EN
English

University of Oviedo

Universidad de Oviedo

School of Computer Science

Quality attributes and design concepts

School of Computer Science

Course 2018/2019

SOFTWARE ARCHITECTURE

Jose E. Labra Gayo

Contents

Types of requirements

Requirements can be categorized as:

- **Functional requirements**: state what the system must do, how it must behave or react to run-time stimuli.
- **Quality attribute requirements**. Annotate (qualify) functional requirements
  - Examples: Availability, modifiability, usability, security,...
  - Also called: non-functional requirements
- **Constraints**. A design decision that has already been made

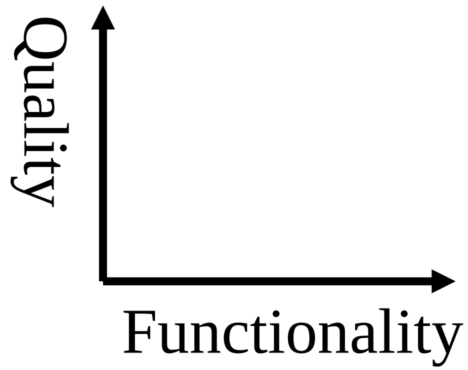
Functional requirements

Functionality = ability of the system to do the work for which it was intended

Functionality has a strange relationship to architecture:

- It does not determine architecture;
  - If functionality were the only requirement, the system could exist as a single monolithic module with no internal structure at all

Functionality and quality attributes are orthogonal



Quality attributes (QA)

Quality attributes annotate (qualify) the functionality

- If a functional requirement is "when the user presses the green buttom an options dialog appears"
  - A performance QA could describe how quickly
  - An availability QA could describe this option could fail, or how often it will be repaired
  - A usability QA could describe how easy is to learn this function

Quality attributes influence the architecture

What is Quality?

Degree to which a system satisfies the stated and implied needs of its stakeholders, providing value

- Degree (not Boolean)
- Quality = Fitness for purpose (*stakeholders* needs)
- Conform to requirements (stated and implied)
- Providing value

Several definitions of quality

`https://en.wikipedia.org/wiki/Software_quality`

Quality attributes and trade-offs

QAs are all good

...but value depends on the project & stakeholder

- *"Best quality"...for what?, for whom?*

QAs are not independent

- Some qualities can conflict
- What matters most?
  - Example: A very secure system can be less usable

There is always a price

- What is your budget?

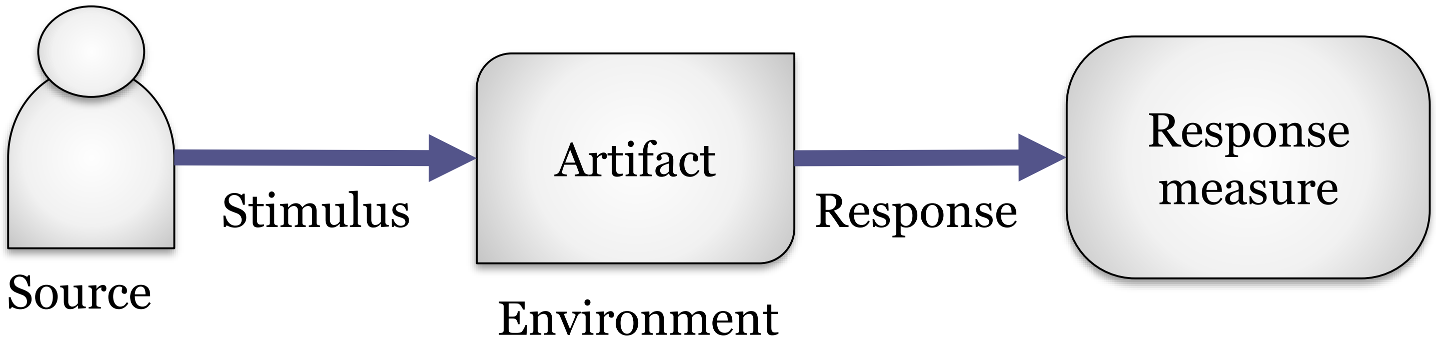
There is no single *perfect* system or architecture!

Specifying quality attributes

- 2 considerations:
  - QAs by themselves are not enough
    - They are non-operational
      - Example: It is meaningless to say that a system shall be modifiable or maintainable
  - The vocabulary describing QAs varies widely
    - It is necessary to describe each attribute separately
- QA scenarios: A common form to specify QA requirements

Quality attribute scenario

- Describe a stimulus of the system and a measurable response to the stimulus
  - Stimulus = event triggered by a person or system
  - The stimulus generates a response
- Must be testable
  - Response must be externally visible and measurable



Components of QA scenario

Source: Person or system that initiates the stimulus

Stimulus: Events that requires the system to respond

Artifact: Part of the system or the whole system

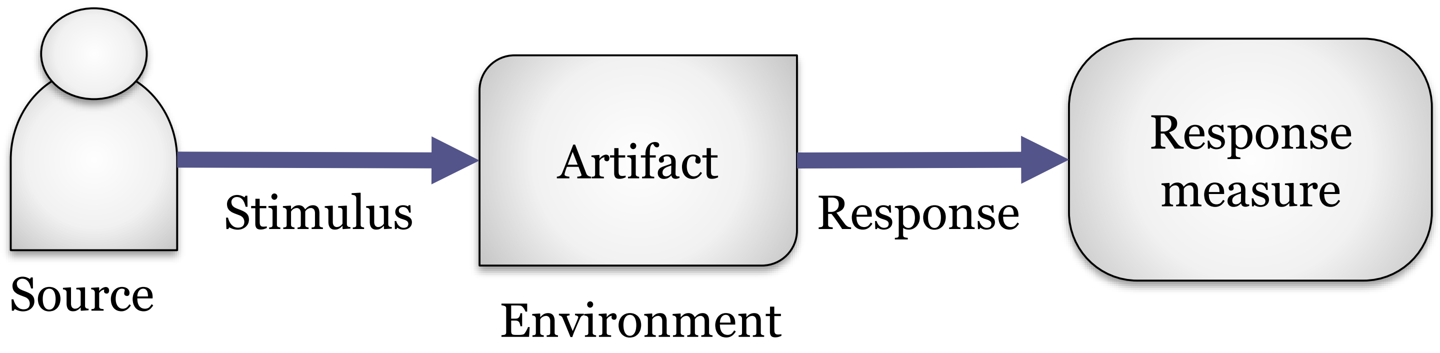
Response: Externally visible action

Response measure: Success criteria for the scenario

- Should be specific and measurable

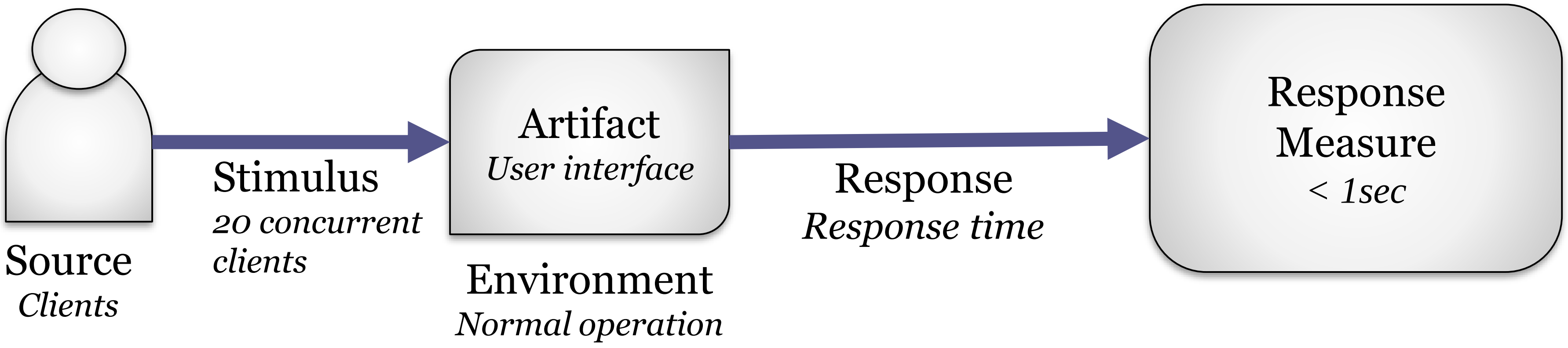
Environment: Operational circumstances

- Should be always defined (even if it is "normal")



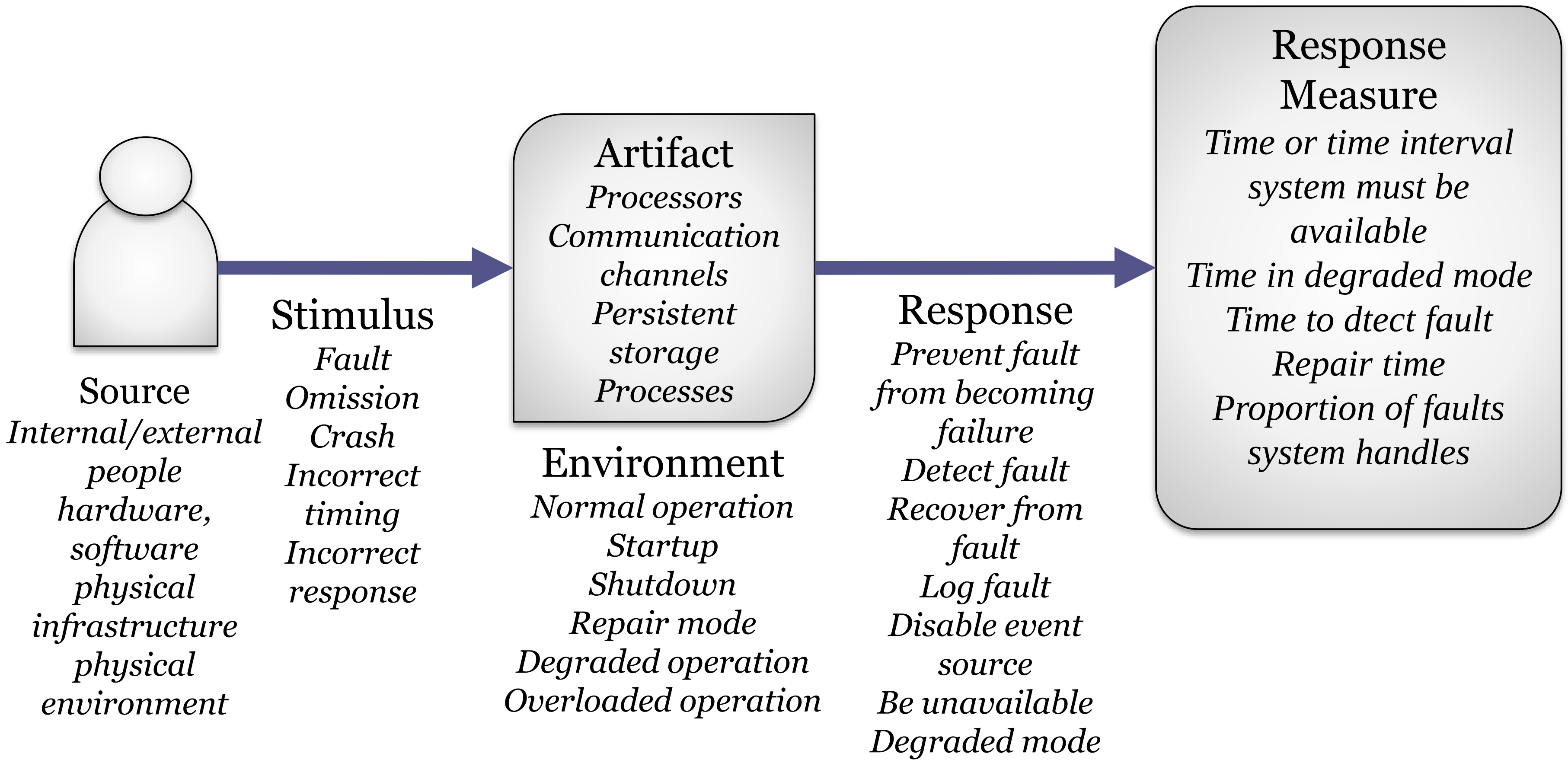
QA scenario, example 1

***Performance**: If there are 20 concurrent clients, the response time should be less than 1 sec. under normal operation*



Source of stimulus	Stimulus	Artifact	Environment	Response	Response measure
Clients	20 concurrent clients	User interface	Normal operation	Response time	<1sec
. . .	. . .	. . .	. . .	. . .	. . .

QA Scenario structure for availability



Source
Internal/external people
hardware,
software
physical infrastructure
physical environment

Stimulus
Fault
Omission
Crash
Incorrect timing
Incorrect response

Artifact
Processors
Communication channels
Persistent storage
Processes

Environment
Normal operation
Startup
Shutdown
Repair mode
Degraded operation
Overloaded operation

Response
Prevent fault from becoming failure
Detect fault
Recover from fault
Log fault
Disable event source
Be unavailable
Degraded mode

Response Measure
Time or time interval system must be available
Time in degraded mode
Time to dtect fault
Repair time
Proportion of faults system handles

Types of QA scenarios

- Usage scenarios
  - The system is used (any use case or system function is executed)
    - Describe how the system behaves in these cases
      - Runtime, performance, memory consumption, throughput,...
- Change (or modification) scenarios
  - Any component within the system, its environment or its operational infrastructure changes
- Failure scenarios:
  - Some part of the system, infrastructure or neighbours fail

Prioritizing QA scenarios

- Scenarios should be prioritized
  - 2 values (Low/Medium/High)
    - How important it is for success (ranked by customer)
    - How difficult it is to achieve (ranked by architect)

Ref	Quality Attribute	Scenario	Priority
1	Availability	When the database does not respond, the system should log the fault and respond with stale data during 3 seconds	High, High
2	Availability	A user searches for elements of type X and receives a list of Xs 99% of the time on average over the course of the year	High, Medium
3	Scalability	New servers can be added during the planned maintainance window (less than 7 hours)	Low, Low
4	Performance	A user sees search results within 5 seconds when the system is at an average load of 2 searches per second	High, High
5	Reliability	Updates to external elements of type X should be reflected on the application within 24 hours of the change	Low, Medium
	. . .	. . .	. . .

Identifying quality attributes

- Finding QAs
  - Most of the time, QAs are not explicit
    - They're only verbally said alongside func. requirements
    - Usually implicit or said without much thought
  - Software architect must do educated guesses
- Quality Attribute Workshops
  - Meetings where stakeholders specify QAs
- Formal checklists
  - ISO25010
  - Wikipedia:
    - https://en.wikipedia.org/wiki/List_of_system_quality_attributes

Typical quality attributes

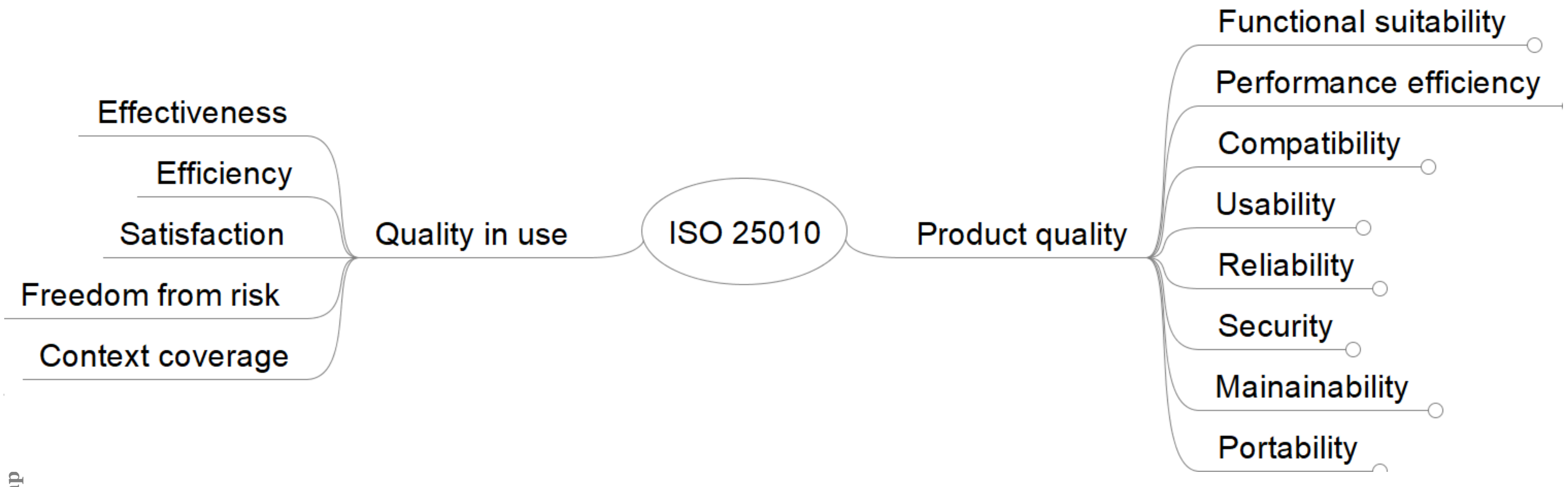
Availability
Modifiability
Performance
Security
Testability
Maintainability
Usability
Scalability
Interoperability
Portability
Changeability
. . .

ISO-25010 Software Quality Model

Systems and software Quality Requirements and Evaluation (SQuaRE)

2 parts:

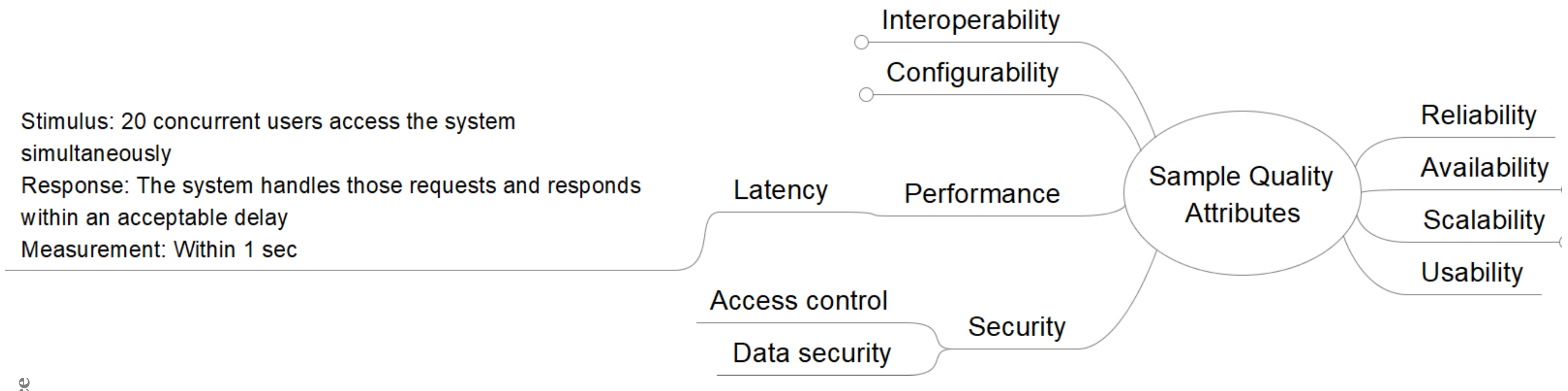
- Quality in-use
- Product quality



https://arquisoft.github.io/Iso25010QualityAttributes.html

Quality Attribute tree

Mindmap representations can be useful to visualize QA scenarios

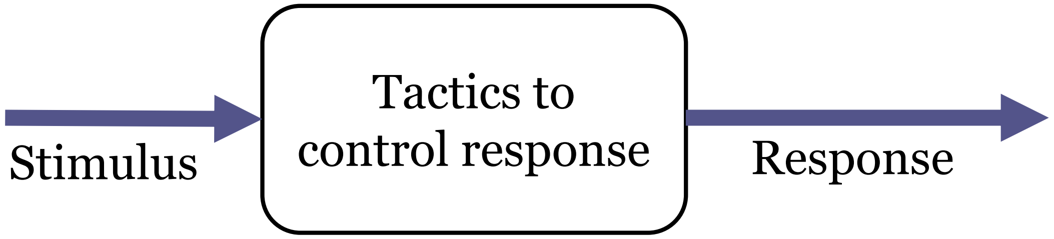

Interoperability
Configurability
Stimulus: 20 concurrent users access the system simultaneously
Response: The system handles those requests and responds within an acceptable delay
Measurement: Within 1 sec
Latency
Performance
Sample Quality Attributes
Reliability
Availability
Scalability
Usability
Access control
Data security
Security

Achieving software architecture

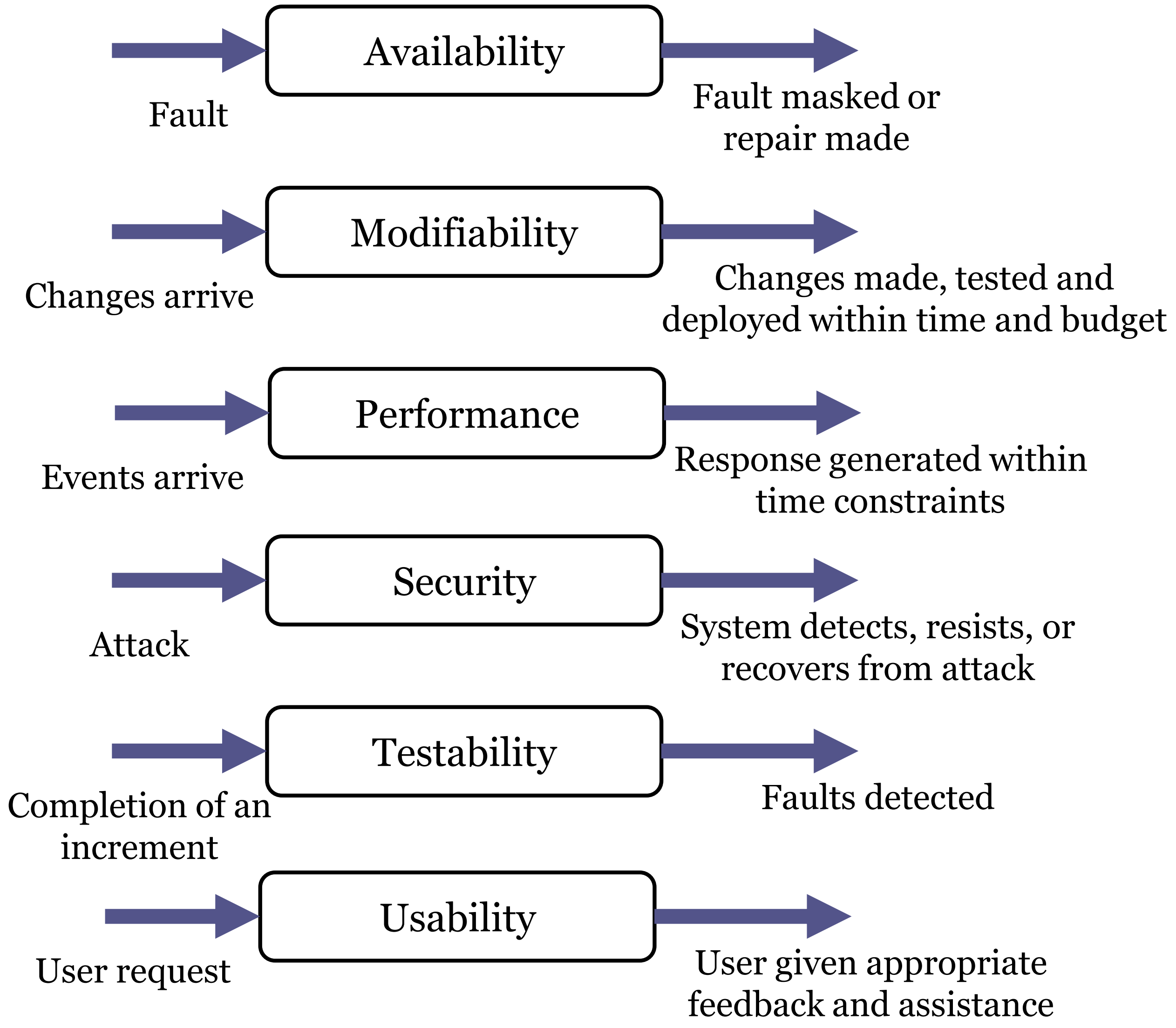
Tactics, styles, patterns, reference architectures

Tactics

- Design techniques to achieve a response to some quality attributes
- Tactics focus on a single quality attribute response
  - They may compromise other quality attributes
- Tactics are intended to control responses to stimuli

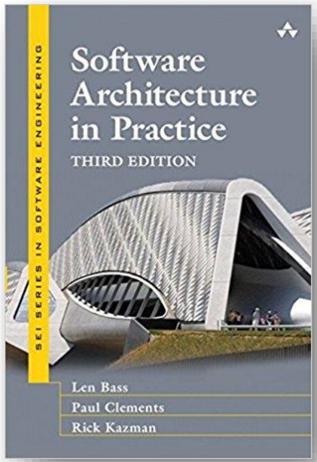


Tactics depend on QA


Fault
Availability
Fault masked or repair made
Changes arrive
Modifiability
Changes made, tested and deployed within time and budget
Events arrive
Performance
Response generated within time constraints
Attack
Security
System detects, resists, or recovers from attack
Completion of an increment
Testability
Faults detected
User request
Usability
User given appropriate feedback and assistance

Where can we find tactics?

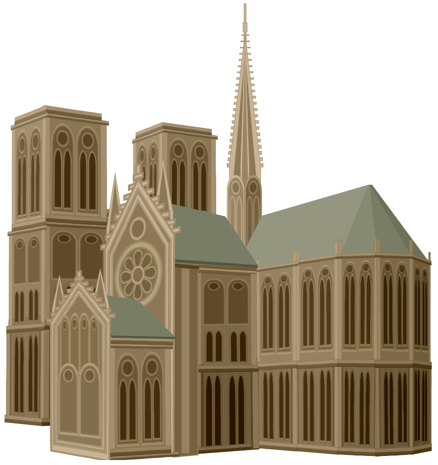
- Architect's own experience
- Documented experience from community
  - Books, conferences, blogs,...
- Tactics evolve with time and trends
  - Book "Software architecture in practice" contains a list of tactics for some quality attributes


Software Architecture in Practice
THIRD EDITION
SEI SERIES IN SOFTWARE ENGINEERING
Len Bass
Paul Clements
Rick Kazman

http://www.ece.ubc.ca/~matei/EECE417/BASS/ch05lev1sec1.html

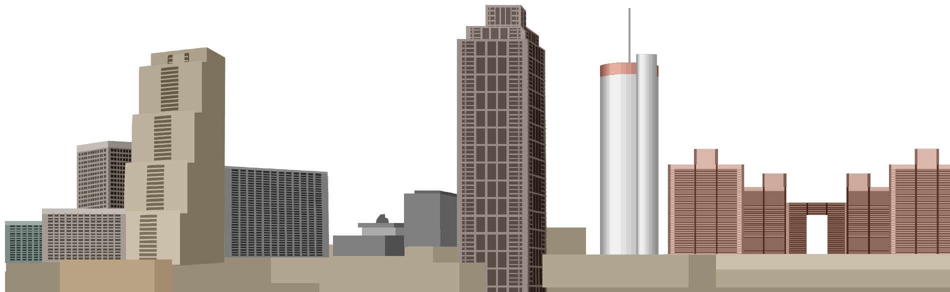
Architectural styles

- Define the general shape of a system
  - They contain:
    - Elements: Components that carry out functionality
    - Constraints: define how to integrate elements
    - List of attributes:
      - Advantages/disadvantages of a style



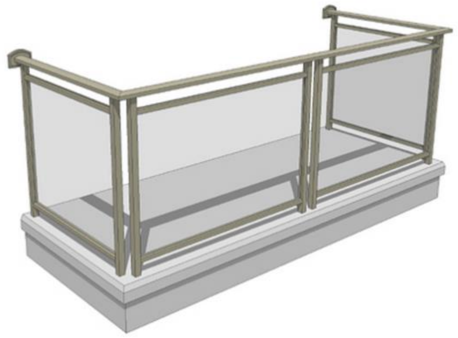
Are there pure styles?

- Pure styles = idealization
- In practice, pure styles rarely appear
- Usually, systems deviate from pure styles...
- ...or combine several architectural styles
- It is important to understand pure styles in order to:
  - Understand pros and cons of a style
  - Assess the consequences of a deviation from the style



Architectural pattern

- Reusable and general solution to some recurring problem that appears in a given context
  - Important parameter: **problem**
- 3 types:
  - Structural: Build time
    - Example: Layers
  - Runtime (behaviour)
    - Example: Pipes & filters
  - Deployment
    - Example: Load-balanced cluster



Pattern vs style

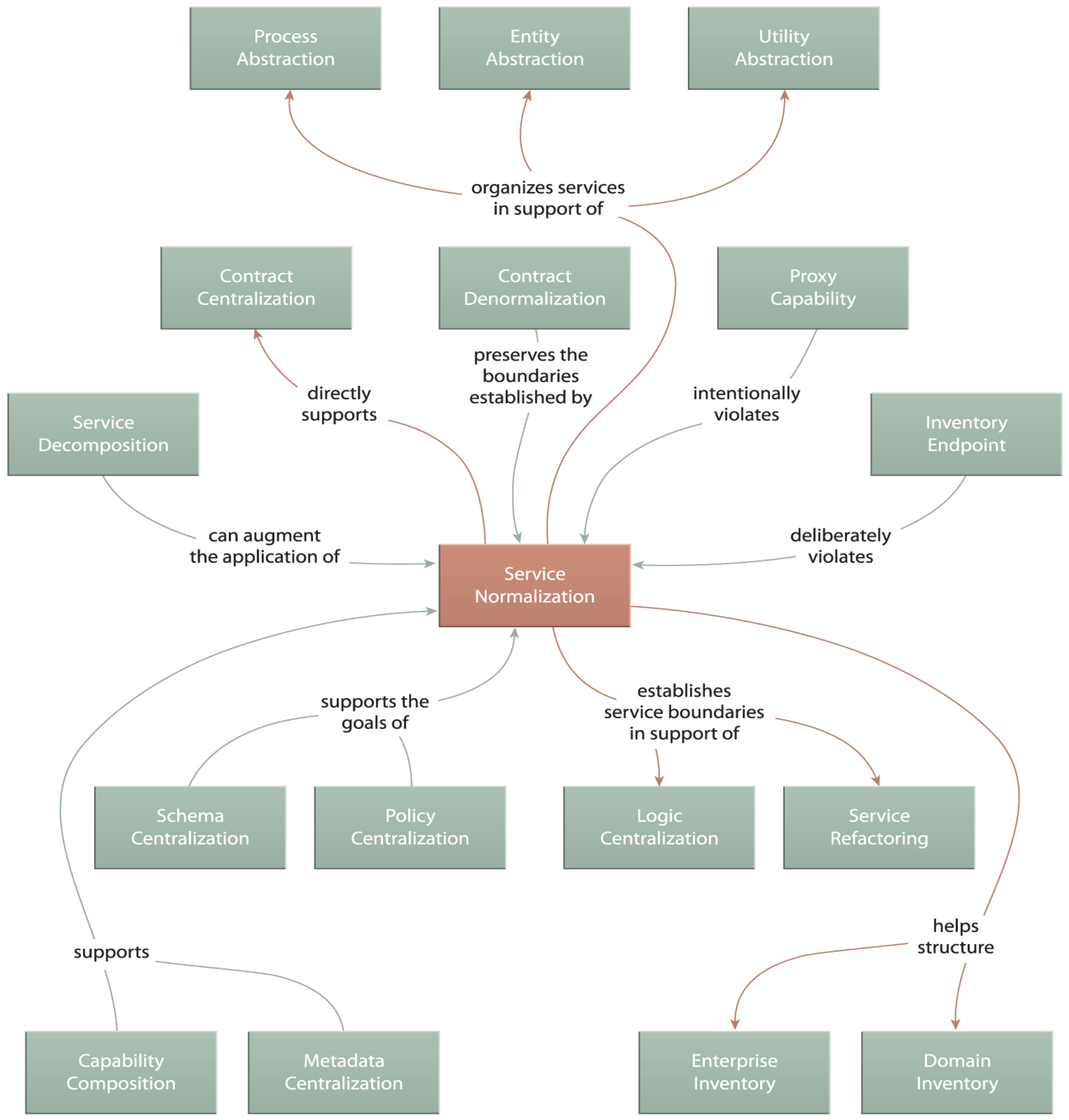
- Pattern = solution to a problem
  - Style = generic
    - Does not have to be associated with a problem
- Style defines general architecture of an application
  - Usually, an application has one style
    - ...but it can have several patterns
  - Patterns can appear at different scales
    - High level (architectural patterns)
    - Design (design patterns)
    - Implementation (idioms)
    - . . .

Pattern vs Style

- Styles, in general, are independent of each other
- A pattern can be related with other patterns
  - A pattern composed of several patterns
  - Interactions between patterns

Pattern languages and catalogs

- Pattern catalog
  - A set of patterns about a subject
    - It does not have to be exhaustive
- Pattern language
  - A full pattern catalog about some subject
    - Goal: document all the possibilities
    - They usually include relationships between patterns
      - Graphical map

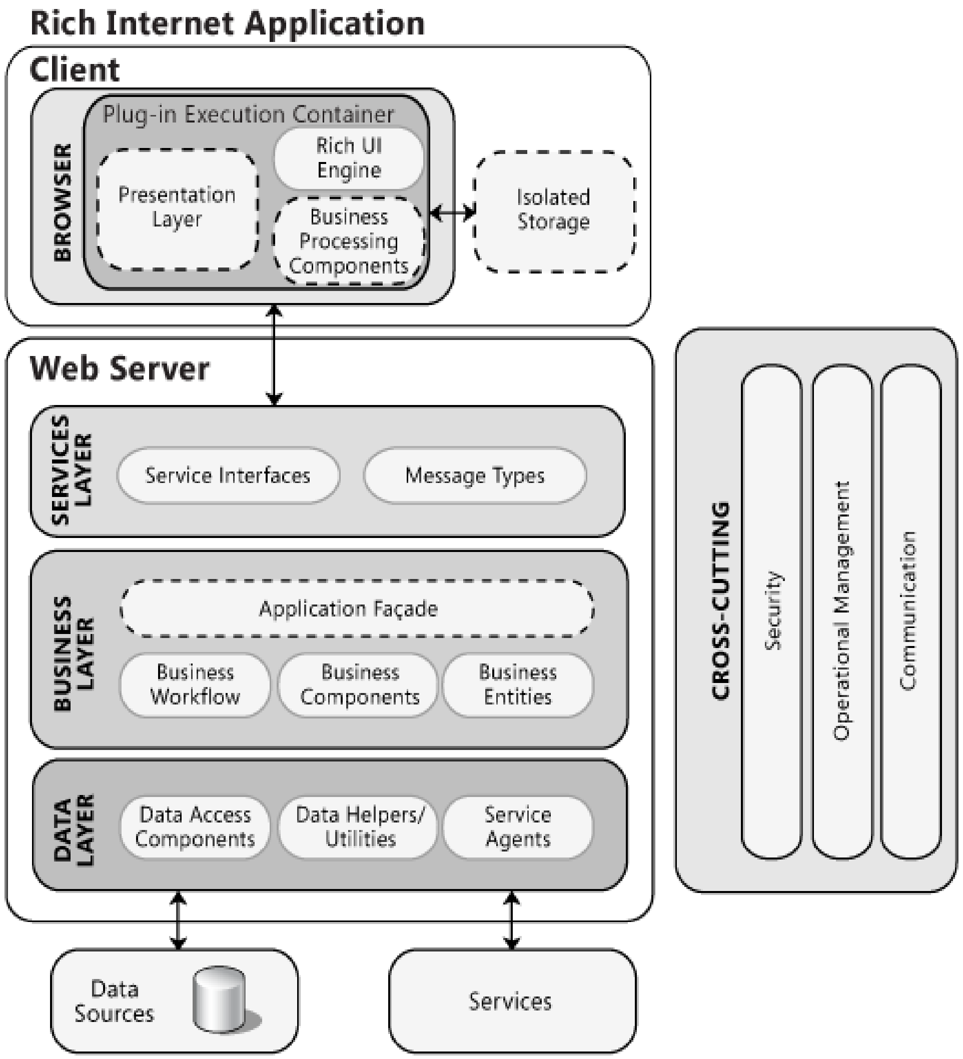


Example of pattern language
Source: "SOA with REST" book

Reference architectures

Blueprints that provide the overall structure for particular types of applications

- They contain several patterns
- Can be the de-facto standard in some domains



Fuente: Microsoft Application Architecture Guide, 2nd Ed.

Externally developed components

- Technology stacks or families
  - MEAN (Mongo,Express,Angular,Node), LAMP (Linux,Apache,MySQL,PHP),...
- Products:
  - COTS: Commercial Off The Self
  - FOSS: Free Open Source Software
    - Be careful with licenses
- Application frameworks
  - Reusable software component
- Platforms
  - Provide complete infrastructure to build & execute applications
    - Example: JEE, Google Cloud

ADD: Attribute-driven design

Defines a software architecture based on QAs

Recursive decomposition process

At each stage tactics and patterns are chosen to satisfy a set of QA scenarios

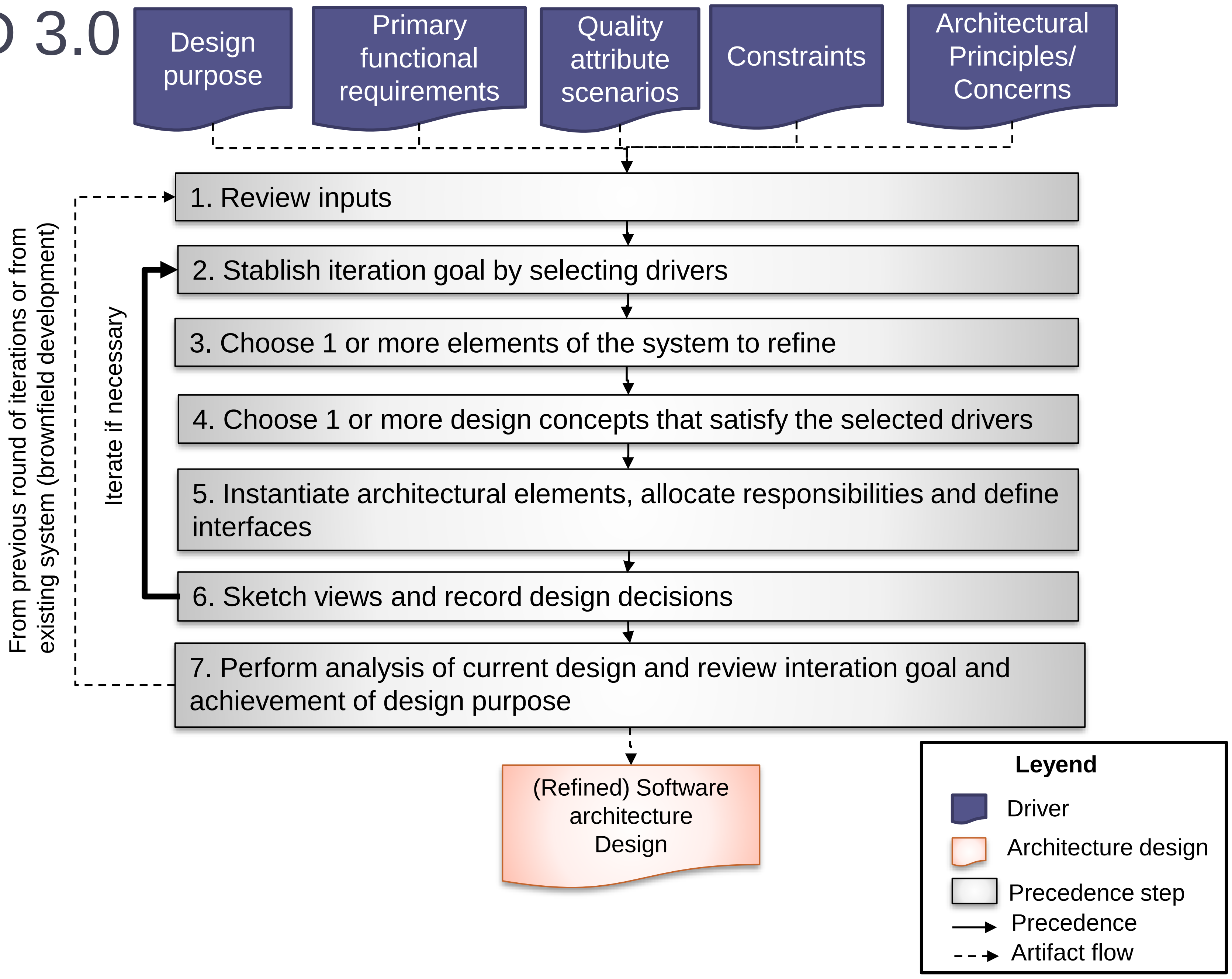
Input

- QA requirements
- Constraints
- Architectural significant functional requirements

Output

- First levels of module decomposition
- Various views of the system as appropriate
- Set of elements with assigned functionalities and the interactions among the elements

ADD 3.0


Design purpose
Primary functional requirements
Quality attribute scenarios
Constraints
Architectural Principles/ Concerns
1. Review inputs
2. Stablish iteration goal by selecting drivers
3. Choose 1 or more elements of the system to refine
4. Choose 1 or more design concepts that satisfy the selected drivers
5. Instantiate architectural elements, allocate responsibilities and define interfaces
6. Sketch views and record design decisions
7. Perform analysis of current design and review interation goal and achievement of design purpose
From previous round of iterations or from existing system (brownfield development)
Iterate if necessary
(Refined) Software architecture Design
Leyend
Driver
Architecture design
Precedence step
Precedence
Artifact flow

Record design decisions

Every design decision is *good enough* but seldom optimal

- It is necessary to record justification and risks affected
- Things to record:
  - What evidence was provided to justify the decision?
  - Who did that?
  - Why were shortcuts taken?
  - Why were trade-offs made?
  - What assumptions did you made?

Driver	Design decisions and location	Rationale and assumptions
QA-1	Introduce concurrency (tactic) in the `TimeServerConnector` and `FaultDetectionService`	Concurrency should be introduced to be able to receive and process several events simultaneously
QA-2	Use of a messaging pattern through the introduction of a message queue in the communications layer	Although the use of a message queue may seem to go against the performance imposed by the scenario, it hill be helpful to support QA-3
...	...	...

Architectural issues

Risks

Unknowns

Problems

Technical debt

Gaps in understanding

Erosion

Drift

Risks

- Risk = something bad that might happen but hasn't happened yet
- Risks should be identified and recorded
  - Risks can appear as part of QA scenarios
- Risks can be mitigated or accepted
  - If possible, identify mitigation tasks

Unknowns

- Sometimes we don't have enough information to know if an architecture satisfies the requirements
  - Under-specified requirements
  - Implicit assumptions
  - Changing requirements
  - ...
- Architecture evaluations can help turn unknown unknowns into known unknowns

Problems

- Problems are bad things that have already passed
  - They arise when one makes design decisions that just doesn't work out the desired way
  - They can also arise because the context changed
    - A decision that was a good idea but no longer makes sense
- Problems can be fixed or accepted
  - Problems that are not fixed can lead to technical debt

Technical debt

- Debt accrued when knowingly or unknowingly wrong or non-optimal design decisions are taken
  - If one pays the instalments the debt is repaid and doesn't create further problems
  - Otherwise, a penalty in the form of interest is applicable
    - If one is not able to pay the bill for a long time the total debt is so large that one must declare bankruptcy
    - In software terms, it would mean the product is abandoned
- Several types:
  - Code debt: Bad or inconsistent coding style
  - Design debt: Design smells
  - Test debt: Lack of tests, inadequate test coverage,...
  - Documentation debt: No documentation for important concerns, outdated documentation,...

Gaps in understanding

- They arise when what stakeholders think about an architecture doesn't match the design
  - In rapidly evolving architectures gaps can arise quickly and without warning
- Gaps can be addressed though education
  - Presenting the architecture
  - Asking questions to stakeholders

Architectural erosion (drift)

- Gap between designed and as-built architecture
  - The implemented system almost never turns out the way the architect imagined it
  - Without vigilance, the architecture drifts from the planned design a little bit every day until the implemented system bears little resemblance to the plan
- Architecturally evident code can mitigate drift

Contextual drift

- It happens any time business or context drivers change after a design decision has been taken
  - It is necessary to continually revisit requirements
  - Evolutionary architecture

Architecture evaluation

ATAM (Architecture Trade-off Analysis Method)

Architecture evaluation method

Simplified version of ATAM:

- Present business drivers
- Present architecture
- Identify architecture approaches
- Generate quality attribute utility tree
- Analyze architectural approaches
- Present results