University of Oviedo

Universidad de Oviedo

EN
English

Communicating
Software Architecture

SOFTWARE
ARCHITECTURE

School of Computer Science

2024-25

Jose E. Labra Gayo

Contents

Communicating software architecture

Goal of documentation

Documentation stakeholders

Views

Documentation and agile projects

Guidelines

Documentation approaches

Kuchten 4+1 views

Views and beyond

C4 model

Arc42

Comunicating Software architecture

Architecture is more than code

The code doesn't tell the whole story

Questions the code doesn't answer

How the software fits into existing system landscape?

Why were the technologies chosen?

What's the overall structure of the system?

Where are the components deployed at runtime?

How do the components communicate?

How and where to add new functionality?

What common patterns and principles are used?

How the interfaces with other systems work?

How security/scalability/… has been achieved?

. . .

Goal of documentation

Main goal: communicate the structure

Understand the *big picture*

Create a **shared vision**: team and stakeholders

Common vocabulary

Describe what the sofware is and how is being built

Focus for **technical** conversations about new features

Provide a **map** to navigate the source code

Justify design decisions

Help new developers that join the team

Documentation requirements

Understandable by different stakeholders

Technical and non-technical stakeholders

Reflect the reality

Be careful of the model-code gap

Move fast and adapt to changes

Adapt to agile projects

Evolutionary architecture

Rules for good documentation

Write documentation from reader's point of view
  Find who will be the readers and their expectations
Avoid unnecessary repetition (DRY principle)
Avoid ambiguity
  Explain the notation (or use a standard one)
  For diagrams, use legends
Use a standard organization or template
  Add TBD/To do when necessary
  Organize for easy of reference/links
Record rationale
Keep documentation current
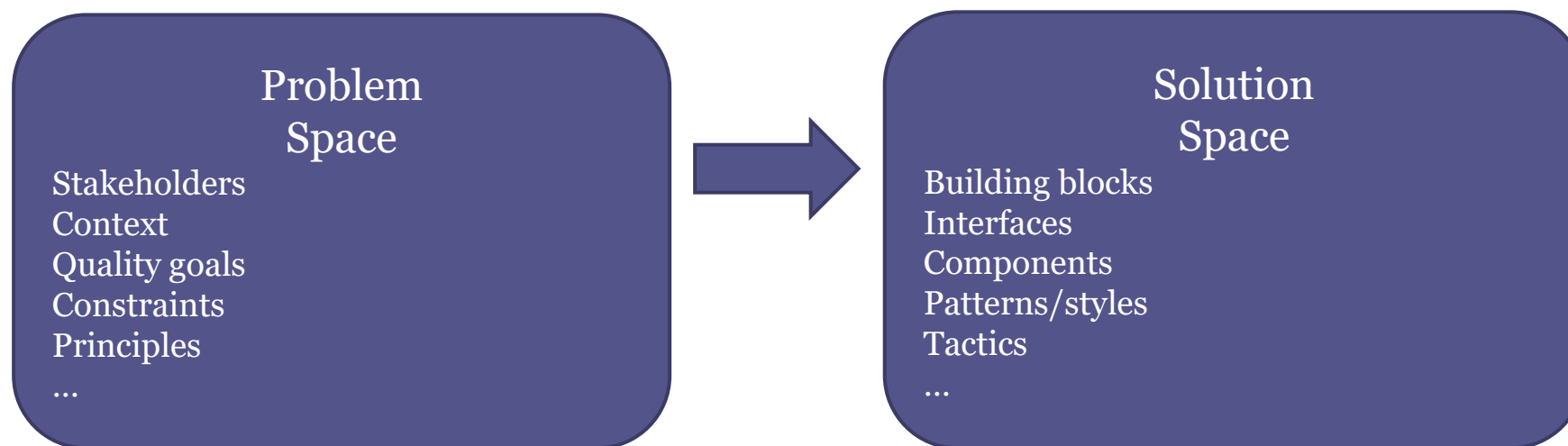
Problem vs Solution space

Software architecture = path from problem to solution

Understand the problem

Design a solution

Rationale for the solutions proposed

Record different design alternatives

Problem
Space

Stakeholders
Context
Quality goals
Constraints
Principles
...

Solution
Space

Building blocks
Interfaces
Components
Patterns/styles
Tactics
...

Views & viewpoints

Software architecture is a complex entity

It cannot be described in a single 1-dimension

It requires several views for different stakeholders

View = A representation of a system with regards to some concerns

Different views support different goals and uses

Viewpoint = A collection of patterns, templates and conventions for constructing a view

Examples: structure, behaviour, deployment

A view is what you see
A viewpoint is where you are looking from

Documenting views

Introduction
- Textual description of the view

Diagram(s)
- Add descriptive title including structures depicted
- Create a legend to explain meaning of symbols
  - Don't forget to explain the lines/arrows

List of elements and responsibilities
- Give descriptive names
- Define your terms (include a glossary)

Rationale

Documenting views

Strive for Consistency and simplicity

Keep elements consistent

Colours, shapes, arrows,…

If you use a colour scheme, follow it consistently

Check names across views,…

Record possible inconsistencies

Avoid too many details

Remember Miller's law

Average person can keep 7 ($\pm$ 2) elements in memory

Tools for diagrams

Sketches

Drawing tools for diagrams

Text-based diagramming tools

Modeling tools

Reverse-engineering the model

Architecture description languages

Sketches

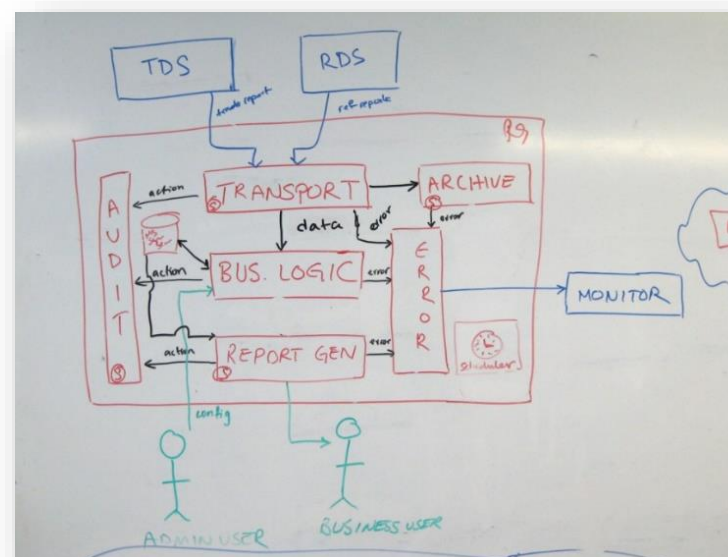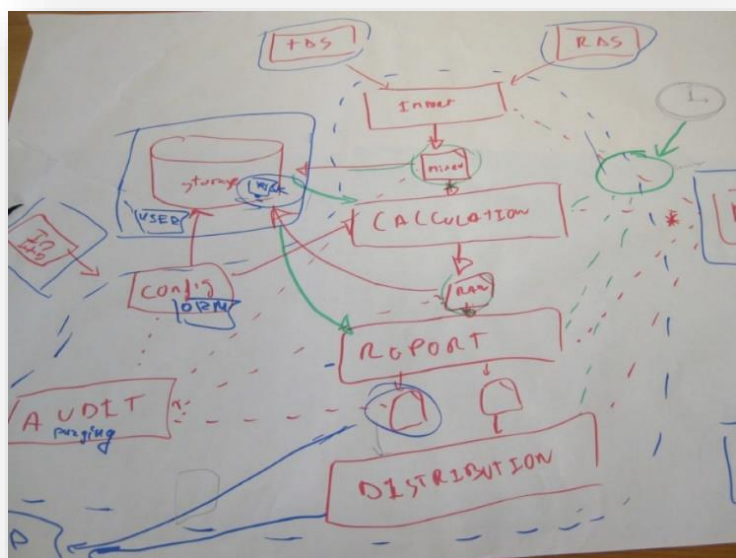Most people start with a sketch on paper or whiteboard

Great way to collaborate and exchange ideas

Usually intended for short lifespan

But sooner or later, they must be recorded

Simple approach: Photos

And later conversion to diagrams or models

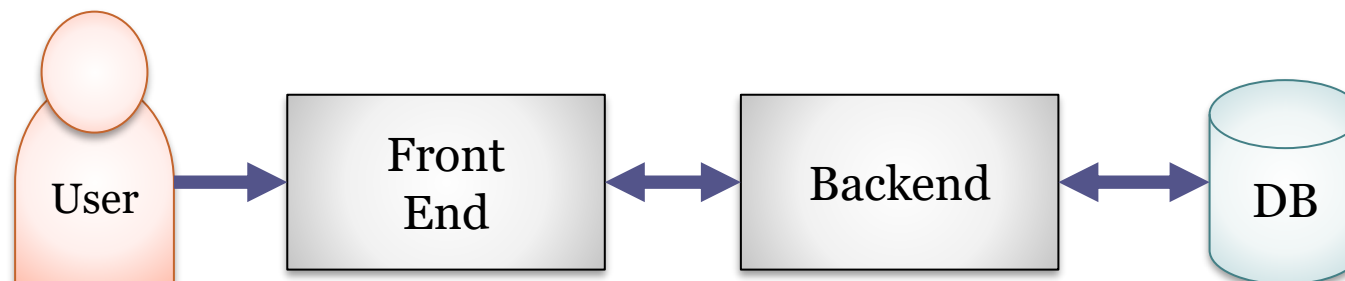Fuente: https://c4model.com/

Drawing tools for Diagrams

Desktop
  Microsoft Visio, Omnigraffle, SimpleDiagrams, …
Web-based:
  draw.io, gliffy, LucidChart,…
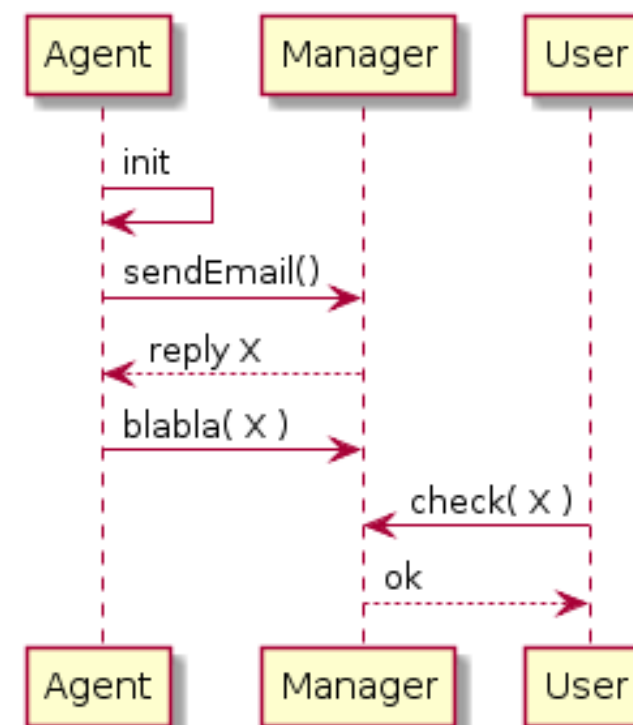Drawing tools of general-purpose tools:
  Word, **Powerpoint**, Keynote,…

Text-based diagramming tools

Usually based on UML

WebSequenceDiagrams, yUML, nomnoml

PlantUML: http://plantuml.com/

```
@startuml
Agent -> Agent : init
Agent -> Manager : sendEmail()
Agent <-- Manager : reply X
Agent -> Manager : blabla( X )
User -> Manager : check( X )
User <-- Manager : ok
@enduml
```

PlantUML Online: https://www.planttext.com/

Modeling tools

Allow to create a model of the software system

Visual representations are generated from model

Alternatives:

Sparx Enterprise Architect, **Visual Paradigm**, Archi, StarUML, ArgoUML, Modelio,…

Usually support different notations

UML, SysML, BPMN, ArchiMate

Useful for up-front design

Good for refactoring & renaming components…

Reverse-engineering the model

Some of the previous modelling tools support this

Static analysis tools:

Structure101, NDepend, Lattix, Sonargraph,…

Create the model based on existing code

Useful to visualize existing codebases

Problem:

Resulting diagrams tend to include too much details

Difficult to see the architecture

Architecture Description Languages: ADLs

Formally define the architecture of a system

Create textual descriptions instead of diagrams

Formal specification

Describes the structure and behaviour

Mostly in academic environments

Not very popular in industrial settings

Some examples:

xArch/xADL (http://isr.uci.edu/projects/xarchuci/)

ACME (http://www.cs.cmu.edu/~able/)

AADL (http://www.aadl.info/)

Software architecture templates

Several possibilities

Kruchten 4+1 views

Views & beyond

C4 model

Arc42 templates ⬅
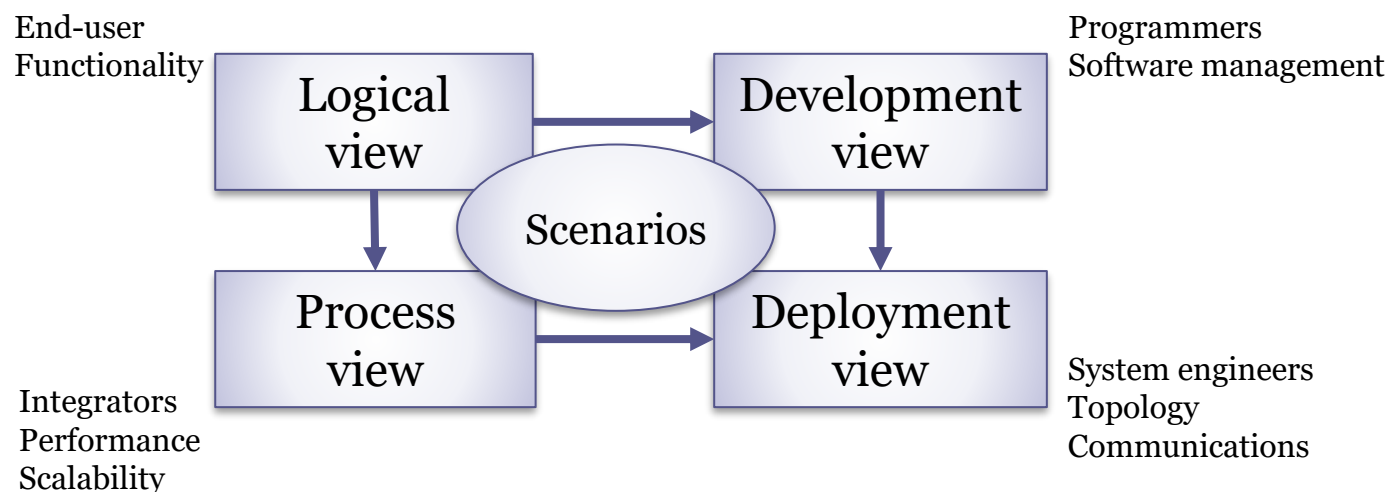
. . .

Kruchten 4+1 views

Embraced as part of Rational Unified Process

5 concurrent views

1 **Logical** view: functionality of the system

2 **Development** view: modules, layers,...

3 **Process** view: execution units, concurrency,…

4 **Physical** view: Infrastructure & deployment topology

(+1) **Scenarios** view: selected use cases or scenarios

End-user
Functionality

Programmers
Software management

Logical view → Development view

Scenarios

Process view → Deployment view

Integrators
Performance
Scalability

System engineers
Topology
Communications

Views and beyond

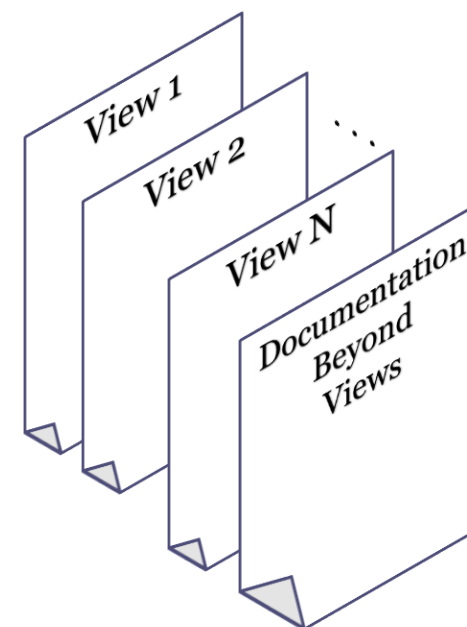Select a set of viewpoints
According to stakeholder's needs
Define views according to those viewpoints
Add a "Beyond views" document
Overall architecture
Information about how views relate

. . .

C4 model (https://c4model.com/)

Describe

Context: System or enterprise context diagram

Container diagram: high level shape

Components diagram: zoom and decompose

Code: UML class diagrams, ER diagrams, …

Documentation guidebook

Context	Code
Functional overview	Data
Quality attributes	Infrastructure architecture
Constraints	Deployment
Principles	Development environment
	Operation and support
	Decision log

Arc42 https://arc42.org/

Structure to document software systems
Goal: Clear, simple and effective
Templates available for several systems

Asciidoc

Word (docx)

Markdown

LaTeX

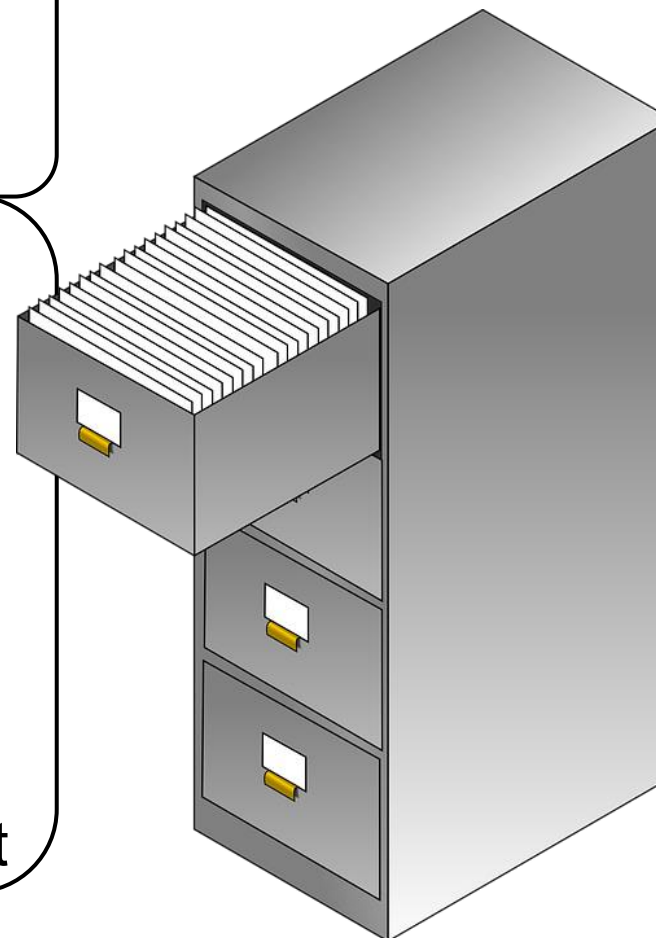ReStructuredText

Confluence

. . .

Arc42 overview

Problem

1.- Introduction and goals

2.- Constraints

3.- Context & scope

Solution

4.- Solution strategy

5.- Building block view

6.- Runtime view

7.- Deployment view

8.- Crosscutting concepts

9.- Architectural decisions

10.- Quality requirements

11.- Risks and technical debt

12.-Glossary

Picture source: https://commons.wikimedia.org/wiki/File:Ficherosclasicoscatalogo.JPG

University of Oviedo

School of Computer Science

1 - Introduction and goals

Short description of:
- Requirements
- Main quality goals
- Stakeholders

Picture source: https://arc42.org/overview/

1 Introduction and goals
1.1 Requirements overview

Short description of functional requirements

Use-case tables

It can link to existing requirements documents

Full requirement documents are usually longer

Select architecturally significant requirements

1 Introduction and goals
1.2 Main quality goals

Enumerate the main quality goals

Quality goals:

Main quality attributes that the system needs to achieve

Format: A simple table can suffice

Example:

https://biking.michael-simons.eu/docs/index.html#_quality_goals

University of Oviedo

School of Computer Science

How to choose quality attributes?

Quality attribute workshops

Involve stakeholders to prioritize quality attributes

It may be helpful to distinguish

Runtime quality attributes

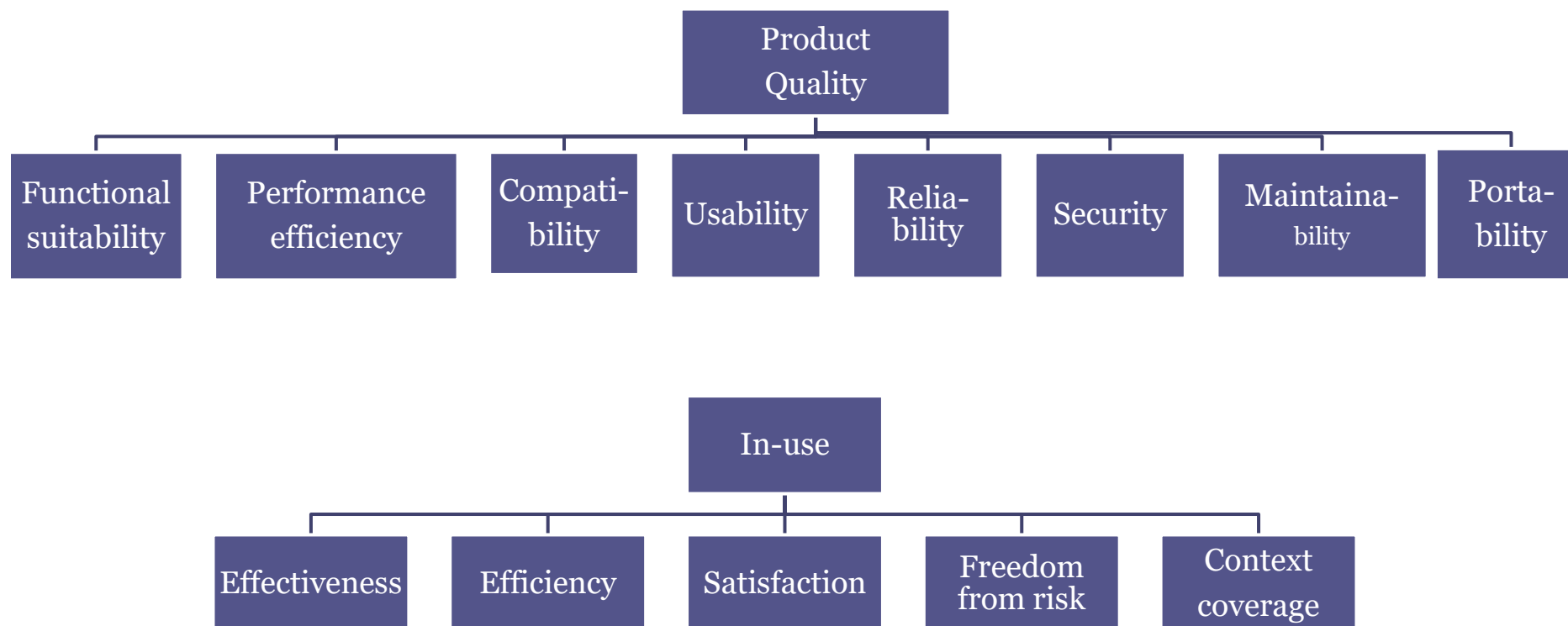Performance, security, availability, usability,…

Non-runtime quality attributes

Modifiability, portability, reusability,testability

Business quality attributes

Cost, schedule, time-to-market, …

How to choose quality attributes?

ISO-25010 Software Quality Model
2 parts: Product quality, Quality in-use

1 Introduction and goals
1.3 Stakeholders

Stakeholder: person who affects, is affected or can contribute to the system and its architecture

Make explicit expectations and motivation

Format: table or map

Stakeholder	Description	Expectations, motivations
. . .	. . .	. . .

2 - Constraints

Anything that constrains teams in design and implementation decisions

Sometimes at organization level

Decisions already taken

Format: a table with explanations

Can be divided in organizational, technical, etc.
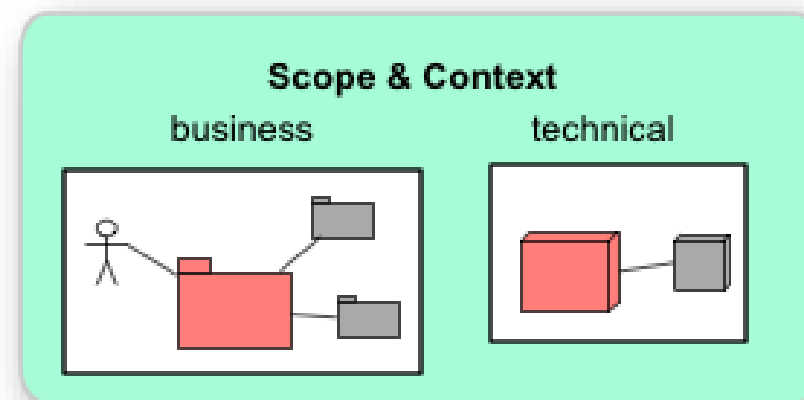
Constraint	Explanation
. . .	. . .

Picture source: https://arc42.org/overview/
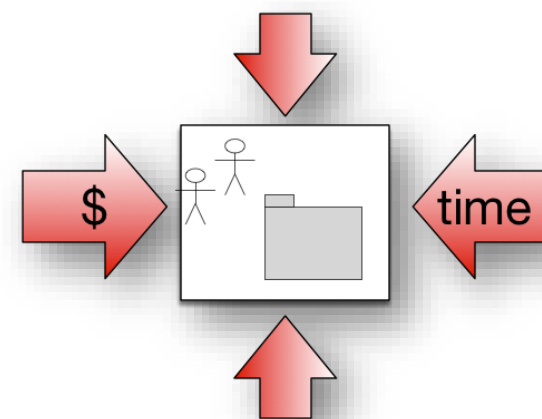
3 - Context and scope

Delimits the system from external partners

    Neighbouring users and systems

Specifies the external interfaces

Business and technical perspective

Picture source: https://arc42.org/overview/

3. Context and Scope
3.1 Business context

Specify all partners involved in the environment

Format: Diagram or table

   Diagrams that show the system as a black box

Optional: Explanation of external interfaces

NOTE:
The business context is mandatory

3 Context and scope
3.2 Technical context

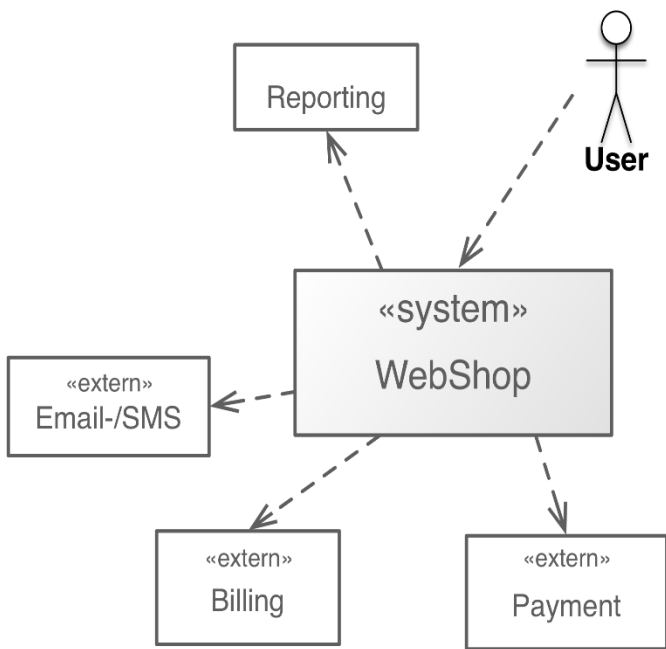Specify Technical interfaces that link the system with the environment

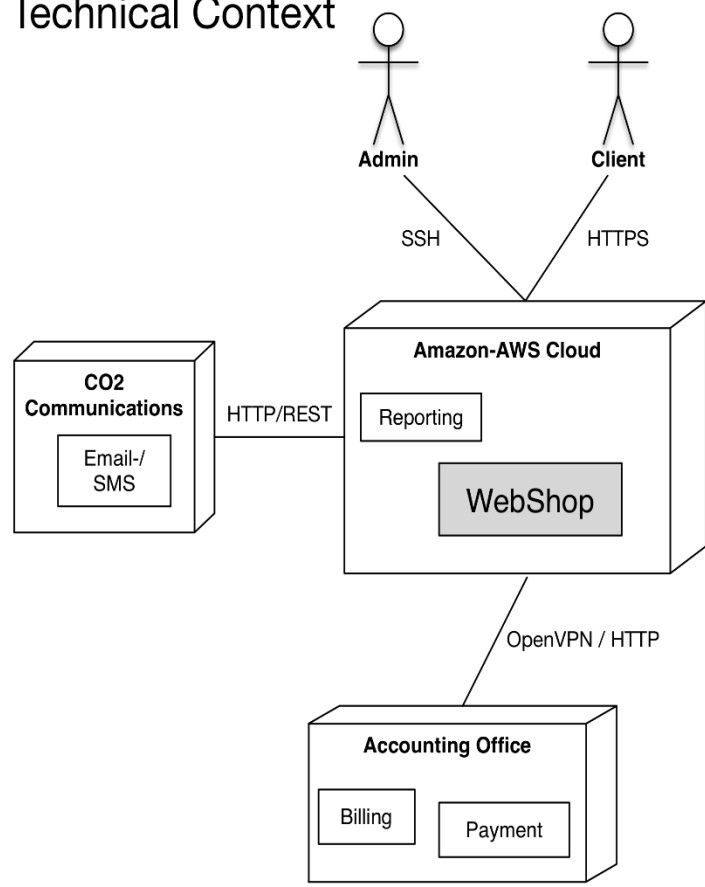Format: Diagram or table

Usually: UML deployment diagrams

NOTE:
The technical context diagram is optional
The deployment view can be enough

Business context vs technical context

4 - Solution strategy

Summary of fundamental decisions and strategies
Can include:
- Technology
- Top-level decomposition
- Approaches to achieve top quality goals
- Relevant organizational decisions.
Format: short text description
Keep explanations of key decisions short

Picture source: https://commons.wikimedia.org/wiki/File:Light_Bulb_or_Idea_Flat_Icon_Vector.svg

5 - Bulding block view

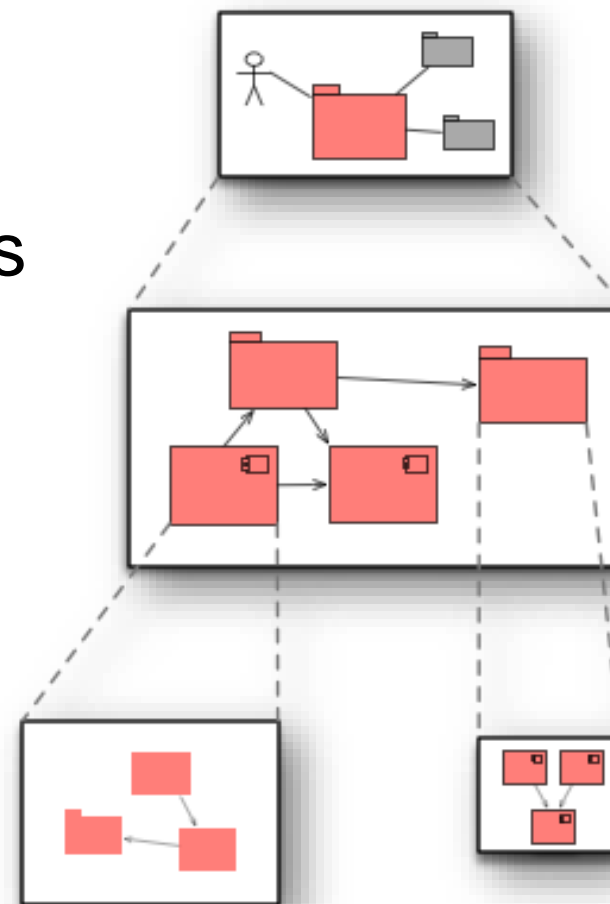Static decomposition of system

Modules of the system

Hierarchy of white boxes containing black boxes

Format:

Start with overall overview diagram

Decompose into other diagrams

Usually: UML Component diagrams

Source: https://arc42.org/overview/

6 - Runtime view

Behavior of building blocks as scenarios
Important use cases or features
Interactions at critical external interfaces
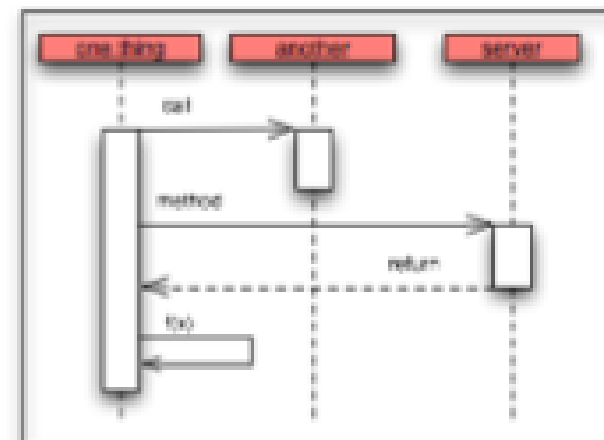   Error and exception behavior.
Format:
   Many notations
      Natural language (list of steps)
      UML sequence diagrams
      Flowcharts
      BPMN

      …

Source: https://arc42.org/overview/
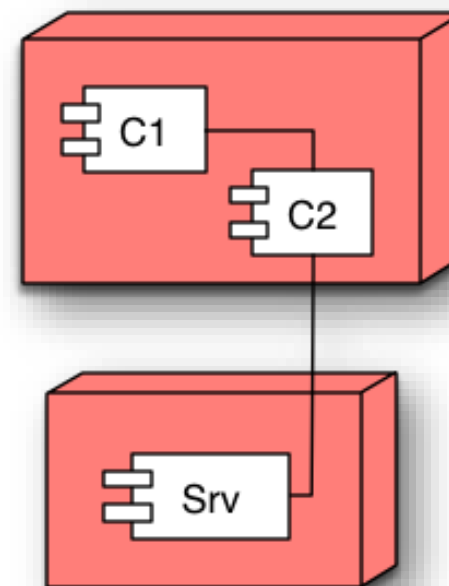
7 - Deployment view

Technical infrastructure with environments, computers,
processors, topologies.
Mapping of (software) building blocks to infrastructure
Format:
Usually: UML deployment diagrams
Add mapping tables

Source: https://arc42.org/overview/
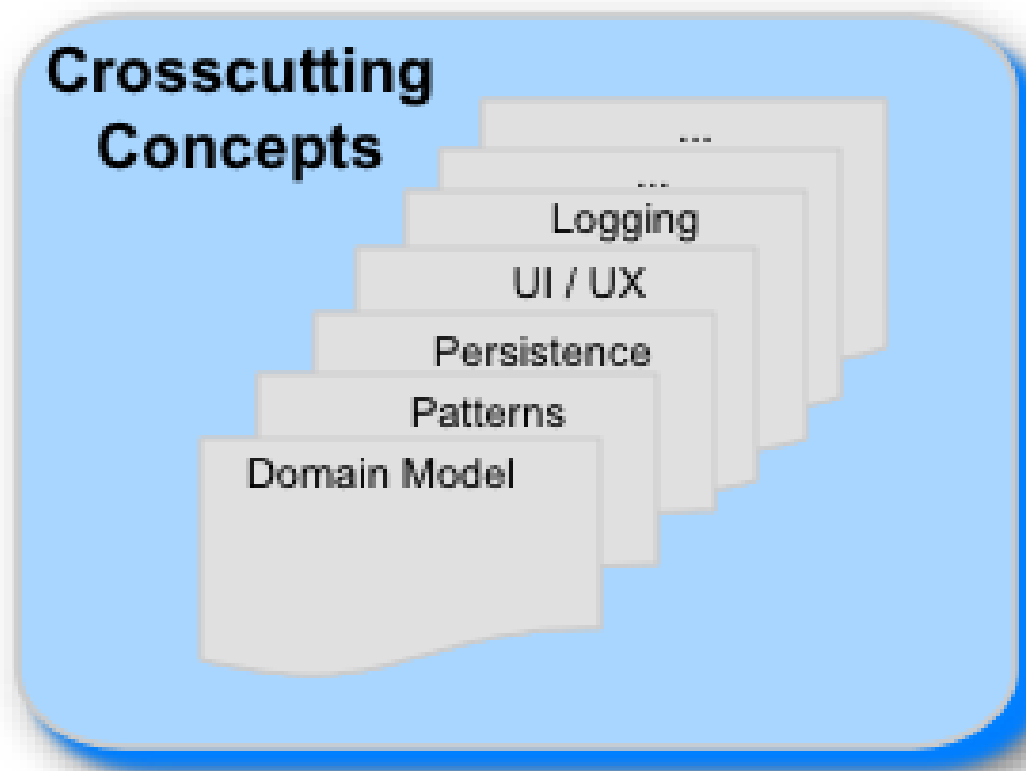
8 - Crosscutting concepts

Approaches relevant in multiple parts of system

Topics like:

Domain model

Architecture pattern and styles

Specific rules

Crosscutting Concepts

...
....
Logging
UI / UX
Persistence
Patterns
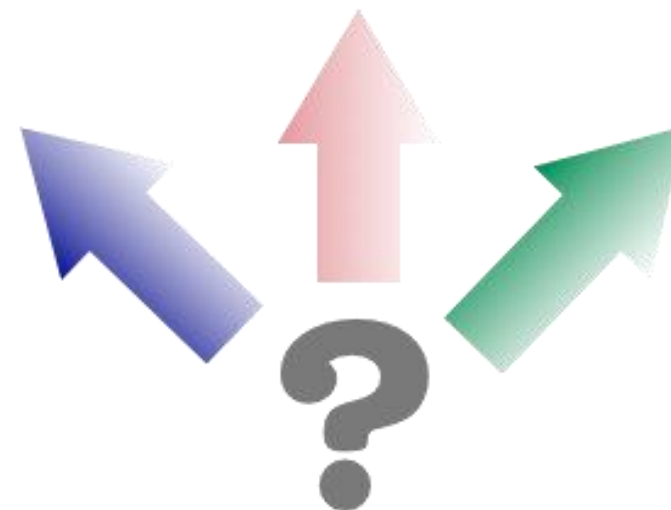Domain Model

Source: https://arc42.org/overview/

9 Architectural decisions

Important, expensive, critical, large scale or risky architecture decisions
Include rationale for the decisions
Format:

List or table ordered by importance
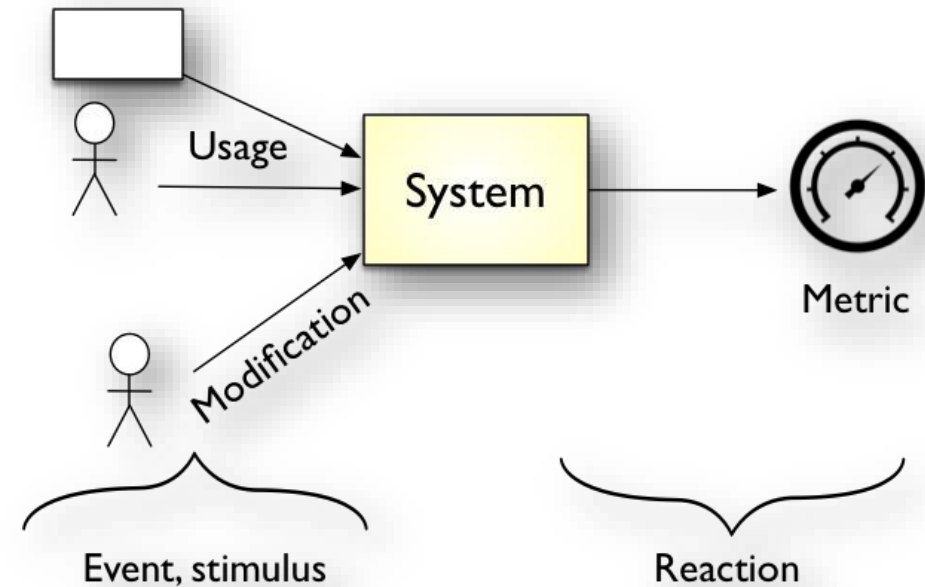Architecture decision record for important decisions

Source: https://arc42.org/overview/

10 - Quality requirements

Quality requirements as scenarios

Quality tree to provide high-level overview

The most important quality goals should have been described in section 1 (quality goals)

10. Quality requirements
10.1 Quality tree

A quality tree with quality scenarios as leafs

Include priorities for an overview

Sometimes, large number of quality requirements.

Format:

   A mind map with quality categories as branches

   Include links to scenarios of the following section

10. Quality requirements
10.2 Quality scenarios

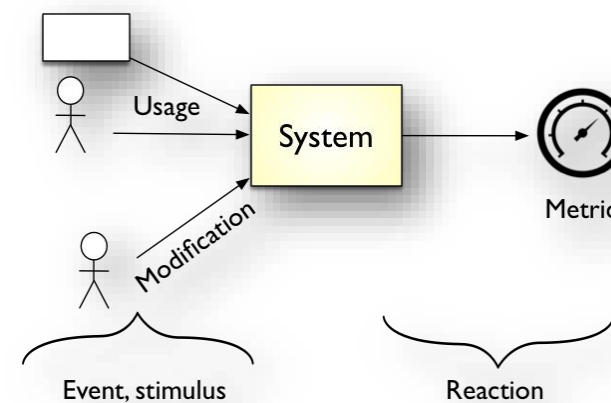Scenarios describe what should happen when a stimulus arrives at the system.

2 types:

Usage: runtime reaction to a certain stimulus.

"The system reacts to a user's request within 1 sec."

Change: modification of the system or its environment

"A new user type must be added"

Format: Tabular or free form text.

Source: https://arc42.org/overview/

11 - Risks and technical debt

Known technical risks or technical debt

What potential problems exist?

What does the development team feel miserable about?

Format:

List or risks/technical debts

Include suggested measures to minimize, mitigate or avoid risks or reduce technical debts.

Source: https://arc42.org/overview/

12 - Glossary

Important domain and technical terms

Terms used by stakeholders when discussing the system

Common vocabulary

Translation reference in multi-language environments

Format: table

Term	Definition
. . .	. . .

Source: https://arc42.org/overview/

Architecturally evident coding style

Drop hints about architecture in the code

Allow readers to infer the design from the code

The code should reflect the architecture

Examples:

Components as packages

Modules in different repos/folders

Some tools that check/enforce architectural constraints

https://www.lattix.com/, https://structurizr.com/