 Template

2021-02-22

**About arc42**

arc42, the Template for documentation of software and system architecture.

By Dr. Gernot Starke, Dr. Peter Hruschka and contributors.

Template Revision: 7.0 EN (based on asciidoc), January 2017

© We acknowledge that this document uses material from the arc 42 architecture template, <http://www.arc42.de>. Created by Dr. Peter Hruschka & Dr. Gernot Starke.

This version of the template contains some help and explanations. It is used for familiarization with arc42 and the understanding of the concepts. For documentation of your own system you use better the *plain* version.

# Introduction and Goals

Describes the relevant requirements and the driving forces that software architects and development team must consider. These include

* underlying business goals, essential features and functional requirements for the system
* quality goals for the architecture
* relevant stakeholders and their expectations

## Requirements Overview

## 

Edit, share and discover high-quality, visually stunning images that are not found on other social media platforms. Interact with like-minded people from all over the world.

* Integrated User Management
* Community Features
* iOS / Android mobile app
* Advanced picture editing & customized filters
* Picture sharing
* Subscription model

## Quality Goals

|  |  |  |
| --- | --- | --- |
| **Priority** | Quality | Motivation |
| 1 | Reliability / Stability | App should be stable and function without bugs and errors |
| 2 | Performance Efficiency | Fast load times despite high quality uploads |
| 3 | Operability / Usability | App should be user friendly and easy to understand & navigate |

## Stakeholders

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Role / Name** | Expectations | | User (photography enthusiasts) | upload images and interact with friends & family | | Marketing Manager | high quality picture sharing app that appeals to a wide target audience | | Project Manager | transparent risk management and quality assurance within budget | | Lead Developer | code quality standards & cross platform integration | | UX / UI Designer | consistent, user-centered & accessible design | |  |  |

# Architecture Constraints

|  |  |  |
| --- | --- | --- |
| Type | Constraints | Background and / or motivation n |
| Technical | Implementation in Flutter | * App should be programmed in Flutter to cover iOS and Android |
| Technical | Third party software/filters must be available under compatible open-source license & installable via a package manager | * Developers/architect should be able to check out the sources, compile and run the application without problems. * External dependencies should be available via package manager/installer |
| Operating system | Mobile OS independent development | * Application should run on both iOs and Android |
| Organizational | Configuration and version control / management | Private git repository to manage commits. |

# System Scope and Context

**Contents.**

System scope and context - as the name suggests - delimits your system (i.e. your scope) from all its communication partners (neighboring systems and users, i.e. the context of your system). It thereby specifies the external interfaces.

If necessary, differentiate the business context (domain specific inputs and outputs) from the technical context (channels, protocols, hardware).

**Motivation.**

The domain interfaces and technical interfaces to communication partners are among your system’s most critical aspects. Make sure that you completely understand them.

**Form.**

Various options:

* Context diagrams
* Lists of communication partners and their interfaces.

## Business Context

## 

## Ein Bild, das Text, Diagramm, Reihe, Plan enthält. Automatisch generierte Beschreibung

## Technical Context

**Contents.**

Technical interfaces (channels and transmission media) linking your system to its environment. In addition a mapping of domain specific input/output to the channels, i.e. an explanation with I/O uses which channel.

**Motivation.**

Many stakeholders make architectural decision based on the technical interfaces between the system and its context. Especially infrastructure or hardware designers decide these technical interfaces.

**Form.**

E.g. UML deployment diagram describing channels to neighboring systems, together with a mapping table showing the relationships between channels and input/output.

**<Diagram or Table>**

**<optionally: Explanation of technical interfaces>**

**<Mapping Input/Output to Channels>**

# Solution Strategy

|  |  |
| --- | --- |
| **Goal/Requirements** | Architectural Approach |
| Reliability / Stability | Microservices |
| Performance Efficiency | Content Delivery Network (CDN) |
| Operability / Usability | Single Page Application architecture |
| Cross-platform | Flutter |

# Building Block View

**Content.**

The building block view shows the static decomposition of the system into building blocks (modules, components, subsystems, classes, interfaces, packages, libraries, frameworks, layers, partitions, tiers, functions, macros, operations, datas structures, …) as well as their dependencies (relationships, associations, …)

This view is mandatory for every architecture documentation. In analogy to a house this is the *floor plan*.

**Motivation.**

Maintain an overview of your source code by making its structure understandable through abstraction.

This allows you to communicate with your stakeholder on an abstract level without disclosing implementation details.

**Form.**

The building block view is a hierarchical collection of black boxes and white boxes (see figure below) and their descriptions.



**Level 1** is the white box description of the overall system together with black box descriptions of all contained building blocks.

**Level 2** zooms into some building blocks of level 1. Thus it contains the white box description of selected building blocks of level 1, together with black box descriptions of their internal building blocks.

**Level 3** zooms into selected building blocks of level 2, and so on.

## Whitebox Overall System

Here you describe the decomposition of the overall system using the following white box template. It contains

* an overview diagram
* a motivation for the decomposition
* black box descriptions of the contained building blocks. For these we offer you alternatives:
  + use *one* table for a short and pragmatic overview of all contained building blocks and their interfaces
  + use a list of black box descriptions of the building blocks according to the black box template (see below). Depending on your choice of tool this list could be sub-chapters (in text files), sub-pages (in a Wiki) or nested elements (in a modeling tool).
* (optional:) important interfaces, that are not explained in the black box templates of a building block, but are very important for understanding the white box. Since there are so many ways to specify interfaces why do not provide a specific template for them. In the worst case you have to specify and describe syntax, semantics, protocols, error handling, restrictions, versions, qualities, necessary compatibilities and many things more. In the best case you will get away with examples or simple signatures.

***<Overview Diagram>***

Motivation

*<text explanation>*

Contained Building Blocks

*<Description of contained building block (black boxes)>*

Important Interfaces

*<Description of important interfaces>*

Insert your explanations of black boxes from level 1:

If you use tabular form you will only describe your black boxes with name and responsibility according to the following schema:

|  |  |
| --- | --- |
| **Name** | **Responsibility** |
| *<black box 1>* | *<Text>* |
| *<black box 2>* | *<Text>* |

If you use a list of black box descriptions then you fill in a separate black box template for every important building block . Its headline is the name of the black box.

### <Name black box 1>

Here you describe <black box 1> according the the following black box template:

* Purpose/Responsibility
* Interface(s), when they are not extracted as separate paragraphs. This interfaces may include qualities and performance characteristics.
* (Optional) Quality-/Performance characteristics of the black box, e.g.availability, run time behavior, ….
* (Optional) directory/file location
* (Optional) Fulfilled requirements (if you need traceability to requirements).
* (Optional) Open issues/problems/risks

*<Purpose/Responsibility>*

*<Interface(s)>*

*<(Optional) Quality/Performance Characteristics>*

*<(Optional) Directory/File Location>*

*<(Optional) Fulfilled Requirements>*

*<(optional) Open Issues/Problems/Risks>*

### <Name black box 2>

*<black box template>*

### <Name black box n>

*<black box template>*

### <Name interface 1>

…

### <Name interface m>

## Level 2

Here you can specify the inner structure of (some) building blocks from level 1 as white boxes.

You have to decide which building blocks of your system are important enough to justify such a detailed description. Please prefer relevance over completeness. Specify important, surprising, risky, complex or volatile building blocks. Leave out normal, simple, boring or standardized parts of your system

### White Box *<building block 1>*

…describes the internal structure of *building block 1*.

*<white box template>*

### White Box *<building block 2>*

*<white box template>*

…

### White Box *<building block m>*

*<white box template>*

## Level 3

Here you can specify the inner structure of (some) building blocks from level 2 as white boxes.

When you need more detailed levels of your architecture please copy this part of arc42 for additional levels.

### White Box <\_building block x.1\_>

Specifies the internal structure of *building block x.1*.

*<white box template>*

### White Box <\_building block x.2\_>

*<white box template>*

### White Box <\_building block y.1\_>

*<white box template>*

# Runtime View

**Contents.**

The runtime view describes concrete behavior and interactions of the system’s building blocks in form of scenarios from the following areas:

* important use cases or features: how do building blocks execute them?
* interactions at critical external interfaces: how do building blocks cooperate with users and neighboring systems?
* operation and administration: launch, start-up, stop
* error and exception scenarios

Remark: The main criterion for the choice of possible scenarios (sequences, workflows) is their **architectural relevance**. It is **not** important to describe a large number of scenarios. You should rather document a representative selection.

**Motivation.**

You should understand how (instances of) building blocks of your system perform their job and communicate at runtime. You will mainly capture scenarios in your documentation to communicate your architecture to stakeholders that are less willing or able to read and understand the static models (building block view, deployment view).

**Form.**

There are many notations for describing scenarios, e.g.

* numbered list of steps (in natural language)
* activity diagrams or flow charts
* sequence diagrams
* BPMN or EPCs (event process chains)
* state machines
* …

## <Runtime Scenario 1>

* *<insert runtime diagram or textual description of the scenario>*
* *<insert description of the notable aspects of the interactions between the building block instances depicted in this diagram.>*

## <Runtime Scenario 2>

## …

## <Runtime Scenario n>

# Deployment View

**Content.**

The deployment view describes:

1. the technical infrastructure used to execute your system, with infrastructure elements like geographical locations, environments, computers, processors, channels and net topologies as well as other infrastructure elements and
2. the mapping of (software) building blocks to that infrastructure elements.

Often systems are executed in different environments, e.g. development environment, test environment, production environment. In such cases you should document all relevant environments.

Especially document the deployment view when your software is executed as distributed system with more then one computer, processor, server or container or when you design and construct your own hardware processors and chips.

From a software perspective it is sufficient to capture those elements of the infrastructure that are needed to show the deployment of your building blocks. Hardware architects can go beyond that and describe the infrastructure to any level of detail they need to capture.

**Motivation.**

Software does not run without hardware. This underlying infrastructure can and will influence your system and/or some cross-cutting concepts. Therefore, you need to know the infrastructure.

Maybe the highest level deployment diagram is already contained in section 3.2. as technical context with your own infrastructure as ONE black box. In this section you will zoom into this black box using additional deployment diagrams:

* UML offers deployment diagrams to express that view. Use it, probably with nested diagrams, when your infrastructure is more complex.
* When your (hardware) stakeholders prefer other kinds of diagrams rather than the deployment diagram, let them use any kind that is able to show nodes and channels of the infrastructure.

## Infrastructure Level 1

Describe (usually in a combination of diagrams, tables, and text):

* the distribution of your system to multiple locations, environments, computers, processors, .. as well as the physical connections between them
* important justification or motivation for this deployment structure
* Quality and/or performance features of the infrastructure
* the mapping of software artifacts to elements of the infrastructure

For multiple environments or alternative deployments please copy that section of arc42 for all relevant environments.

***<Overview Diagram>***

Motivation

*<explanation in text form>*

Quality and/or Performance Features

*<explanation in text form>*

Mapping of Building Blocks to Infrastructure

*<description of the mapping>*

## Infrastructure Level 2

Here you can include the internal structure of (some) infrastructure elements from level 1.

Please copy the structure from level 1 for each selected element.

### *<Infrastructure Element 1>*

*<diagram + explanation>*

### *<Infrastructure Element 2>*

*<diagram + explanation>*

…

### *<Infrastructure Element n>*

*<diagram + explanation>*

# Cross-cutting Concepts

**Content.**

This section describes overall, principal regulations and solution ideas that are relevant in multiple parts (= cross-cutting) of your system. Such concepts are often related to multiple building blocks. They can include many different topics, such as

* domain models
* architecture patterns or design patterns
* rules for using specific technology
* principal, often technical decisions of overall decisions
* implementation rules

**Motivation.**

Concepts form the basis for *conceptual integrity* (consistency, homogeneity) of the architecture. Thus, they are an important contribution to achieve inner qualities of your system.

Some of these concepts cannot be assigned to individual building blocks (e.g. security or safety). This is the place in the template that we provided for a cohesive specification of such concepts.

**Form.**

The form can be varied:

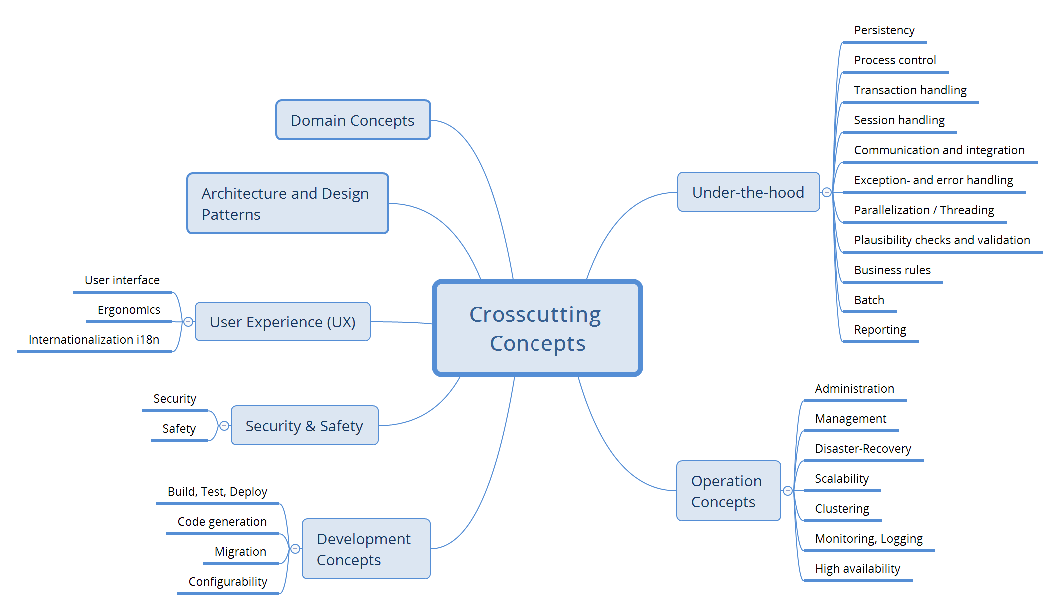
* concept papers with any kind of structure
* cross-cutting model excerpts or scenarios using notations of the architecture views
* sample implementations, especially for technical concepts
* reference to typical usage of standard frameworks (e.g. using Hibernate for object/relational mapping)

**Structure.**

A potential (but not mandatory) structure for this section could be:

* Domain concepts
* User Experience concepts (UX)
* Safety and security concepts
* Architecture and design patterns
* "Under-the-hood"
* development concepts
* operational concepts

Note: it might be difficult to assign individual concepts to one specific topic on this list.



## *<Concept 1>*

*<explanation>*

## *<Concept 2>*

*<explanation>*

…

## *<Concept n>*

*<explanation>*

# Design Decisions

# 

|  |  |  |
| --- | --- | --- |
| **Problem** | Considered Alternatives | Decision |
| Cross-platform development | Flutter, React + Swift | Flutter; avoid need of additional ressources for development |
| Security issues | validation of uploads, encryption, access controls, rate limiting | validation of uploads using AI and upload limitations |
| Insufficient data storage | NoSQL databases, SQL databases | NoSQL databases to handle large-scale data |

# Quality Requirements

**Content.**

This section contains all quality requirements as quality tree with scenarios. The most important ones have already been described in section 1.2. (quality goals)

Here you can also capture quality requirements with lesser priority, which will not create high risks when they are not fully achieved.

**Motivation.**

Since quality requirements will have a lot of influence on architectural decisions you should know for every stakeholder what is really important to them, concrete and measurable.

## Quality Tree

**Content.**

The quality tree (as defined in ATAM – Architecture Tradeoff Analysis Method) with quality/evaluation scenarios as leafs.

**Motivation.**

The tree structure with priorities provides an overview for a sometimes large number of quality requirements.

**Form.**

The quality tree is a high-level overview of the quality goals and requirements:

* tree-like refinement of the term "quality". Use "quality" or "usefulness" as a root
* a mind map with quality categories as main branches

In any case the tree should include links to the scenarios of the following section.

## Quality Scenarios

**Contents.**

Concretization of (sometimes vague or implicit) quality requirements using (quality) scenarios.

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

For architects, two kinds of scenarios are important:

* Usage scenarios (also called application scenarios or use case scenarios) describe the system’s runtime reaction to a certain stimulus. This also includes scenarios that describe the system’s efficiency or performance. Example: The system reacts to a user’s request within one second.
* Change scenarios describe a modification of the system or of its immediate environment. Example: Additional functionality is implemented or requirements for a quality attribute change.

**Motivation.**

Scenarios make quality requirements concrete and allow to more easily measure or decide whether they are fulfilled.

Especially when you want to assess your architecture using methods like ATAM you need to describe your quality goals (from section 1.2) more precisely down to a level of scenarios that can be discussed and evaluated.

**Form.**

Tabular or free form text.

# Risks and Technical Debts

# 

|  |  |
| --- | --- |
| **Risk/Technical Debt** | Description |
| dependency on external services | For both AI integration and use of filters for image editing we use external services – possible bottleneck when the future maintenance is unclear. |
| high costs for CDN servers | High costs & uncertainty for providing fast services all over the world |
| lack of Flutter developers | Difficulty to find Flutter developers that cover \*both\* iOS and Android development |

# Glossary

**Contents.**

The most important domain and technical terms that your stakeholders use when discussing the system.

You can also see the glossary as source for translations if you work in multi-language teams.

**Motivation.**

You should clearly define your terms, so that all stakeholders

* have an identical understanding of these terms
* do not use synonyms and homonyms
* A table with columns <Term> and <Definition>.
* Potentially more columns in case you need translations.

|  |  |
| --- | --- |
| Term | Definition |
| *<Term-1>* | *<definition-1>* |
| *<Term-2>* | *<definition-2>* |