EN
English

University of Oviedo

Universidad de Oviedo

School of Computer Science

Achieving software architecture

SOFTWARE
ARCHITECTURE

Course 2020/21

Jose E. Labra Gayo

School of Computer Science

Achieving software architecture

- Design concepts
  - Tactics, styles, patterns, reference architectures
  - Externally developed components
- Methodologies
  - ADD
- Making decisions
- Architectural issues
- Architecture evaluation

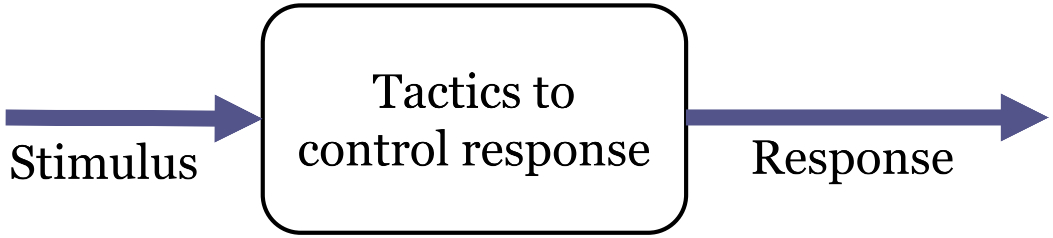
Tactics

Design techniques to achieve a response to some quality attributes

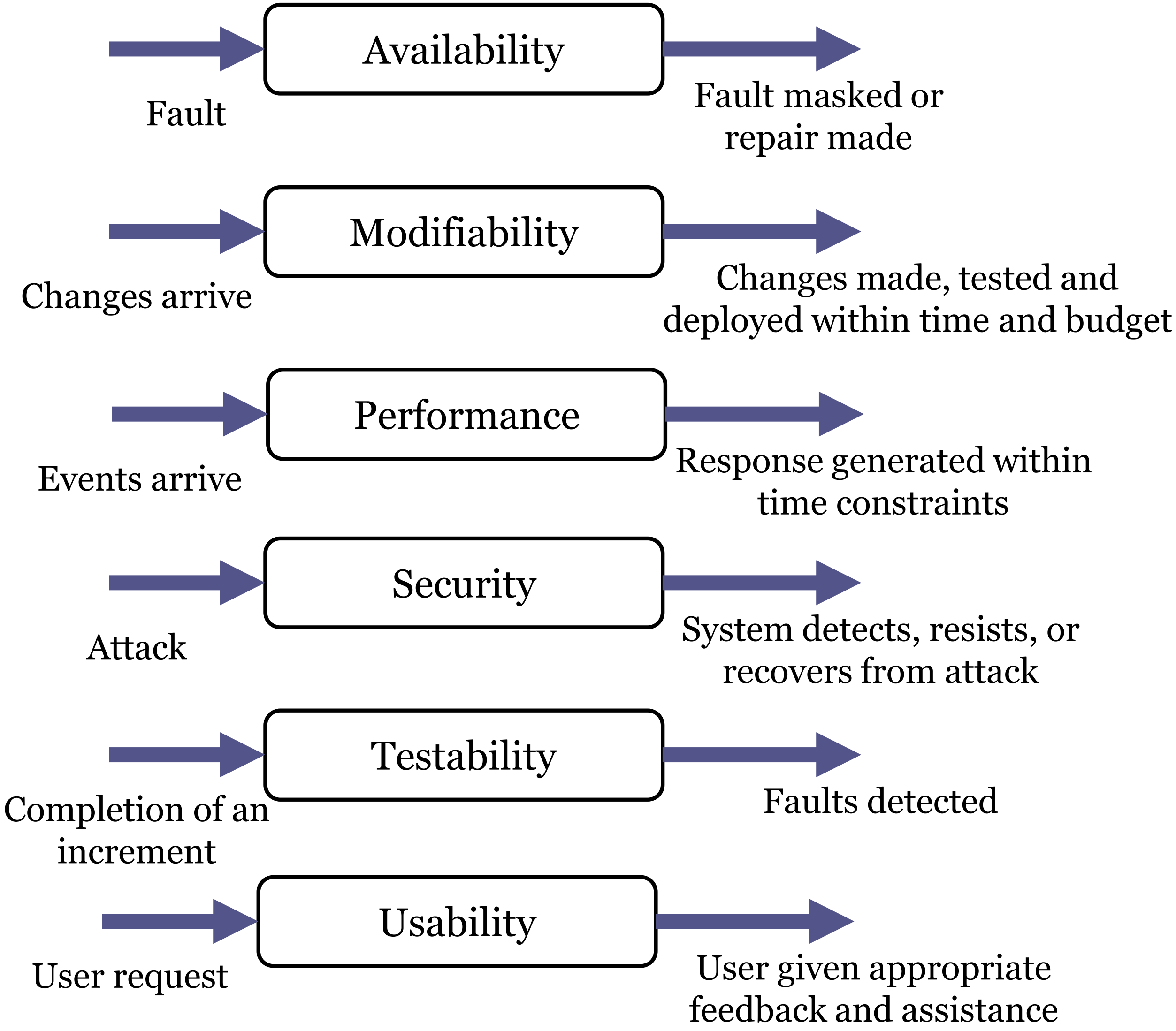
Tactics focus on a single quality attribute response

- 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

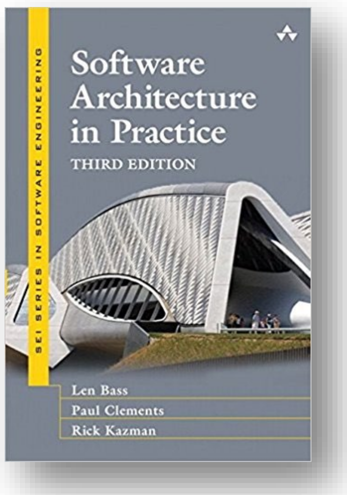
Completion of an increment → Testability → Faults detected

User request → Usability → User given appropriate feedback and assistance

University of Oviedo

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

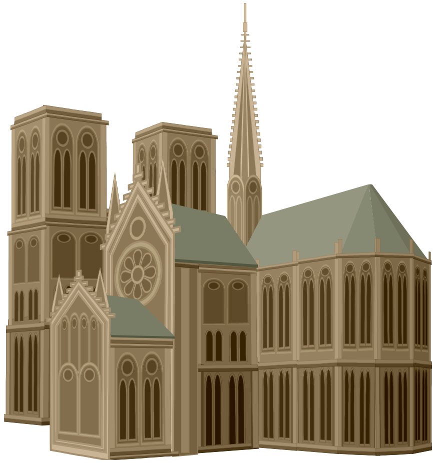

Software Architecture in Practice
THIRD EDITION
SEI SERIES IN SOFTWARE ENGINEERING
Len Bass
Paul Clements
Rick Kazman

http://www.ece.ubc.ca/~matei/EECE417/BASS/ch05lev1sec1.html
https://www.cs.unb.ca/~wdu/cs6075w10/sa2.htm

School of Computer Science

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

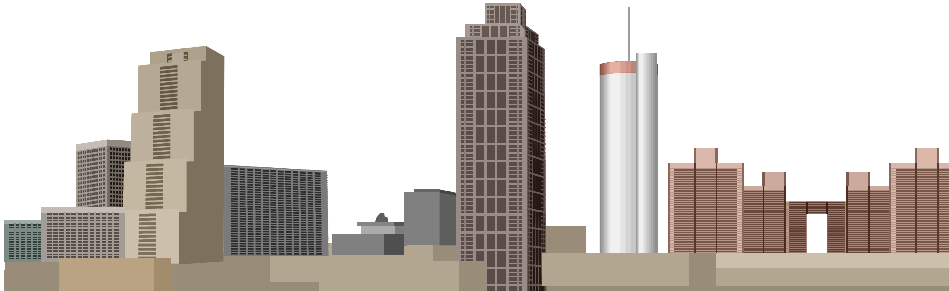
In practice, pure styles rarely appear

Usually, systems deviate from pure styles...

...or combine several architectural styles

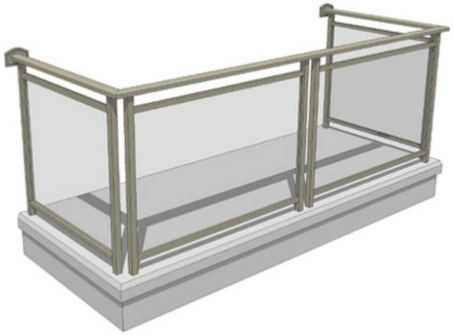
It is important to understand pure styles in order to:

- 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

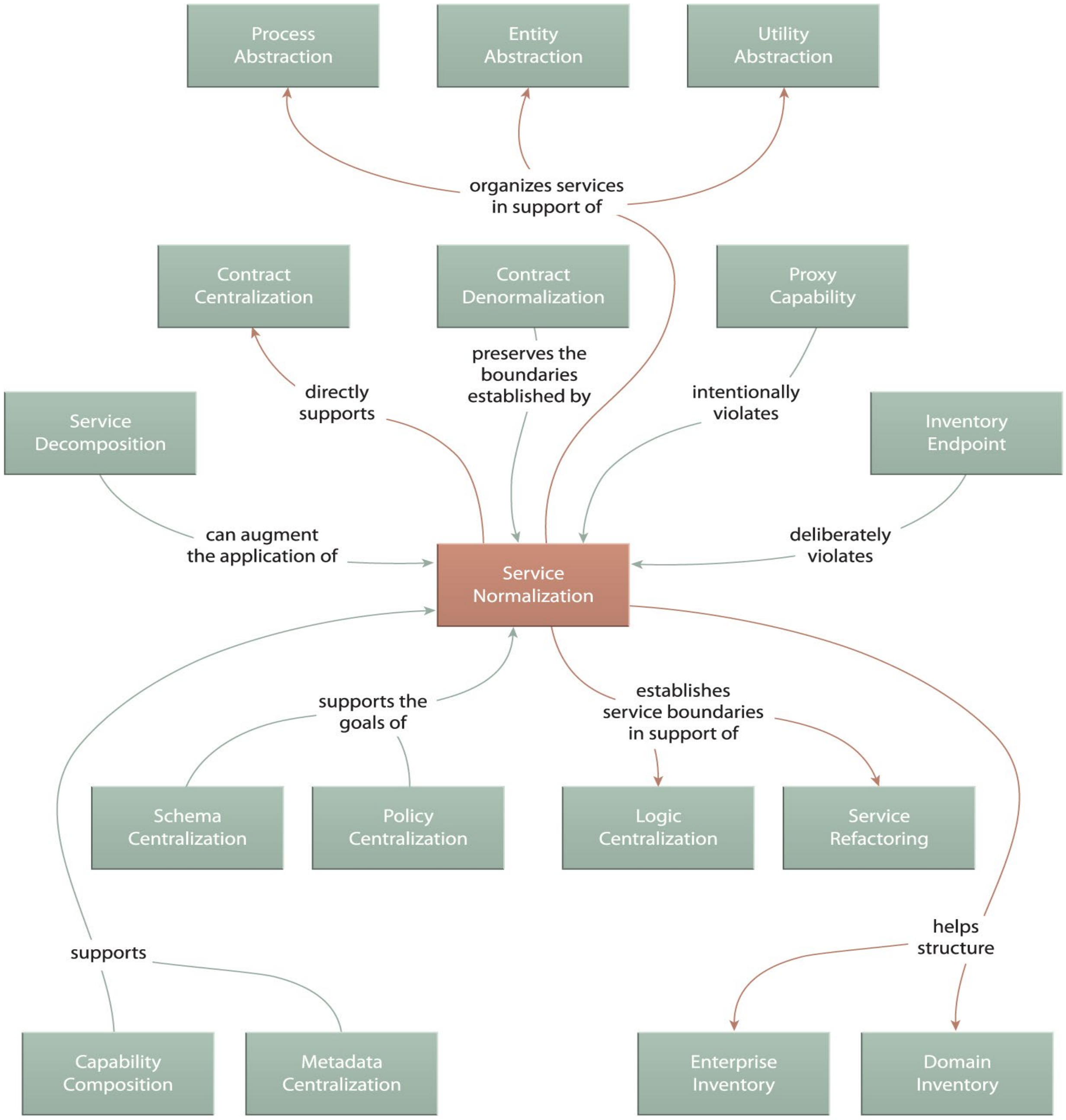
- Pattern = solution to a problem
  - Style = generic
    - Does not have to be associated with a problem
- Style defines general architecture of an application
  - Usually, an application has one style
    - ...but it can have several patterns
  - Patterns can appear at different scales
    - High level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

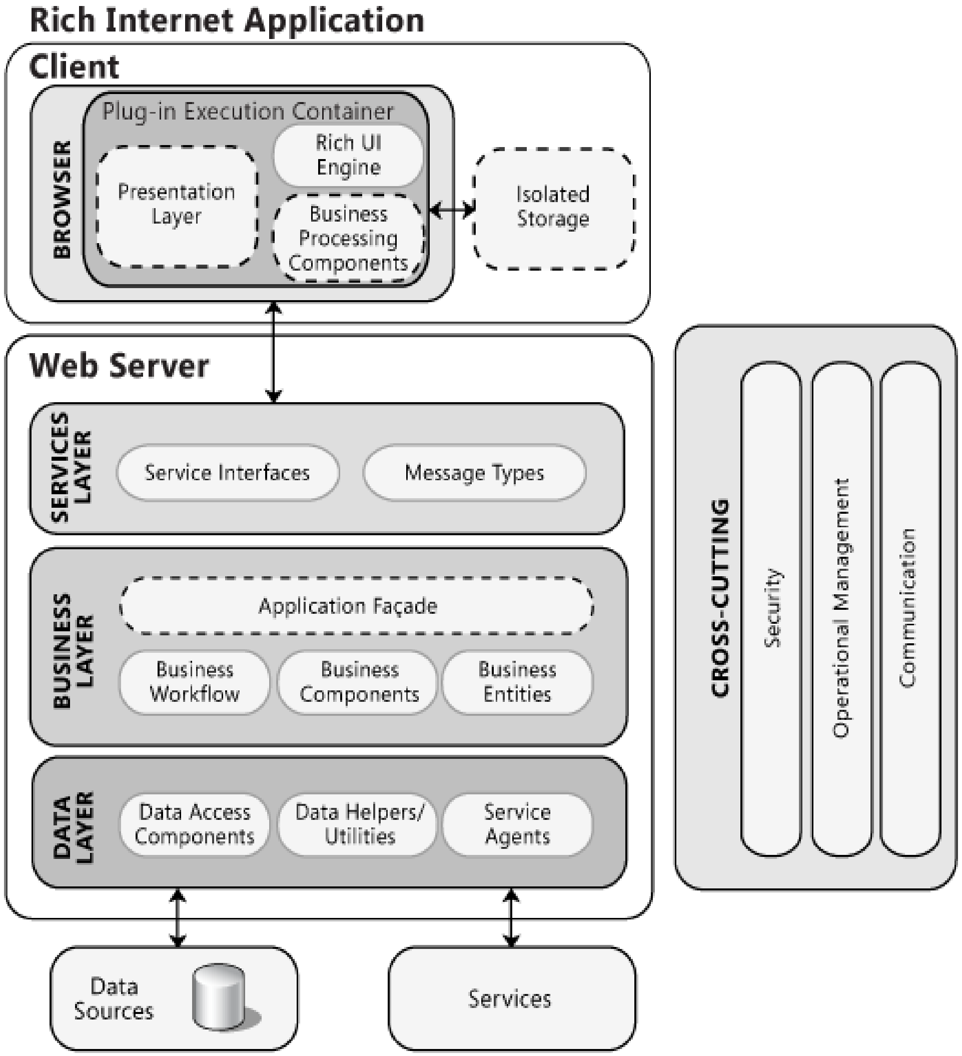
- Pattern catalog
  - A set of patterns about a subject
    - It does not have to be exhaustive
- Pattern language
  - A full pattern catalog about some subject
    - Goal: document all the possibilities
    - They usually include relationships between patterns
      - Graphical map



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
  - Complete infrastructure to build & run applications
    - Example: JEE, Google Cloud
- Libraries

Attribute driven design

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

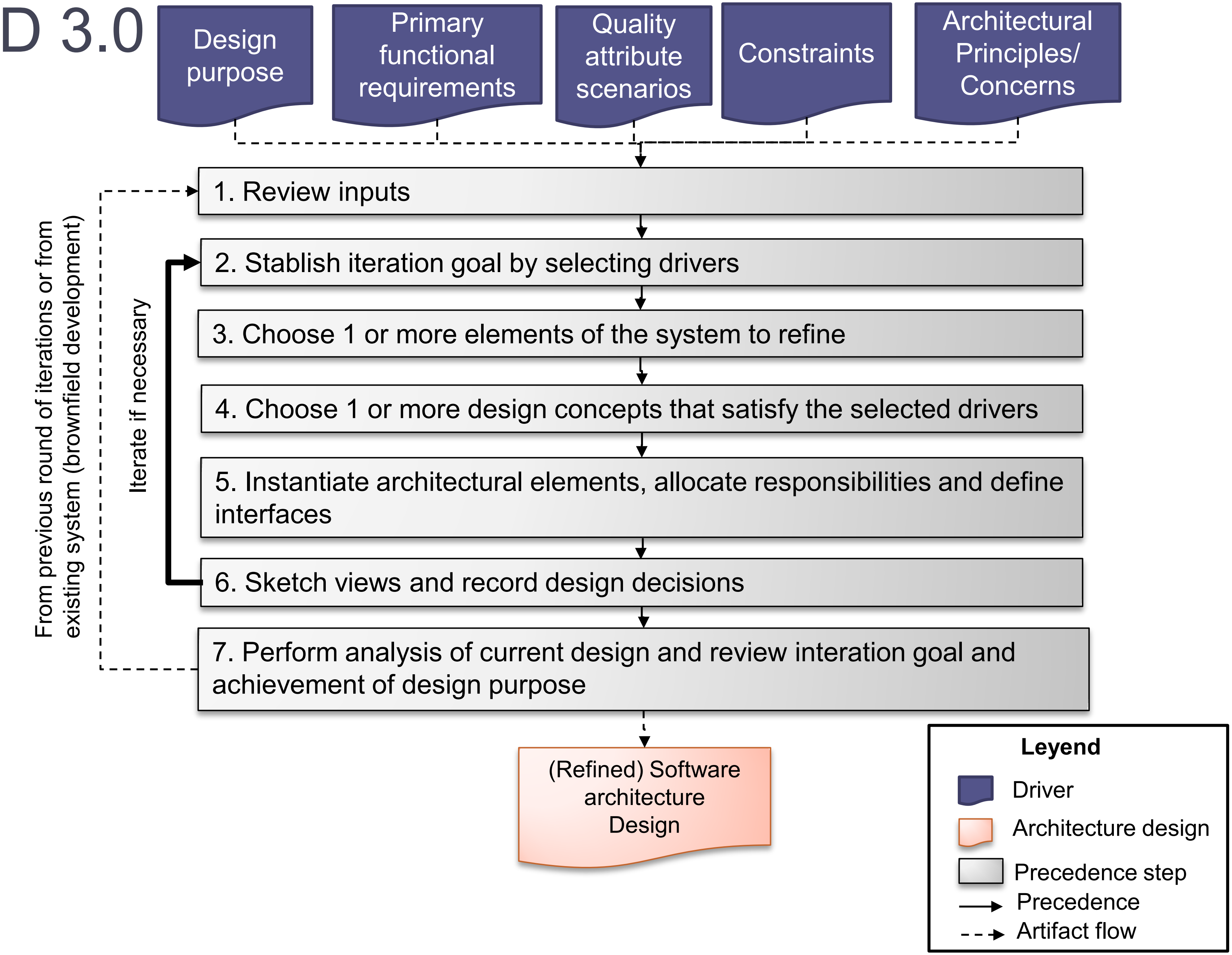
Input

- 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

Iterate if necessary

From previous round of iterations or from existing system (brownfield development)

(Refined) Software architecture Design

Leyend

Driver

Architecture design

Precedence step

Precedence

Artifact flow

University of Oviedo

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
. . .	. . .	. . .

School of Computer Science

University of Oviedo

Architectural decision records

Templates: https://adr.github.io/

Basic structure:

- Title
  - Short descriptive title
- Status
  - Proposed, accepted, superseded
- Context
  - What is forcing to make the decision
  - Include alternatives
- Decision
  - Decision and corresponding justification
- Consequences
  - Expected impact of the decision

For drafts, it may be useful to use RFCs (Request for comments)

School of Computer Science

Architectural issues

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

Risk assessment table

Assess risks in two dimensions:

- Impact of the risk
- Likelihood of that risk appearing

The values can be: low (1), medium (2), high (3)

Likelihood of risk occuring

Overall impact of risk

	Low (1)	Medium (2)	High (3)
Low (1)	1	2	3
Medium (2)	2	4	6
High (3)	3	6	9

Example of risk assessment

Risk criteria	Customer registration	Order Fulfillment
Scalability	2	1
Availability	3	2
Performance	4	3
Security	6	1
Data integrity	9	1

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

Architectures evaluation

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
- Analyse architectural approaches
- Present results

Cost Benefit Analysis Method (CBAM)

1. Choose scenarios and architectural strategies
2. Assess quality attribute benefits
3. Quantify the benefits of architectural strategies
4. Quantify the costs and schedule implications of the architectural strategies
5. Calculate the desirability of each option
6. Make architectural design decisions