Software Architecture Documentation -

**<Project title>**

Version 01

Date: 9-2-2016

ALTEN

Boris Ivanov

< Confidential, Restricted, Internal>

<Draft, For Review or Released>

*PREPARATION INSTRUCTIONS*

*DO NOT ALTER THIS TEMPLATE. Keep chapter numbering as is. DO NOT INSERT OR REMOVE CHAPTERS. The only exception to this is this page and the appendix pages. They can be removed from the document. If information is to be added, do this by adding it as an appendix and refer to this from within the template. If a chapter is not used tailor it out. If a paragraph is tailored out of this document, the resulting document shall contain the corresponding paragraph number and title, followed by "This paragraph has been tailored out."*

*Multiple paragraphs and subparagraphs, any section, paragraph, or subparagraph in this document may be written as multiple paragraphs or subparagraphs to enhance readability.*

*If a description required by this document has been published in another document, reference to an entry in that document is preferred over including the description itself.*

*SDD template - version history:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Version*** | ***Date*** | ***State*** | ***Author*** | ***Remark*** |
| ***0.1*** | *24-09-2019* | *For review* | *P. Voorwinden, J. Bonnet, M, Hessel, A. Prins* | *- Initial version from arc42 template. Rationale: effective, lean and pragmatic architecture documentation and communication* |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

*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.*

***Remove this page in approved or final versions.***

Version History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | State | Author | Remark |
| 01 |  |  |  |  |

Acronyms and Abbreviations

|  |  |
| --- | --- |
| Term | Explanation |
| <ABC> | <First three letters of the alphabet> |

Referenced documents

*< This table shall list the number, title, revision, and date of all leading documents referenced in this document. >*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Reference | Title | Date | Author |
| 01 |  |  |  |  |

Contents

1 Introduction and goals 6

1.1 Requirements overview 6

1.2 Quality goals 6

1.3 Stakeholders 6

2 Architecture constraints 8

3 System scope and context 9

3.1 Business context 9

3.2 Technical context 9

4 Solution strategy 10

5 Building block view 11

5.1 Whitebox overall system 12

5.1.1 <Name black box 1> 12

5.1.2 <Name black box 2> 13

5.1.3 <Name black box n> 13

5.1.4 <Name interface 1> 13

5.1.5 <Name interface m> 13

5.2 Level 2 13

5.2.1 White box <building block 1> 13

5.2.2 White box *<building block 2>* 13

5.2.3 White box *<building block m>* 13

5.3 Level 3 13

5.3.1 White box <\_building block x.1\_> 13

5.3.2 White box <\_building block x.2\_> 13

5.3.3 White box <\_building block y.1\_> 13

6 Runtime view 14

6.1 <Runtime scenario 1> 14

6.2 <Runtime scenario 2> 14

6.3 <Runtime scenario n> 14

7 Deployment View 15

7.1 Infrastructure level 1 15

7.2 Infrastructure level 2 15

7.2.1 *<Infrastructure Element 1>* 15

7.2.2 *<Infrastructure Element 2>* 15

7.2.3 *<Infrastructure Element n>* 16

8 Cross-cutting concepts 17

8.1 <Concept 1> 18

8.2 <Concept 2> 18

8.3 <Concept n> 18

9 Design decisions 19

10 Quality requirements 20

10.1 Quality Tree 20

10.2 Quality Scenarios 20

11 Risks and Technical Debts 21

# 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

**Contents.**

*Short description of the functional requirements, driving forces, extract (or abstract) of requirements. Link to (hopefully existing) requirements documents (with version number and information where to find it).*

**Motivation.**

*From the point of view of the end users a system is created or modified to improve support of a business activity and/or improve the quality.*

**Form.**

*Short textual description, probably in tabular use-case format. If requirements documents exist this overview should refer to these documents.*

*Keep these excerpts as short as possible. Balance readability of this document with potential redundancy w.r.t to requirements documents.*

## Quality goals

***Contents.***

*The top three (max five) quality goals for the architecture whose fulfillment is of highest importance to the major stakeholders. We really mean quality goals for the architecture. Don’t confuse them with project goals. They are not necessarily identical.*

***Motivation.***

*You should know the quality goals of your most important stakeholders, since they will influence fundamental architectural decisions. Make sure to be very concrete about these qualities, avoid buzzwords. If you as an architect do not know how the quality of your work will be judged …*

***Form.***

*A table with quality goals and concrete scenarios, ordered by priorities*

## Stakeholders

***Contents.***

*Explicit overview of stakeholders of the system, i.e. all person, roles or organizations that*

* *should know the architecture*
* *have to be convinced of the architecture*
* *have to work with the architecture or with code*
* *need the documentation of the architecture for their work*
* *have to come up with decisions about the system or its development*

***Motivation.***

*You should know all parties involved in development of the system or affected by the system. Otherwise, you may get nasty surprises later in the development process. These stakeholders determine the extent and the level of detail of your work and its results.*

***Form.***

*Table with role names, person names, and their expectations with respect to the architecture and its documentation.*

|  |  |  |
| --- | --- | --- |
| Role/Name | Contact | Expectations |
| <Role-1> | <Contact-1> | <Expectation-1> |
| <Role-2> | <Contact-2> | <Expectation-2> |

# Architecture constraints

***Contents.***

*Any requirement that constrains software architects in their freedom of design and implementation decisions or decision about the development process. These constraints sometimes go beyond individual systems and are valid for whole organizations and companies.*

***Motivation.***

*Architects should know exactly where they are free in their design decisions and where they must adhere to constraints. Constraints must always be dealt with; they may be negotiable, though.*

***Form.***

*Simple tables of constraints with explanations. If needed you can subdivide them into technical constraints, organizational and political constraints and conventions (e.g. programming or versioning guidelines, documentation or naming conventions)*

# 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

***Contents.***

*Specification of all communication partners (users, IT-systems, …) with explanations of domain specific inputs and outputs or interfaces. Optionally you can add domain specific formats or communication protocols.*

***Motivation.***

*All stakeholders should understand which data are exchanged with the environment of the system.*

***Form.***

*All kinds of diagrams that show the system as a black box and specify the domain interfaces to communication partners.*

*Alternatively (or additionally) you can use a table. The title of the table is the name of your system, the three columns contain the name of the communication partner, the inputs, and the outputs.*

*<Diagram or Table>*

*<optionally: Explanation of external domain interfaces>*

## 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

***Contents.***

*A short summary and explanation of the fundamental decisions and solution strategies, that shape the system’s architecture. These include*

* *technology decisions*
* *decisions about the top-level decomposition of the system, e.g. usage of an architectural pattern or design pattern*
* *decisions on how to achieve key quality goals*
* *relevant organizational decisions, e.g. selecting a development process or delegating certain tasks to third parties.*

***Motivation.***

*These decisions form the cornerstones for your architecture. They are the basis for many other detailed decisions or implementation rules.*

***Form.***

*Keep the explanation of these key decisions short.*

*Motivate what you have decided and why you decided that way, based upon your problem statement, the quality goals and key constraints. Refer to details in the following sections.*

# 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:*

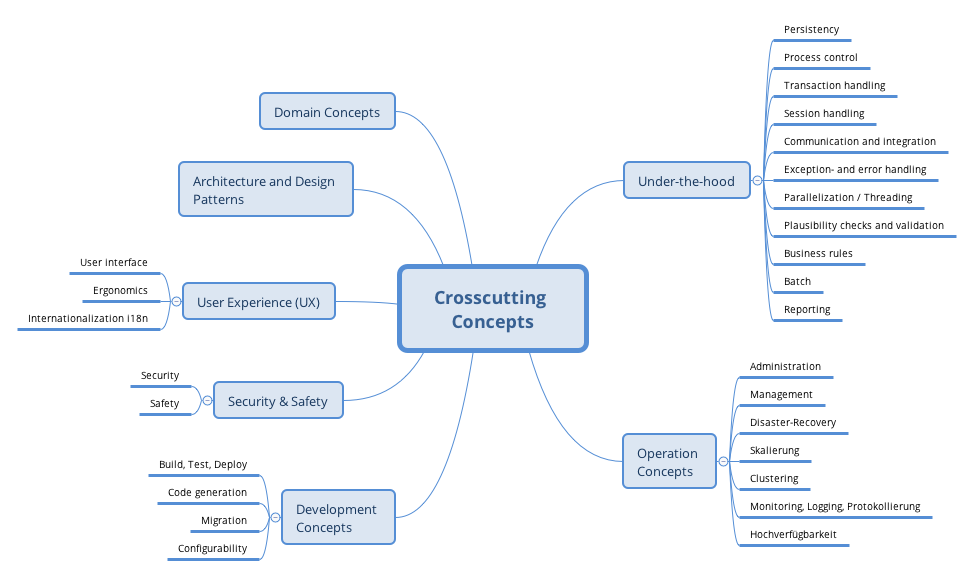
* *concept papers with any kind of structure*
* *cross-cutting model excerpts or scenarios using notations of the architecture views*
* *sample implementations, especially for technical concepts*
* *reference to typical usage of standard frameworks (e.g. using Hibernate for object/relational mapping)*

***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

***Contents.***

*Important, expensive, large scale or risky architecture decisions including rationals. With "decisions" we mean selecting one alternative based on given criteria.*

*Please use your judgement to decide whether an architectural decision should be documented here in this central section or whether you better document it locally (e.g. within the white box template of one building block).*

*Avoid redundancy. Refer to section 4, where you already captured the most important decisions of your architecture.*

***Motivation.***

*Stakeholders of your system should be able to comprehend and retrace your decisions.*

***Form.***

*Various options:*

* *List or table, ordered by importance and consequences or:*
* *more detailed in form of separate sections per decision*
* *ADR (architecture decision record) for every important decision*

# 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

***Contents.***

*A list of identified technical risks or technical debts, ordered by priority*

***Motivation.***

*“Risk management is project management for grown-ups” (Tim Lister, Atlantic Systems Guild.)*

*This should be your motto for systematic detection and evaluation of risks and technical debts in the architecture, which will be needed by management stakeholders (e.g. project managers, product owners) as part of the overall risk analysis and measurement planning.*

***Form.***

*List of risks and/or technical debts, probably including suggested measures to minimize, mitigate or avoid risks or reduce technical debts.*