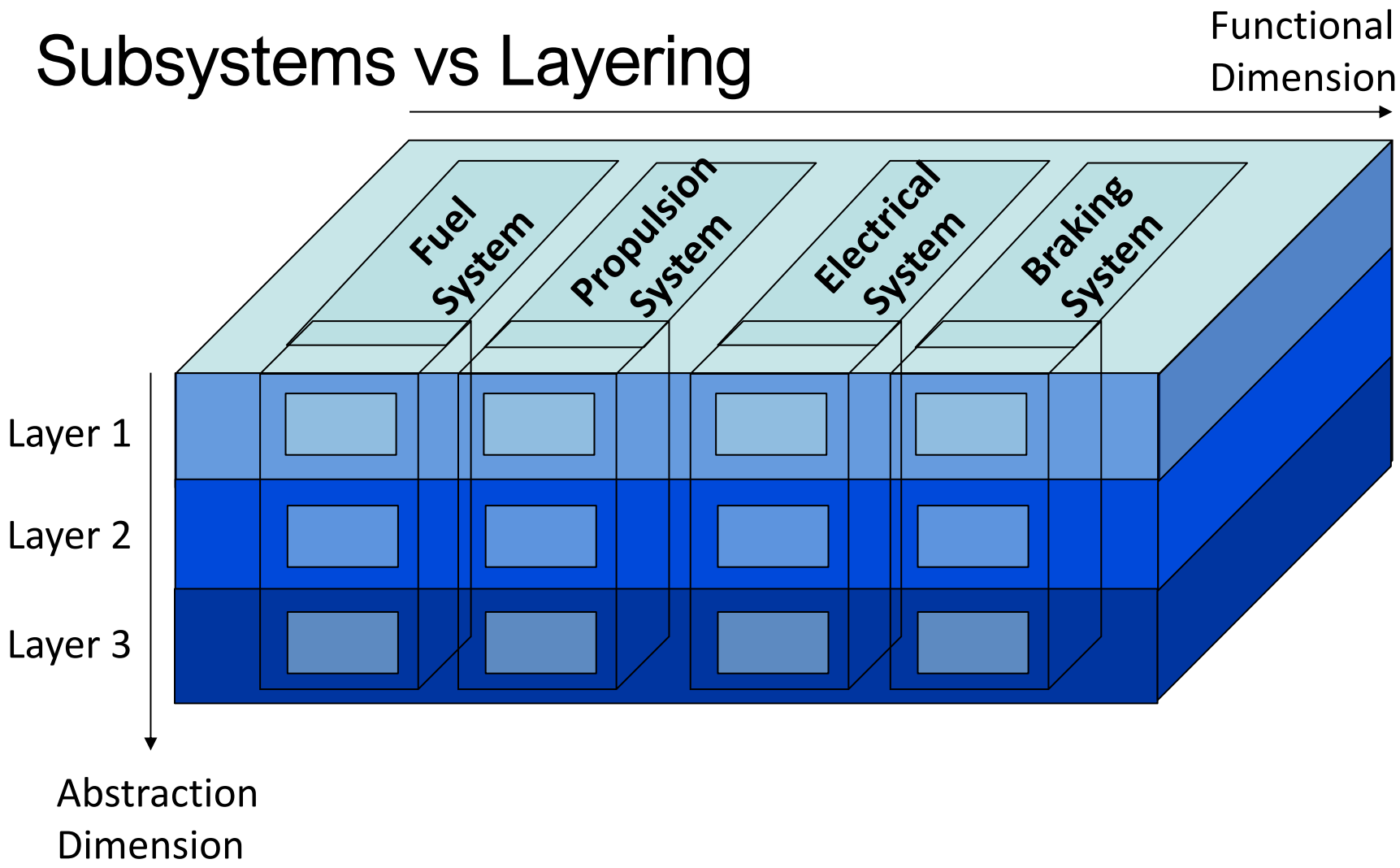


System Components & Layers

Functional Decomposition



System Components & Layers

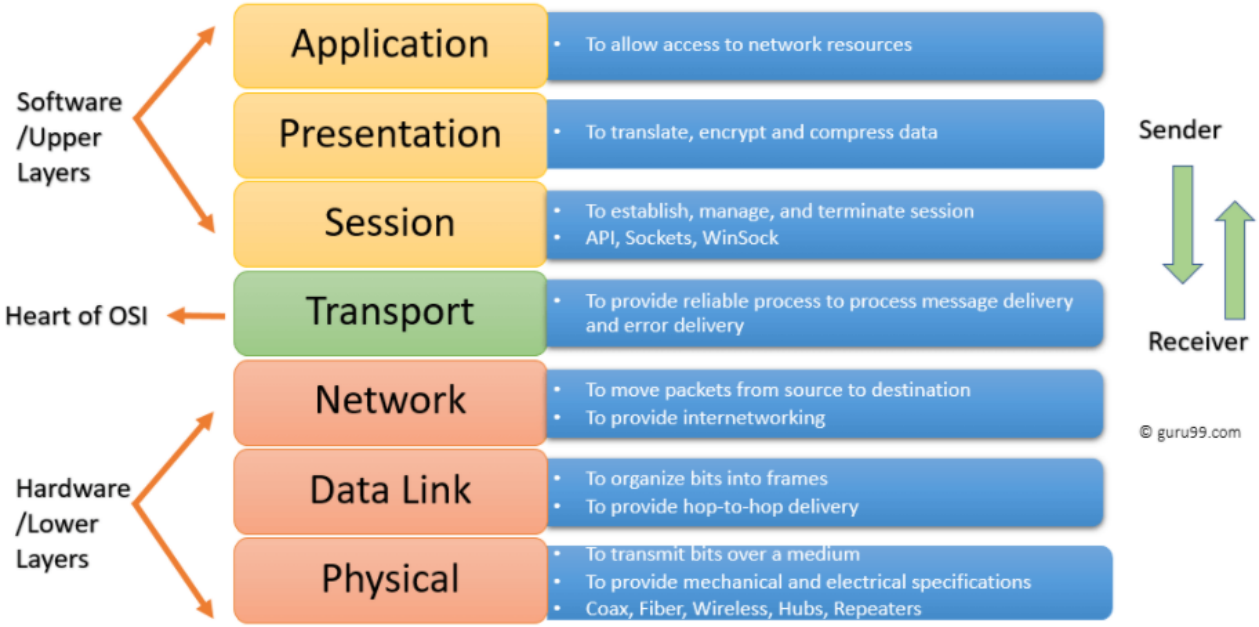


System Components & Layers

Subsystems vs Layering

7 Layers of the OSI Model

Open Systems Interconnection model (OSI model) is a layered server architecture system
Each layer is defined according to a specific function to perform.
All these seven layers work collaboratively to transmit the data from one layer to another.



Network Layers Diagram

OSI goal is to enable the interoperability of diverse communication systems with standard communication protocols. The model partitions a communication system into abstraction layers

Learning Focus

- Ability to define subsystem models and site examples
- Identify functional decomposition and system layered abstraction
- Understand the purpose of collaborating layers in OSI model

Outline


Concept alignment


Software Architecture Contexts


System Requirements


System Components & Layers


Use Cases Brief


Quality Attributes


QA Scenarios

Use Cases Brief

Use Case (how actors will use the system)

- Determine the types of users or systems that will use the system.
- It is a typical sequence of actions that an actor performs in order to complete a given task.
- An actor is a role that a user or some other system plays when interacting with system.
- Most of the actors will be users; a given user may be considered as several different actors
- A use case should include only actions in which the actor interacts with the system.

Use Cases Brief

How to describe a single use case

1. **Name**. Give a short, descriptive name to the use case.
2. **Actors**. List the actor or actors who can perform this use case.
3. **Goals**. Explain what the actor or actors are trying to achieve.
4. **Preconditions**. Describe the state of the system before the use case occurs.
5. **Summary**. Summarize what occurs as the actor or actors perform the use case.
6. **Related use cases** . List use cases that may be generalizations, specializations, extensions or inclusions of this one.
7. **Steps** . Describe each step of the use case using a two-column format.
8. **Postconditions** . What state is the system in following the completion of the use case.

Example (Use Case)

Briefly describe a use case for leaving a particular automated car park (parking lot). Use case: Exit car park, paying cash. Actors: Car drivers, Goals: To leave the parking lot after having paid the amount due.

Preconditions: The driver must have entered the car park with his or her car, and must have picked up a ticket upon entry

Summary: When a driver wishes to exit the car park, he or she must bring his or her car to the exit barrier and interact with a machine to pay the amount due.

Related use case: Exit car park by paying using a debit card

Steps:

Actor actions	*System responses*
1. Drive to exit barrier, triggering a sensor.	2a. Detect presence of a car.
	2b. Prompt driver to insert his or her card.
3. Insert ticket.	4. Display amount due.
5. Insert money into slot.	6a. Return any change owing.
	6b Prompt driver to take the change (if any).
	6c. Raise barrier.
7. Drive through barrier, triggering a sensor.	8. Lower barrier.

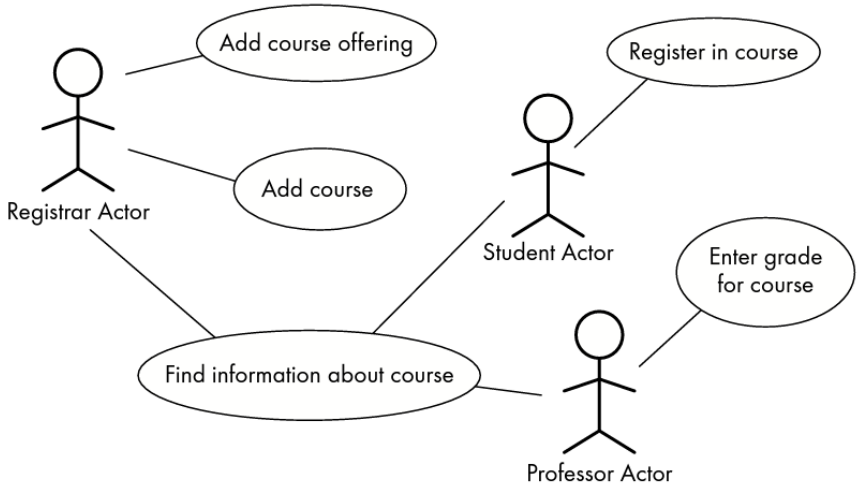
Use Cases Brief

Use case diagrams

Use case diagrams are UML's notation for showing the relationships among a set of use cases and actors.

They help a software engineer to convey a high-level picture of the functionality of a system.

There are two main symbols in use case diagrams: an actor is shown as a stick person and a use case is shown as an ellipse. Lines indicate which actors perform which use cases.



A simple use case diagram showing three actors and five use cases

Use Cases Brief

The use case modeler can use extensions , generalizations or inclusions to represent different types of relationships among use cases.

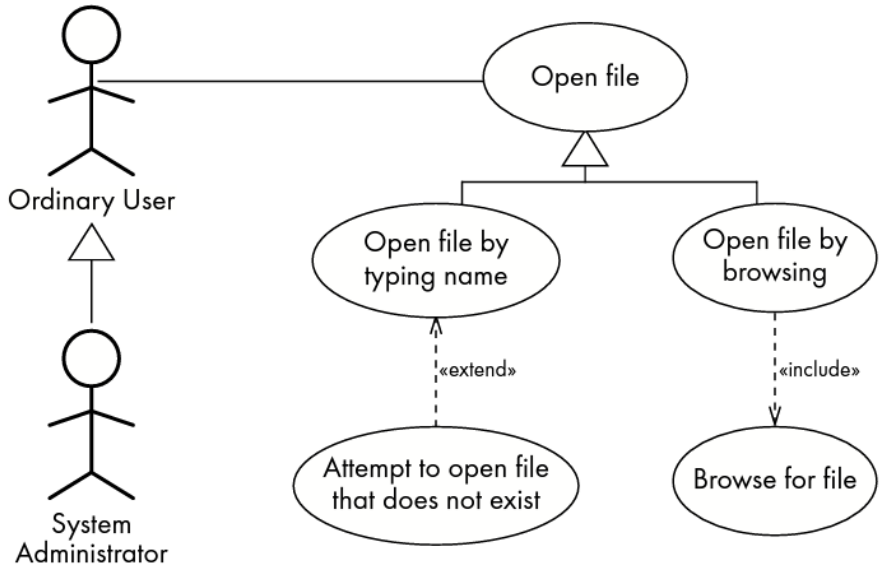
Extensions are used to make optional interactions or handle exceptional cases.

Generalizations use triangle symbol: several similar use cases can be shown along with a common generalized use case. Same like parent and child.

Inclusions allow you to express a part of a use case so that you can capture commonality between several different use cases.

Use Cases Brief

The open triangle points to a generalization. The «extend» and «include» stereotypes show the other relationships between use cases. Note that actors can also be arranged in a generalization hierarchy.



Extension, generalization and inclusion in a use case diagram

Use Cases Brief

The benefits of use cases for software development

Use cases:

- Can help to define the scope of the system
- The number of use cases is a good indicator of a project's size.
- Develop and validate the requirements
- Helps stakeholders to understand requirements
- Can be used to structure user manuals.

Use Cases Brief

Scenarios

A scenario is an instance of a use case

It can help to clarify the associated use case.

It is also often simply a use case instance .

Example: Describe a concrete scenario corresponding to the 'Exit car park, paying cash' use case from Example 4.11.

Steps:

Actor actions	*System responses*
Drives to the exit barrier.	Detects the presence of a car.
	Displays: 'Please insert your ticket'.
Inserts ticket.	Displays: 'Amount due $2.50'.
Inserts $1 into the slot.	Displays: 'Amount due $1.50'.
Inserts $1 into the slot.	Displays: 'Amount due $0.50'.
Inserts $1 into the slot.	Returns $0.50.
	Displays: 'Please take your $0. 50 change'.
	Raises barrier.
Drives through barrier, triggering sensor.	Lowers barrier.

Learning Focus

- Understand use cases and how they help in system requirement development
- Ability to show a use case for a given system functionality
- Know how to use generalize, include and extend use cases in use case diagram
- Understand use scenario and how to instantiate it from a use case

Outline


Concept alignment


Software Architecture Contexts


System Requirements


System Components & Layers


Use Cases & Scenarios


Quality Attributes


QA Scenarios

Quality Attributes

Systems are frequently redesigned not because they are functionally deficient.

Stakeholders decide value and priorities of functions and attribute.

It is the mapping of a system's functionality onto software architecture that determines the architecture's quality attributes.

A quality requirement is a specification of the acceptable quality attribute.

A quality attribute is a measurable or testable property of a system.

Quality attributes should be communicated based on KPIs that are agreeable across all stakeholders.

Quality Attributes

Managing Software Quality

Main issues:

- Quality cannot be added as an afterthought (really!)
- In order to control, we must measure and use defined (KPIs)
- Product quality vs process quality (Response vs. Efficiency)
- Stakeholders are the juries when it comes to testing quality
- There is always iteration of quality and cyclic development process
- This explains why we have release and versioning management
- Attributes qualification should be defined (technically and legally)

Quality Attributes

Attribute	Description
Performance	How fast does it respond or execute?
Availability	Is it available when and where I need to use it?
Safety	How well does it protect against damage?
Usability	How easy it is for people to learn and use?
Interoperability	How easily does it interconnect with other systems?
Integrity	Does it protect against unauthorized access and data loss?
Installability	How easy is it to correctly install the product?
Robustness	How well does it respond to unexpected operating conditions?
Reliability	How long does it run before experiencing a failure?
Recoverability	How quickly can the user recover from a failure?

Attributes may imply different perspective based on system context

Quality Attributes

Attribute	Description
Efficiency	How well does it utilize processor capacity, disk space, memory, bandwidth, and other resources?
Flexibility	How easily is it to modify functionality / UI?
Maintainability	How easy is it to correct defects or make changes?
Portability	How easily can it be made to work on other platforms?
Reusability	How easily can we use components in other systems?
Scalability	How easily can I add more users, servers, or other extensions?
Supportability	How easy will it be support after installation?
Testability	Can I verify that it is implemented correctly?

Quality Attributes

Quality attributes [edit]

Notable quality attributes include:

- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- agility (see Common subsets below)
- auditability
- autonomy
- availability
- compatibility
- composability
- configurability
- correctness
- credibility
- customizability
- debuggability
- degradability
- determinability
- demonstrability
- dependability (see Common subsets below)
- deployability
- discoverability
- distributability
- durability
- effectiveness
- efficiency
- evolvability
- extensibility
- failure transparency
- fault-tolerance
- fidelity
- flexibility
- inspectability
- installability
- integrity
- interchangeability
- interoperability
- learnability
- localizability
- maintainability
- manageability
- mobility
- modifiability
- modularity
- observability
- operability
- orthogonality
- portability
- precision
- predictability
- process capabilities
- producibility
- provability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- resilience
- responsiveness
- reusability
- robustness
- safety
- scalability
- seamlessness
- self-sustainability
- serviceability
- securability
- simplicity
- stability
- standards compliance
- survivability
- sustainability
- tailorability
- testability
- timeliness
- traceability
- transparency
- ubiquity
- understandability
- upgradability
- usability
- vulnerability

Many of these quality attributes can also be applied to data quality.

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

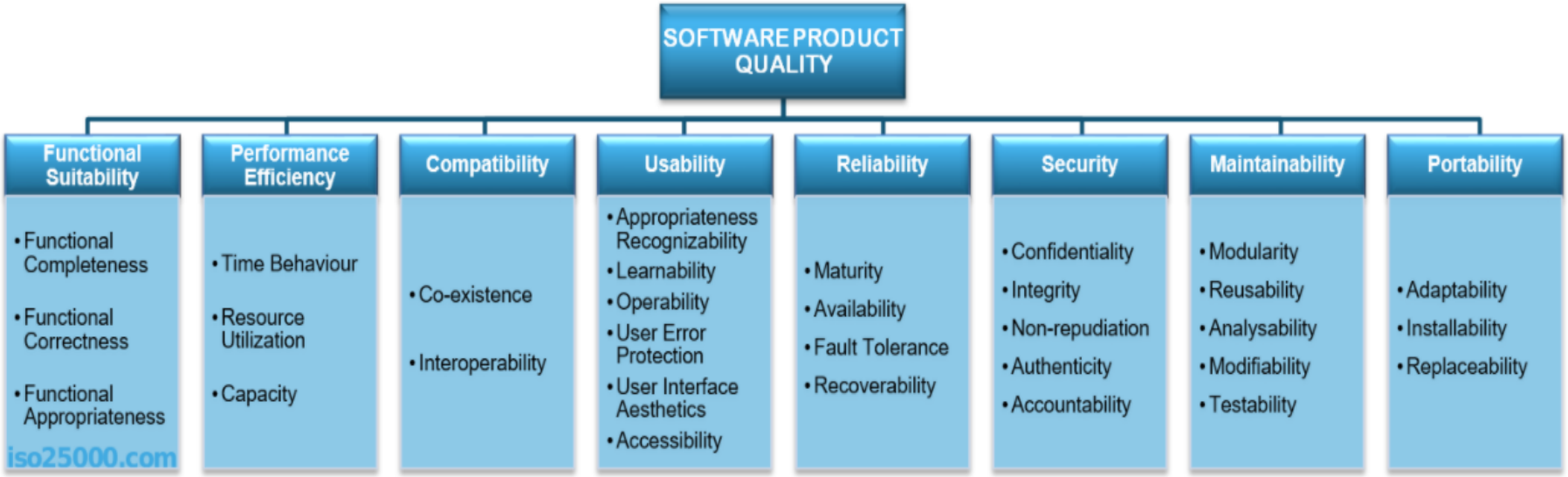
and more.....

Quality Attributes

ISO/IEC 25010 Quality Model*

The quality of a system is the degree to which the system satisfies the stated and implied needs of its various stakeholders.

The product quality model defined in ISO/IEC 25010 comprises the eight quality characteristics shown in the following figure:



*https://iso25000.com/index.php/en/iso-25000-standards/iso-25010#:~:text=ISO%2FIEC%2025010&text=The%20quality%20model%20determines%20which,stakeholders%2C%20and%20thus%20provides%20value.

Quality Attributes

Availability

Availability is the percentage of time when system it is operational.

$$A = \frac{MTBF}{MTBF + MTTR}$$

Mean Time Between Failures (MTBF)

Number of hours that pass before a component fails

E.g. 2 failures per million hours: MTBF = 10^6 / 2 = 0,5 * 10^6 hr

Availability	Downtime
90% (1-nine)	36.5 days/year
99% (2-nines)	3.65 days/year
99.9% (3-nines)	8.76 hours/year
99.99% (4-nines)	52 minutes/year
99.999% (5-nines)	5 minutes/year
99.9999% (6-nines)	31 seconds/year !

Calculate **MTTR** by dividing the total **time** spent on unplanned maintenance by the number of **times** an asset has failed

Quality Attributes

Performance

Performance is composed of the following sub-attributes:

Time behavior - system responses and processing times

- Throughput – number of bytes handled per second
- Response time/Turn-around time/ signal Latency & jitter

Resource utilization - Degree to which the amounts and types of resources used

Capacity - relate to stress condition perfomance

Quality Attributes

Security

Security can be characterized as a system providing

1. **Nonrepudiation** is the property that assures transaction authenticity
2. **Confidentiality** data or services are protected from unauthorized access.
3. **Integrity** is the property that data or services are being delivered as intended.
4. **Availability** is the property that the system will be available for legitimate use.
5. **Auditable** system tracks activities within is sufficient to trace and audit
6. **Compliance & Privacy**: Laws compliance: General Data Protection Regulation (**GDPR**)

Quality Attributes

Compatibility

Degree to which a product, system or component can exchange information with other products, systems or components.

It has the following sub-characteristics:

Co-existence - Degree to which a product can perform its required functions efficiently while sharing a common environment and resources.

Interoperability - Degree to which two or more systems, products or components can exchange information and use the information that has been exchanged

The trade-off decision

You can choose 2
but all 3 is expensive

Real life we trade-off

Quality Attributes

Evaluating Quality Attributes

Quality attributes can be evaluated through:

- **Scenario-based evaluation**: eg. *scenarios* for assessing maintainability
- **Simulation**: a part of the architecture is implemented and executed in the actual system context.
- **Test Environment**: Controlled nonproduction testing with similar environment
- **Mathematical modeling**: checking for *potential deadlocks, and performance.*.
- **Prototype or emulated concept design:** Build a concept system, **MVP** (suitable for large systems)

Quality Attributes

Evaluating Quality Attributes

Overlapping concerns

Performance: due to DDS attack or poor design

Security: due to poor layering or internal compromise

Untestable Concern

The quality attribute should be tested in all the circumstances. (stress condition)

Gathering Quality Attribute Information

Quality requirements and design constraints are enabled by two main techniques:

- Quality Attribute Scenario (QAS) and
- Quality Attribute Workshop (QAW).

Learning Focus

- Understand QAs relationship to system functionality, and how they qualify system functional requirement.
- How to manage QAs, identify, measure and test them in the right context
- Understand the product quality model defined in ISO/IEC 25010 with the eight quality characteristics
- Understand Qas for Security, Availability, and Performance in more details.
- Understand the techniques used for evaluating system QAs

Outline


Concept alignment


Software Architecture Contexts


System Requirements


System Components & Layers


Use Cases & Scenarios


Quality Attributes


QA Scenarios

QA Scenarios

Quality Attribute Scenario (QAS)*

QAS appears to solve the untestable and overlapping concerns.

The aim of a QAS is to capture the explicit and testable quality requirements

It does it in the same way the **use case scenarios** do for functional requirements by initiating a use case instant.

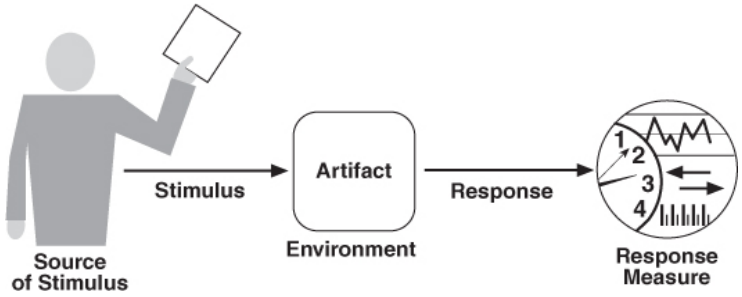
QAS consists of six parts.

* "Software Architecture for Business", by Lina Khalid, ISBN 978-3-030-13631-4 © Springer Nature Switzerland AG 2020

QA Scenarios

We specify quality attribute requirements, we capture them formally as six parts of QAS:

1. *Source of stimulus*. (a human, or any other actuator) that generated the stimulus.
2. *Stimulus.* A condition that requires a response. For different quality it means something specific.
3. *Environment*. The system may be in an overload condition, test, or in normal operation.
4. *Artifact.* Some artifact is stimulated. This may be a collection or whole system, or pieces of it.
5. *Response.* The response is the activity undertaken as the result of the arrival of the stimulus.
6. *Response measure.* A response should be measurable so that the requirement can be tested.



Parts of a quality attribute scenario (ex. web portal responsiveness).

QA Scenarios

There are two types of QAS: general and concrete:
- A General scenario do not belong to any system.
- A Concrete scenario belongs to a particular system under specific conditions.

Table 3.1 General QAS for availability quality

Source	The source can be internal or external, for example, people, hardware, software, physical infrastructure, etc.
Stimulus	Fault
Artifacts	Processors, communication channels, persistent storage, process
Environment	Normal operation, shutdown, repair mode, overloaded operation
Response	Prevent fault from becoming failure Detect the fault Recovery from the fault
Response measure	Availability percentage, for example (99.999%), time to detect fault, time to repair fault, time or time interval in which a system can be in a degraded mode, etc.

Table 3.3 Concrete table

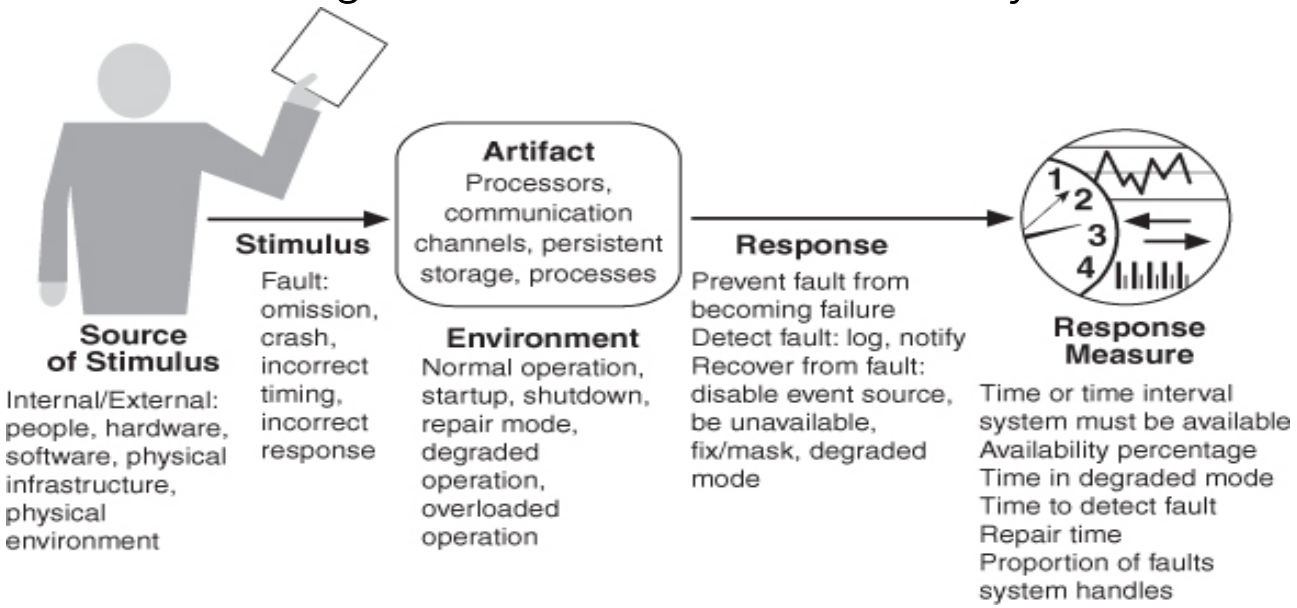
Source	Internal hardware
Stimulus	Crash
Artifacts	Processors
Environment	Normal operation
Response	Detect the fault Recovery from the fault
Response measure	System can be in a degraded mode no more than 15 minutes

QA Scenarios

We develop first the **general quality attribute scenarios**, for a specific attribute such as availability.

Then we translate them to the specific requirement of the system under development to get concrete scenarios, by specifying the source and the stimulus.

A general scenario for availability



QA Scenarios

Quality Attribute Workshop (QAW)

Quality Attribute Workshop is a facilitated method for a few-days workshop.
It connects stakeholders in the early part of the life cycle in order to find quality attributes for the existing system.

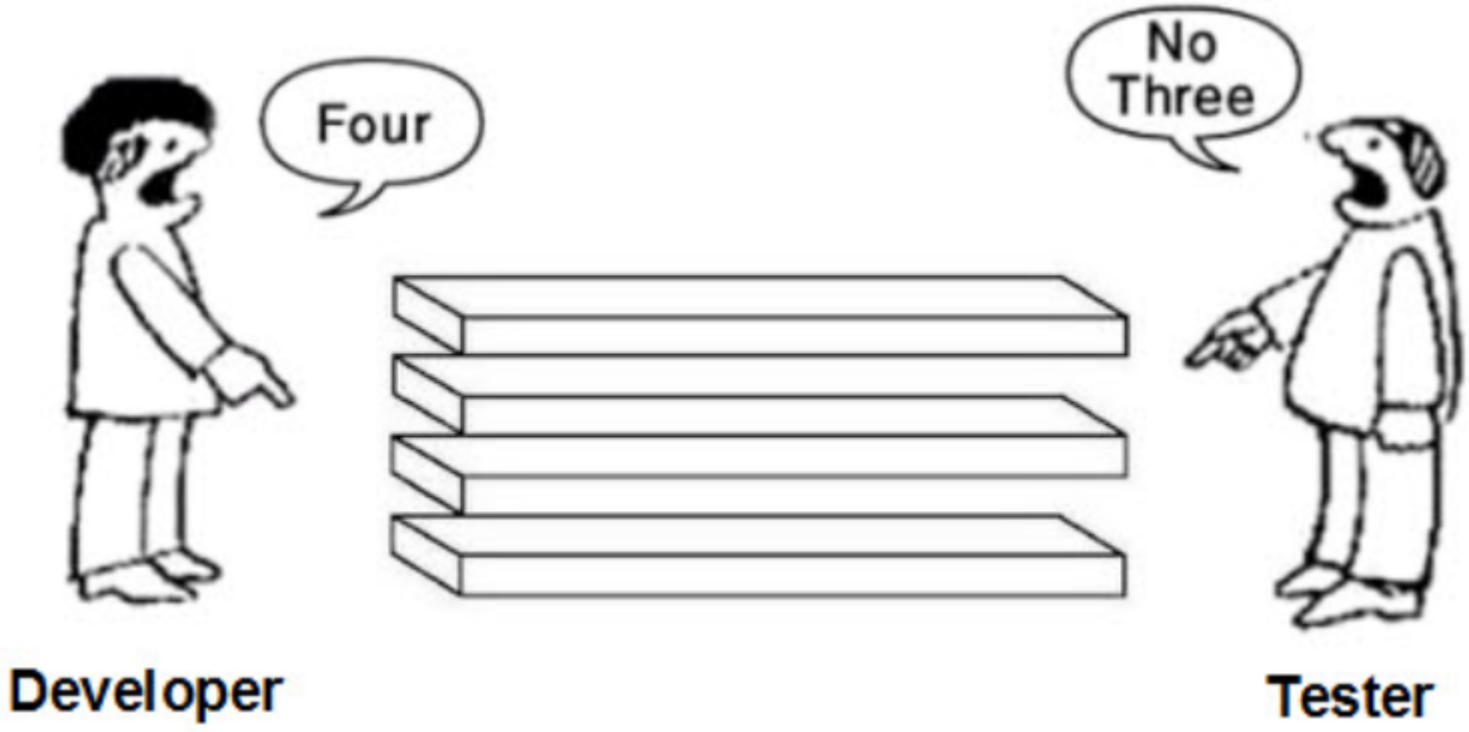
The important thing to know about QAW is that:

- It is focused on the stakeholders.
- It is scenario based.
- It is used before the software architecture begins.
- It is focused on the system level concerns and on the role of software in the system.

Learning Focus

- Understand purpose of QAS and how to use them
- How to capture the 6 parts general and concrete scenarios
- Understand the purpose and activities of the QAW

QAs context



See you at Architecture Styles III
& Blockchains Architecture