Architecture Documentation Wiki Documentation

Release 0.0.

Raphael Dürscheid, based on Template by Dr. Peter

Sep 17, 2019

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION AND GOALS

This document documents the software requirements and describes how the SCRUM team plans to implement them. - It should allow the developers to ideally develop the system without having to ask questions or take ad-hoc requirements. - It should ease the onboarding of new team members - It should inform the testers on how to run and test the system

The introduction to the architecture documentation should list the driving forces that software architects must consider in their decisions. This includes on the one hand the fulfillment of functional requirements of the stakeholders, on the other hand the fulfillment of or compliance with required constraints, always in consideration of the architecture goals.

1.1 Relevant Documents

Links to the relevant documents such as

- VA Softwarentwicklung
- Entwicklungsplan / Wartungsplan
- Risikomanagement Akte
-

1.2 Requirements Overview

The **stakeholder requirements** are found in the corresponding Jira **instance.** The user stories link to the relevant system tests.

Todo: Insert link to the stakeholder requirements

The **software requirements** define the system from a blackbox/interfaces perspective. They are split into the following sections:

- **User Interfaces** - *User Interfaces*
- **Technical Interfaces** - *Technical Interface*
- **Runtime Interfaces and Constraints** - *Technical Constraints / Runtime Interface Requirements*

The *Produktbeschreibung* and especially the *Zweckbestimmung* give an overview of the intended use of this system.

Todo: Insert link to the Produktbeschreibung

Insert link to the Zweckbestimmung

1.3 Quality Goals

Contents.

The top three (max five) goals for the architecture and/or constraints whose fulfillment is of highest importance to the major stakeholders. Goals that define the architecture's quality could be:

- availability
- modifiability
- performance
- security
- testability
- usability

Motivation.

If you as an architect do not know how the quality of your work can be judged . . .

Form.

Simple tabular representation, ordered by priorities

Background Information.

NEVER start developing an architecture if these goals have not been put into writing and have not been signed by the major stakeholders.

Sources.

The DIN/ISO 92000 Standard contains an extensive set of possible quality goals.

1.4 Stakeholders

Contents.

A list of the most important persons or organizations that are affected by or can contribute to the architecture.

Motivation.

If you do not know the persons participating in or concerned with the project you may get nasty surprises later in the development process. Should your project manager maintain this list, make sure that all the people influencing the architecture are part of it.

Form.

Simple table with role names, person names, their knowledge as pertaining to architecture, their availability, etc. .Stakeholders

Role/Name	Goal/Boundaries
Expected Participation and Contribution	The name or role of a stakeholder
Why will this stakeholder have an interest in the architecture?	what do you expect as a contribution

CHAPTER

TWO

ARCHITECTURE CONSTRAINTS

Contents.

Any requirement that constrains software architects in their freedom of design decisions or the development process.

Especially the software-requirements for the

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.

Informal lists, structured by the sub-sections of this section.

Examples.

see subsections

Background information.

In the optimal case constraints are defined by requirements. In any case, at least the architects must be aware of constraints.

2.1 Technical Constraints / Runtime Interface Requirements

Contents.

List all technical constraints in this section. This category covers runtime interface requirements and constraints such as:

- Hard- and software infrastructure
- applied technologies - operating systems - middleware - databases - programming languages

Technical Constraints (pot. link to US)	
Hardware Constraints	
C1	insert description here
C2	insert description here
C3	insert description here
Software Constraints	
C4	insert description here
C5	insert description here
C6	insert description here
Operating System Constraints	
C7	insert description here
C8	insert description here
C9	insert description here
Programming Constraints	
C10	insert description here
C11	insert description here
C12	insert description here

Table: List of Technical Constraints

2.2 Conventions

We follow the coding guidelines:

Todo: Input links to coding guidelines

The software development process is defined by the Entwicklungsplan

---

Todo: Input link to Entwicklungsplan

Contents.

List all conventions that are relevant for the development of your software architecture.

Form.

Either insert the conventions directly in this document or refer to other documents.

- Coding guidelines
- Documentation guidelines
- Guidelines for version and configuration management
- Naming conventions

2.3 Organizational Constraints

Contents.

Enter all organizational, structural, and resource-related constraints. This category also covers standards and legal constraints that you must comply with.

Organizational Constraints	
Organization and Structure	
C1	insert description here
C2	insert description here
Resources (Budget, Time, Personnel)	
C3	insert description here
C4	insert description here
Organizationa l Standards	
C5	insert description here
C6	insert description here
Legal Factors	
C7	insert description here
C8	insert description here

Table: List of Organizational Constraints

CHAPTER

THREE

SYSTEM SCOPE AND CONTEXT

Contents.

The context view defines the boundaries of the system under development to distinguish it from neighboring systems. It thereby identifies the system's relevant external interfaces, defining the **software requirements aspects** of **user interfaces** and **technical interfaces**.

Motivation.

The interfaces to neighboring systems are among most critical and risky aspects of a project. Ensure early on that you have understood them in their entirety.

- Various context diagram (see below)
- Lists of neighboring systems and their interfaces.

3.1 Relevant Regulatory Requirements

- List of regulatory requirements
- that are applicable to this system

3.2 Software Safety Class

This software requires the software safety class: **A**

The default is: A

Todo: Input the required safety class

3.3 Context and External Interfaces Overview

Contents.

Identify all neighboring systems and specify all logical business data that is exchanged with the system under development.

List of all (a-l-l) neighboring systems.

Motivation.

Understanding the information exchange with neighboring systems (i.e. all input flows and all output flows).

Form.

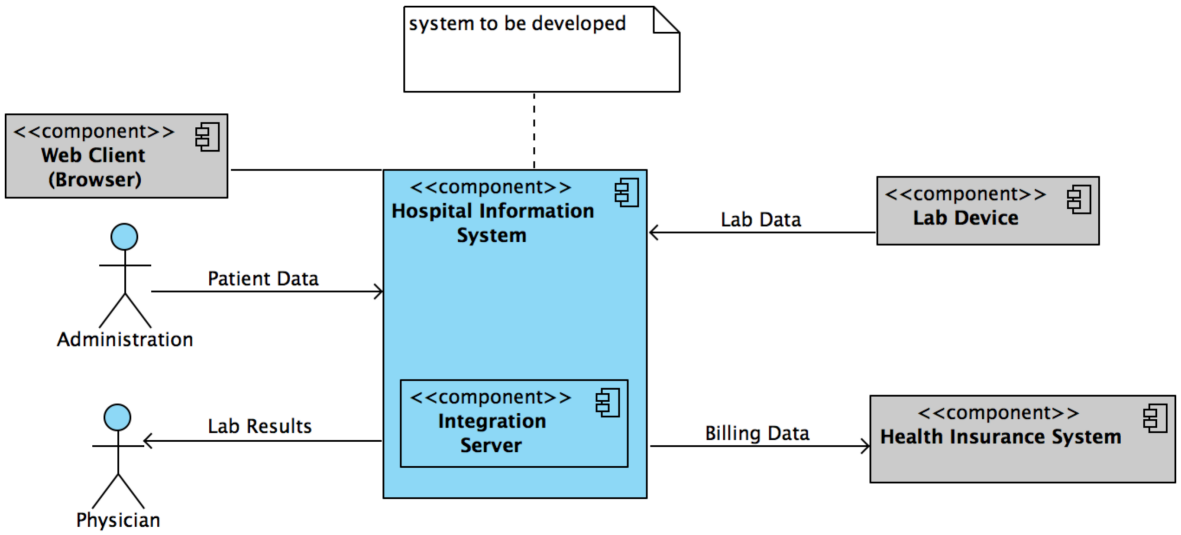

Fig. 1: **UML-type context diagram** - shows the birds eye view of the system (black box) described by this architecture within the ecosystem it is to be placed in. Shows orbit level interfaces on the user interaction and component scope.

3.4 User Interfaces

Contents.

Specification of the **User Interaction Behaviour** (Dynamic and Static) based on stakeholder requirements.

Motivation.

Understanding and communication with the stakeholder regarding the User interaction, especially the GUI.

3.4.1 Dynamic UI Behaviour

3.4.2 Static UI

3.5 Technical Interface

This section describes the data interfaces to other systems around it. It follows the 4 levels of interoperability (**IO**):

- Structural interoperability
- Syntactic interoperability
- Semantic interoperability
- Organisatorial interoperability

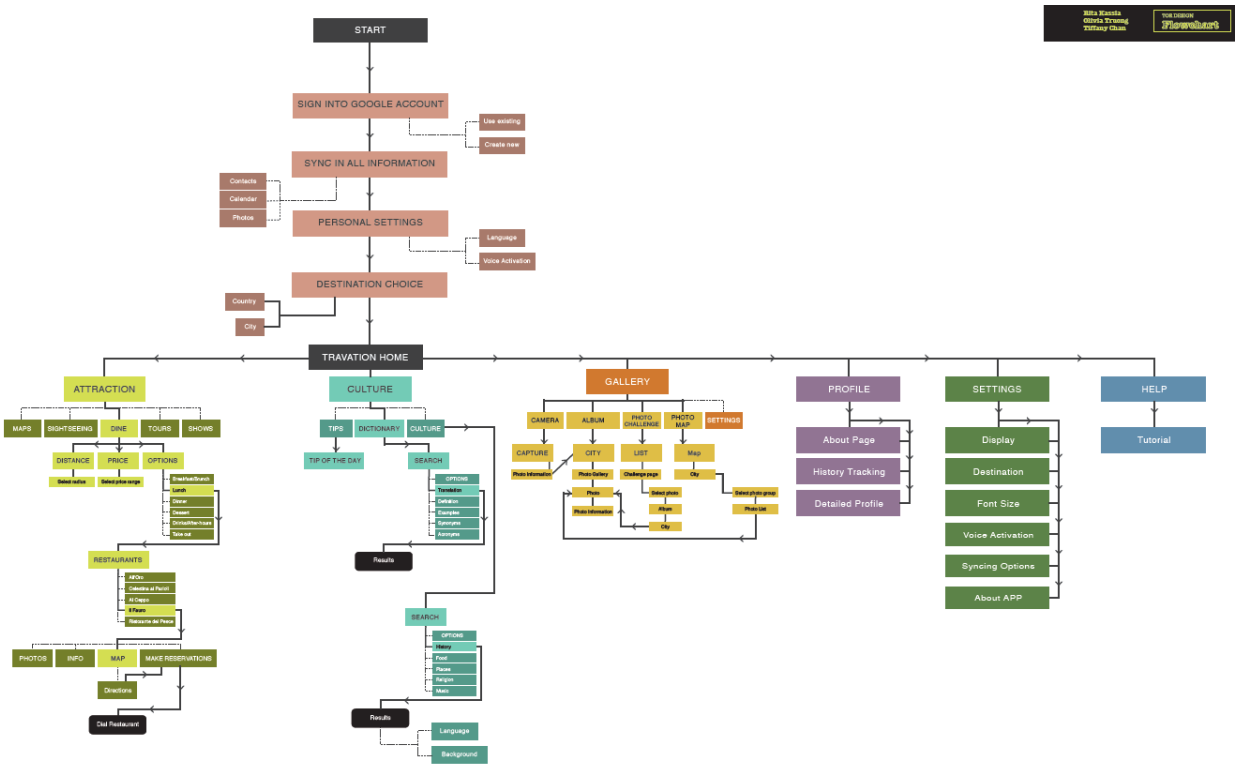

Fig. 2: Diagram showing the dynamic behaviour of the user interface i.e. State diagram

3.5.1 Interface Template

Interface ID	
Name	Name of the Interface
Version	
Structural IO	Datastreams btwn systems. i.e. TCP/IP, RS232, . . .
Syntactic IO	Units within the stream. i.e. XML, CSV, HL7, DICOM
Semantic IO	Common definition of unit meaning.
Organisatorial IO	Workflows, Roleconcepts,. . .
Contact person(s) or organization	

Table: Interface Template

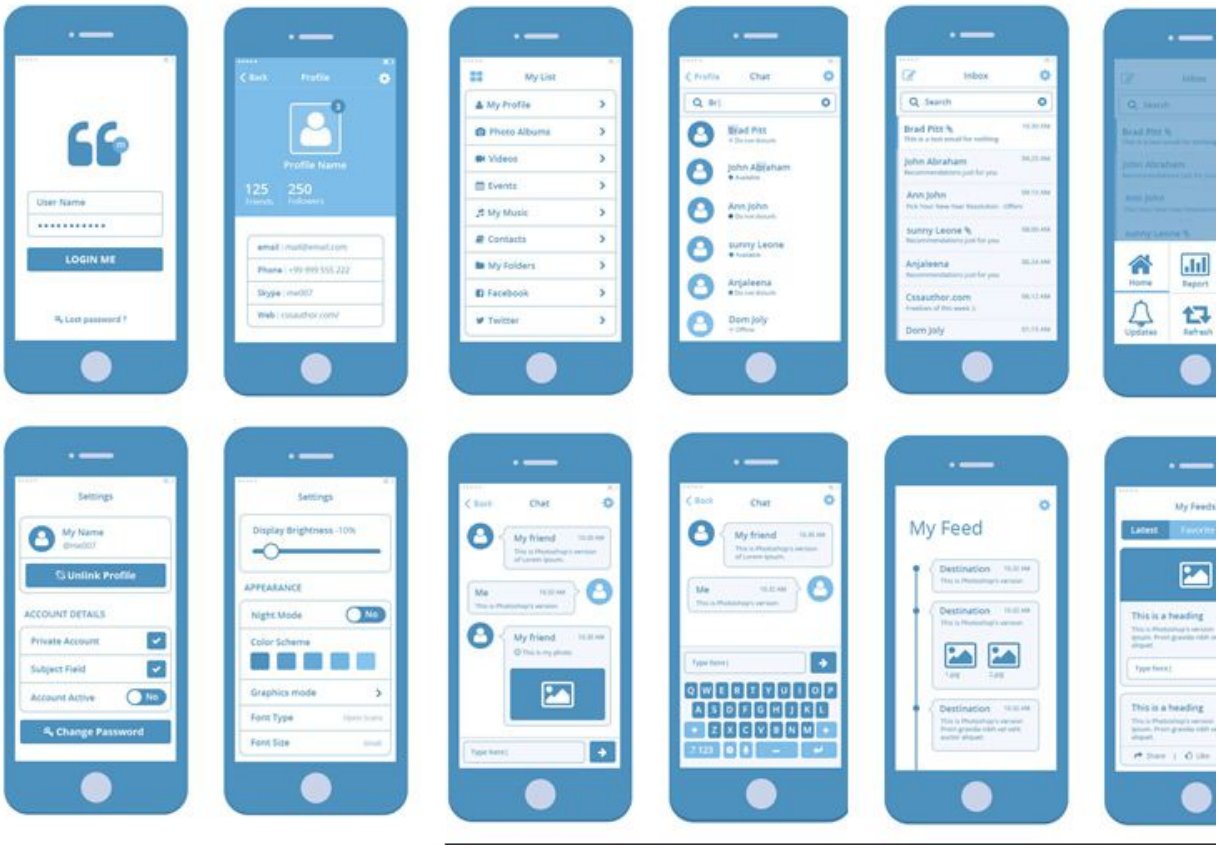

Fig. 3: Mock-up screens of the individual views/screens of the GUI i.e. Wireframes, whiteboard sketches

CHAPTER

FOUR

SOLUTION STRATEGY

Contents.

A short summary and explanation of the fundamental solution ideas and strategies.

Motivation.

An architecture is often based upon some key solution ideas or strategies. These ideas should be familiar to everyone involved into the architecture.

Form.

Diagrams and / or text, as appropriate. Keep it short, i.e. 1 or 2 pages at most!

CHAPTER

FIVE

TEST STRATEGY

Describe

- Where the tests are kept
- How they are invoced
- ...

5.1 System Tests

5.2 Integration Tests

5.3 Unit Tests

CHAPTER

SIX

BUILDING BLOCK VIEW

Contents.

- Static decomposition of the system into building blocks and the relationships thereof.
- Description of SOUPs
- Description of database schemata

We specify the system based on the blackbox view from *Context and External Interfaces Overview* by now considering it a whitebox and identifying the next layer of blackboxes inside it. We re-iterate this zoom-in until specific granularity is reached.

Motivation.

This is the most important view, that must be part of each architecture documentation. In building construction this would be the floor plan.

6.1 SOUPS

Contains a list of the SOUPs used by this system. They are clearly marked and shown in *Building blocks overview*

Name	Author, URL	Version	Description of its task	Safety Class
body row 1, column 1	column 2	column 3	column 4	
body row 2	...	...		

6.2 Database schemata

Description of the database schemata.

6.3 Level 1

6.3.1 Overview

Here you describe the white box view of level 1 according to the white box template. SOUPs are clearly marked.

Todo: <insert overview diagram here>

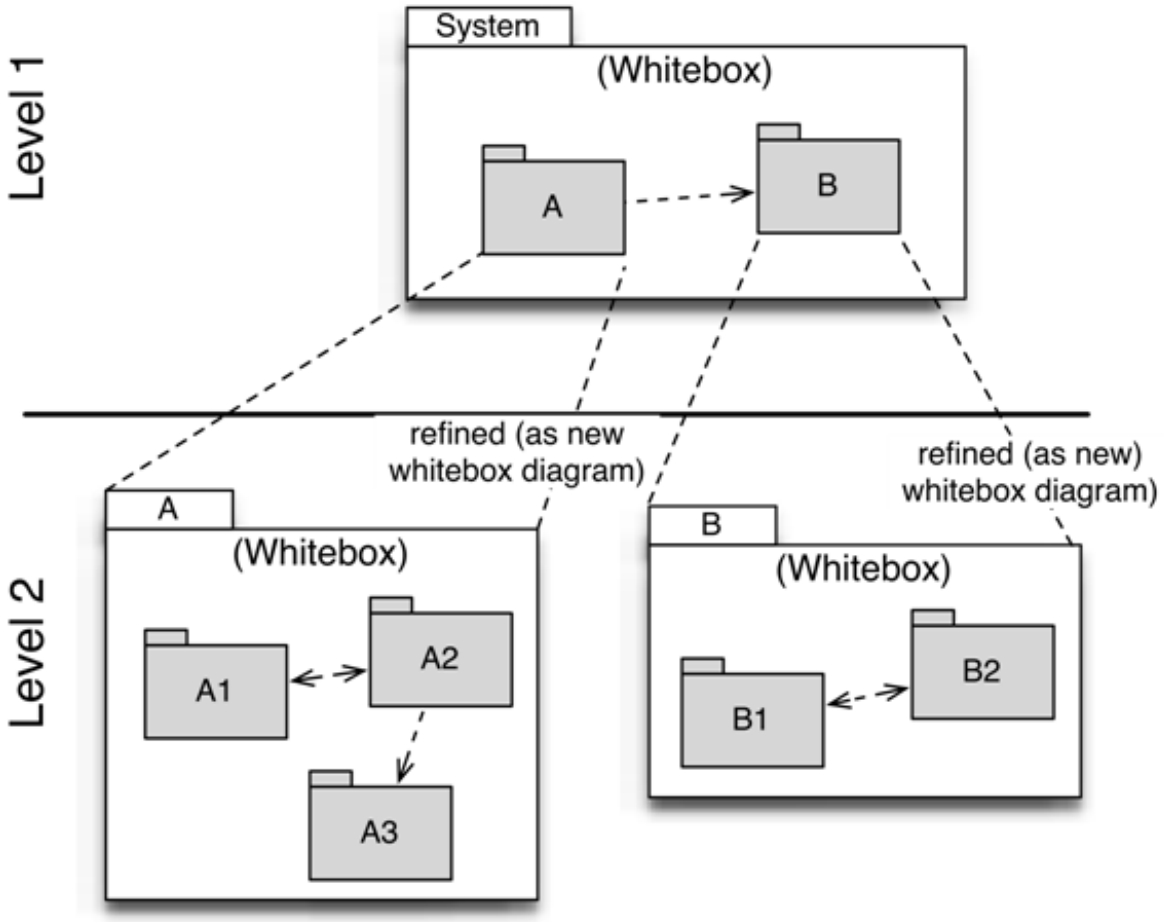

Fig. 1: Building blocks overview

6.3.2 List of components

Todo: Doxygen type list of components

6.3.3 Components

Building Block Name 1 (Black Box Description)

Structure according to black box template:

- Purpose / Responsibility:
- Interface(s):
- Implemented User Stories:
- Variability:
- Performance attributes:
- Repository / Files:
- Other administrative information: Author, Version, Date, Revision History
- Open issues:

Todo: <insert the building block's black box template here>

Building Block Name 2 (Black Box Description)

. . .

6.4 Level 2

Describe all building blocks comprising level 1 as a series of white box templates. The structure is given below for three building blocks and should be duplicated as needed.

6.4.1 Components

Building Block Name 1 (White Box Description)

Shows the inner workings of the building block in form of a diagrams with local building blocks 1 – n, as well as their relationships and interdependencies.

Todo: <insert diagram of building block 1 here>

6.4.2 Building Block Name 1.1 (Black Box Description)

Structure according to black box template:

- Purpose / Responsibility:
- Interface(s):
- Implemented requirements:
- Variability:
- Performance attributes:
- Repository / Files:
- Other administrative information: Author, Version, Date, Revision History
- Open issues:

6.4.3 Building Block Name 1.2 (Black Box Description)

Structure according to black box template

...

6.5 Level 3

Describe all building blocks comprising level 2 as a series of white box templates. The structure is identical to the structure of level 2. Duplicate the corresponding sub-sections as needed. Simply use this section structure for any additional levels you would like to describe.

CHAPTER

SEVEN

RUNTIME VIEW

Contents.

alternative terms:

- Dynamic view
- Process view
- Workflow view

This view describes the behavior and interaction of the system's building blocks as runtime elements (processes, tasks, activities, threads, . . .).

Select interesting runtime scenarios such as:

- How are the most important use cases executed by the architectural building blocks?
- Which instances of architectural building blocks are created at runtime and how are they started, controlled, and stopped.
- How do the system's components co-operate with external and pre-existing components?
- How is the system started (covering e.g. required start scripts, dependencies on external systems, databases, communications systems, etc.)?

  Note

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

Candidates are:

1. The top 3 – 5 use cases
2. System startup
3. The system's behavior on its most important external interfaces
4. The system's behavior in the most important error situations

Motivation.

Especially for object-oriented architectures it is not sufficient to specify the building blocks with their interfaces, but also how instances of building blocks interact during runtime.

Form.

Document the chosen scenarios using UML sequence, activity or communications diagrams. Enumerated lists are sometimes feasible.

Using object diagrams you can depict snapshots of existing runtime objects as well as instantiated relationships. The UML allows to distinguish between active and passive objects.

7.1 Runtime Scenario 1

- Runtime diagram (or other adequate description of scenario!)
- Description of the notable aspects of the interactions between the building block instances depicted in this diagram.

7.2 Runtime Scenario 2

- Runtime diagram (or other adequate description of scenario!)
- Description of the notable aspects of the interactions between the building block instances depicted in this diagram.

7.3 ...

some more

7.4 Runtime Scenario n

- Runtime diagram (or other adequate description of scenario!)
- Description of the notable aspects of the interactions between the building block instances depicted in this diagram.

CHAPTER

EIGHT

DEPLOYMENT VIEW

Contents.

This view describes the environment within which the system is executed. It describes the geographic distribution of the system or the structure of the hardware components that execute the software. It documents workstations, processors, network topologies and channels, as well as other elements of the physical system environment. The deployment view shows the system from the operator's point of view. Please explain how the systems' building blocks are aggregated or packaged into deployment artifacts or deployment units.

Motivation.

Software is not much use without hardware. The minimum that is needed by you as a software architect is sufficient detail of the underlying (hardware) deployment so that you can assign each software building block that is relevant for the system's operations to some hardware element. (This also holds for any COTS that is a prerequisite for the operations of the overall system.) These models should enable the operator to properly install the software.

Form.

The UML provides deployment diagrams for describing this view. Use these – possibly in a nested manner if necessary. (The top level deployment diagram should already be part of your context view, showing your infrastructure as a single black box (cf. chapter 3.2). Here you are zooming into this black box with additional deployment diagrams.) Diagrams by your hardware-oriented colleagues who describe processors and channels are also usable. You should abstract these to aspects relevant for software deployment.

8.1 Infrastructure Level 1

8.1.1 Deployment Diagram Level 1

- Shows the deployment of the overall system to 1 – n processors or sites as well as the physical connections among these elements.
- Lists the most important reasons that led to this deployment structure, i.e. the specific selection of nodes and channels.
- Should also mention rejected alternatives incl. reasons for their rejection.

8.1.2 Processor 1

<insert node template here>

Node Template:

- Description

- Deployed software building blocks
- Quality attributes (performance, constraints, . . .)
- Other administrative information
- Open issues

8.1.3 Processor 2

<insert node template here>

8.1.4 . . .

8.1.5 Processor n

<insert node template here>

8.1.6 Channel 1

Contents.

Specification of the channel's attributes, as relevant for software architecture.

Motivation.

Specify at least those attributes of the communications channels that you need for proving fulfillment of non-functional requirements such as maximal throughput, probability for faults, etc.

Form.

Often you will refer to a standard (e.g. CAN-Bus, 10Mbit Ethernet, IEEE 1394, . . .).

If you need more information use a structure similar to the node template (especially to document quality aspects of channels like throughput, error rates, . . .)

8.1.7 Channel 2

8.1.8 . . .

8.1.9 Channel m

8.2 Infrastructure Level 2

Contents.

Additional deployment diagrams with similar structure as above, refining each node of infrastructure level 1 that needs more details to map the software blocks.

Motivation.

To describe additional details of the infrastructure, as needed by software deployment.

CHAPTER

NINE

CONCEPTS

Contents.

The following chapters cover examples of frequent cross-cutting concerns (a.k.a. aspects in some programming languages) Fill in these chapters if there is **NO** building block that covers this aspect. If some of the concepts are not relevant for your project mention this fact instead of removing the section.

Motivation.

Some concept cannot be "factored" into a separate building block of the architecture (e.g. the topic "security"). This section of the template is the location where you can describe all decision for such a cross cutting topic in one central place. Nevertheless, you have to make sure that all your building blocks conform to such decisions.

Form.

The form can be varied. Some concepts are plain natural language text with a freely chosen structure, some others may include models/scenarios using notations that are also applied in architecture views.

9.1 Domain Models

Contents.

Business (or logical) models, domain models - they all describe subject-specific, business aspects, without relation to technology. Often these structures or concepts are re-used at many places within an architecture, especially within chapter [Building Block View]...

Motivation.

Often parts of these models reappear in the building block view, but in order to „keep them clean from technnology" they can be captured as a concept.

Form.

Usually either ER-Models (Entity-Relationship Models) or UML class diagrams (mainly containing entity classes) are used to document domain models.

9.2 Recurring or Generic Structures and Patterns

Motivation.

Sometimes a hierarchical decomposition of building blocks is insufficient for giving an overview of detailed interdependencies between individual building blocks. The following sections are intended to describe generic or specific dependencies among any set of building blocks – possibly even across different levels. We call a dependency **generic** if it appears more than once in the architecture, and *specific*if it is unique.

Form.

Use building block models (class diagrams, package diagrams, component diagrams, etc.) and related descriptions in the same way as in the hierarchical decomposition. Often it is practical to support understandability by adding specific runtime views to explain these recurring structures.

9.2.1 Recurring or Generic Structure 1

<insert diagram and descriptions here>

9.2.2 Recurring or Generic Structure 2

<insert diagram and descriptions here>

9.3 Persistency

Persistency means moving data from (volatile) memory to a durable storage medium (and back).

Some of the data that a software system is processing must be written to and read from persistent storage media.

- Volatile storage media (main memory or cache) are not designed for permanent storage. Data is lost if the hardware is switched off.
- The amount of data processed by commercial software systems normally exceeds the capacity of main memory.
- Hard disks, optical media and tapes often contain large amounts of existing business data that represent a significant investment.

Persistency is a technical issue that normally does not appear as part of the actual business functionality. An architect must deal with this issue nevertheless because most software systems require efficient access to persistently stored data. This is relevant for essentially all commercial and most technical systems; embedded systems on the other hand often differ in their data management requirements.

9.4 User Interface

Software systems that are used interactively by (human) users require a user interface. These can be graphical, textual, or voice user interfaces.

9.5 Ergonomics

Ergonomics of software systems deals with the improvement (optimization) of their usability with respect to objective and subjective factors. Key ergonomic factors are user interface, reactivity (subjective performance) as well as availability and robustness of the system.

9.6 Flow of Control

Flow of control for software systems is related to visible flows (on the - graphical - user interface) as well as the flow of background activities. Therefore this section should cover control of the user interface as well as control of workflows.

9.7 Transaction Processing

A transaction is a sets of operations or activities that must be processed either in its entirety or not at all. The term is especially relevant in the database area with the important notion of ACID-transactions (atomic, consistent, isolated, durable).

9.8 Session Handling

A session identifies an active connection between a client and a server. The session state must be preserved, which is esp. important if stateless protocols such as HTTP are used for communications. Session handling is a critical challenge esp. for intra- and internet-systems and can strongly influence the performance of a system.

9.9 Security

The security of software systems deals with mechanisms that ensure data confidentiality, integrity, and availability.

Typical issues are:

- How can data be protected during transport (e.g. via open networks such as the internet)?
- How can communicating entities ensure mutual trust?
- How can communicating entities identify each other and be protected against faked identities?
- How can communicating entities prove data provenience or certify validity of data?

The topic of IT-security often touches upon legal aspects, sometimes even international law.

9.10 Safety

The safety of software systems deals with mechanisms that ensure that human life or our environment is not endangered. Describe your concept here: identify those parts of the system that might endanger life and describe mechanism to ensure proper safety.

9.11 Communications and Integration

Communication: Exchange of data between system components. Covers communications within one process or address space, between different processes (inter-process communication – IPC), and between different systems.

Integration: Combination of existing systems in a new context. Also known as: (Legacy) Wrapper, Gateway, Enterprise Application Integration (EAI).

9.12 Distribution

Distribution: Design of software systems whose parts are executed on different – physically separated – hardware systems.

Distribution covers issues such as calling methods on remote systems (remote procedure call – RPC or remote method invocation – RMI), the transfer of data or documents among distributed parties, the choice of optimal modes of interaction or communications patterns (such as synchronous / asynchronous, publish-subscribe, peer-to-peer).

9.13 Plausibility and Validity Checks

How and where do you check plausibility and validity of (input) data, esp. user inputs?

9.14 Exception/Error Handling

How are exceptions and errors handled systematically and consistently?

How can the system reach a consistent state after an error? Is this done automatically or is manual interaction required?

This aspect is also related to logging and tracing,

Which kind of exceptions and errors are handled by the system? Which kinds of errors are forwarded to which external interface and which are handled fully internally?

How do you use the exception handling mechanisms of your programming language? Do you use checked or unchecked exceptions?

9.15 System Management and Administration

Larger software systems are often executed in controlled environments (data centers) under oversight of operators or administrators. These stakeholders require specific information on the applications' states during runtime as well as special means of control and configuration.

9.16 Logging, Tracing

There are two ways of documenting an application's status during runtime: **Logging** and **Tracing**. In both cases the application is extended with function or method calls that write state information, but there is a difference in their usage:

- Logging can cover business or technical aspects or any combination of both.
- Business logs are normally prepared for end users, administrators or operators. They contain information on the business processes that are executed by the application. This kind of logging may also be related to auditing.
- Technical logs contain information for operators or developers. These are used for error detection and system optimization.
- Tracing is intended to provide debugging information for developers or support personnel. It is primarily used for error detection and analysis.

9.17 Business Rules

How do you deal with business logic and business rules? Is business logic implemented in the corresponding business classes or is it handled in a central component? Do you use a rule engine for the interpretation of business rules (production system, forward-/backward-chaining)?

9.18 Configurability

The flexibility of a software system is influenced by its configurability, i.e. the possibility to make certain decisions about usage of the system at a late point in time.

Configurability can occur at the following events:

- During development: For example server, file, or directory names could be stored directly in the code ("hard-coded").
- During deployment or installation: Configuration information for a specific installation (such as the installation path) can be given.
- At system startup: Information can be read dynamically before or during system startup.
- During application execution: Configuration information is queried or read during runtime.

9.19 Parallelization and Threading

Applications can be executed in parallel processes or threads. This creates a need for synchronization points. The theory of parallel processing serves as a foundation for this aspect. The architecture and implementation of parallel systems needs to consider many technical details such as address spaces, applied mechanisms for synchronization – guards, semaphores, etc. – processes and threads, parallelism in the operating system, parallelism in virtual machines. etc.

9.20 Internationalization

This section covers support for usage of the system in different countries, i.e. adjusting the system to country specific attributes. Internationalization (often abbreviated as "i18n" where "18" refers to the eighteen characters between the I and the n) covers translation of text, usage of character encodings, display of fonts, writing of numbers and dates, and other (external) aspects.

9.21 Migration

In many cases a new software system is intended to replace an existing legacy system. As an architect you should not only consider your shiny new architecture but also all organizational and technical aspects that must be considered for the introduction or migration of the architecture.

- Concept, process, or tools for data transfer and initial data creation.
- Concept for system introduction or temporary parallel operations of legacy system and new system.

Is it necessary to migrate existing data? How do you execute any needed syntactic or semantic transformations?

9.22 Testability

Support for simple (and if possible automated) tests. This aspect is the basis for the important implementation pattern of "continuous integration". Projects should support at least daily build-and-test cycles. Important keywords for this aspect are unit tests and mock objects.

9.23 Scaling, Clustering

How can your system grow in a way that can cope with more load or a larger number of users and still keep up performance and throughput.

9.24 High Availability

How can you achieve high availability of your system? Do you use redundancy of major parts? Or do you distribute your system to different processors or locations. Are you running standby- systems?

9.25 Code Generation

How and where do you use code generators to create parts of the system from models or domain specific languages (DSL's)?

9.26 Build-Management

How is the overall system created from is (source code) building blocks? Which repositories contain source code, where are configuration files, test cases, test data and build scripts (make, ant, maven) stored?

CHAPTER

TEN

DESIGN DECISIONS

Contents.

Design decision are documented in the associated Design RFC.

Motivation.

It is advantageous if all important design decisions can be found in one place.

Form.

Informal list, if possible ordered by the decisions' importance for the reader.

10.1 Decision Topic Suggestions

- What exactly is the challenge?
- Why is it relevant for the architecture?
- What consequences does the decision have?
- Which constraints do you have to keep in mind?
- What factors influence the decision?
- Which assumption have you made?
- How can you check those assumptions?
- Which risks are you facing?
- Which alternative options did you consider?
- How do you judge each one?
- Which alternatives are you excluding deliberately?
- Who (if not you) has decided?
- How has the decision been justified?
- When did you decide?

CHAPTER

ELEVEN

QUALITY SCENARIOS

This chapter summarizes all you (or other stakeholders) might need to systematically evaluate the architecture against the quality requirements.

It contains:

- Quality Tree (sometimes called utility tree), an overview of the quality requirements
- Evaluation or quality scenarios - detailed descriptions of the quality requirements or goals.

11.1 Quality Tree

Content.

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

Motivation.

When you want to evaluate the quality (especially risks to certain quality attributes) with methods like ATAM, you need to systematically refine your quality goals (from chapter 1.2). The quality tree shows the top-down refinement of the stakeholder-specific notion of quality.

Form.

We personally prefer mind maps to a pure tree-like structure, as mind maps allow arbitrary cross-references between scenarios, attributes and intermediate nodes. Often it is difficult to assign scenarios to single quality attributes, as the scenario refers to several qualities at once. Simply draw references from such scenarios to all affected nodes!

11.2 Evaluation Scenarios

Contents.

Scenarios describe a system's reaction to a stimulus in a certain situation. They thus characterize the interaction between stakeholders and the system. Scenarios operationalize quality criteria and turn them into measurable quantities.

Two scenarios are relevant for most software architects:

- Usage scenarios (also called application scenarios or use case scenarios) 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.

If you design safety critical systems a third type of scenarios is important for you:

- Boundary or stress scenarios describe how the system reacts to exceptional conditions. Examples: How does the system react to a complete power outage, a serious hardware failure, etc.

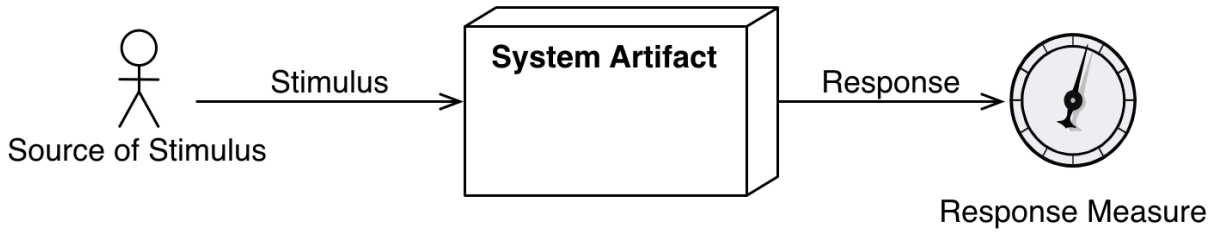

Fig. 1: Schematic depiction of scenarios

Scenarios comprise the following major parts (according to [Starke05], original structure from [Bass+03]):

- Stimulus: Describes a specific interaction between the (stimulating) stakeholder and the system. Example: A user calls a functions, a developer implements an extension, an administrator installs or configures the system.
- Source of the stimulus: Describes where the stimulus comes from. Examples: internal or external, user, operator, attacker, manager.
- Environment: Describes the system's state at the time of arrival of the stimulus. This should list all preconditions that are necessary for comprehension of the scenario. Examples: Is the system under normal or maximal load? Is the data base available or down? Are any users online?
- System artifact: Describes the part of the system is affected by the stimulus. Examples: The whole system, the data base, the web server.
- System response: Describes the system's reaction to the stimulus as determined by the architecture. Examples: Is the function called by the user executed. How long does the developer need for implementation? Which parts of the system are affected by the installation/configuration?
- Response measure: Describes how the response can be measured or evaluated. Examples: Downtime in hours, correctness yes/no, time for code change in person days, reaction time in seconds.

Motivation.

You need scenarios for the evaluation and review of architectures. They take the role of a "benchmark" and aid in measuring the architecture's achievement of its objectives regarding the non-functional requirements and quality attributes.

Form.

Tabular or free text. Explicitly highlight the scenario's elements (source, environment, artifact, response, measure). Alternatively, use similar notations as those suggested in [Runtime View].

Background Information.

There are relations between scenarios and the runtime view. Often you can use scenarios of the runtime view fully or as a basis for evaluation. Evaluation scenarios additionally contain response measures that are often not considered in the pure execution focus of runtime scenarios.

CHAPTER

TWELVE

TECHNICAL RISKS

Contents.

A list of identified technical risks, ordered by priority. The full list of risks can be found in the *Risikomanagementakte*

Todo: Insert link to the Risikomanagementakte

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 technical risks in the architecture, which will be needed by project management as part of the overall risk analysis.

Form.

List of risks with probability of occurrence, amount of damage, options for risk avoidance or risk mitigation, ...

CHAPTER

THIRTEEN

GLOSSARY

Contents.

The most important terms of the software architecture in alphabetic order.

Motivation.

It should not be necessary to explain the usefulness of a glossary ...

Form.

A simple table with columns <Term> and <Definition>

Glossary			
Term	Synonym	Description	
Term	Synonym	Description	

CHAPTER

FOURTEEN

ABOUT ARC42

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

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

Template Revision: 6.5 EN (based on asciidoc), Juni 2014

Note

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.

CHAPTER

FIFTEEN

LITERATURE AND REFERENCES

Starke-2014 Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden. Carl Hanser Verlag, 6, Auflage 2014.

Starke-Hruschka-2011 Gernot Starke und Peter Hruschka: Softwarearchitektur kompakt. Springer Akademischer Verlag, 2. Auflage 2011.

Zörner-2013 Softwarearchitekturen dokumentieren und kommunizieren, Carl Hanser Verlag, 2012

CHAPTER

SIXTEEN

EXAMPLES

- HTML Sanity Checker
- DocChess (german)
- Gradle (german)
- MaMa CRM (german)
- Financial Data Migration (german)

CHAPTER

SEVENTEEN

ACKNOWLEDGEMENTS AND COLLABORATIONS

arc42 originally envisioned by Dr. Peter Hruschka and Dr. Gernot Starke.

Sources We maintain arc42 in *asciidoc* format at the moment, hosted in GitHub under the aim42-Organisation.

Issues We maintain a list of open topics and bugs.

We are looking forward to your corrections and clarifications! Please fork the repository mentioned over this lines and send us a *pull request*!

17.1 Collaborators

We are very thankful and acknowledge the support and help provided by all active and former collaborators, uncountable (anonymous) advisors, bug finders and users of this method.

17.1.1 Currently active

- Gernot Starke
- Stefan Zörner
- Markus Schärtel
- Ralf D. Müller
- Peter Hruschka
- Jürgen Krey

17.1.2 Former collaborators

(in alphabetical order)

- Anne Aloysius
- Matthias Bohlen
- Karl Eilebrecht
- Manfred Ferken
- Phillip Ghadir
- Carsten Klein
- Prof. Arne Koschel

- Axel Scheithauer

BuildingBlockName Whitebox description

Overview / structure *The following illustration shows the inner building blocks constituting _BuildingBlockName* and their interconnection.

<paste Overview-Diagram here>

Reasoning *Explain the reasoning behind this structure*

Enclosed building blocks *(shortly) itemize the building blocks enclosed within _BuildingBlockName* with name and their purpose_

Building block 1	Description 1
Link to Blackbox description	Building block 2
Description 2	Link to Blackbox description
Building block 3	Description 3

Interfaces *itemize the internal and external interfaces of _BuildingBlockName* with name and a (short) description_

Interface 1	Description 1
Interface 2	Description 2
Interface 3	Description 3

BuildingBlockName Blackbox description

Purpose: *describe here Purpose and Responsibilities of this building block*

Interface 1	Description 1
Interface 2	Description 2
Interface 3	Description 3

Table: Interfaces *BuildingBlockName*

Related Locations/Files: *Point to a relevant starting point in source code, or name packages or modules of the most relevant classes to this building block*

BuildingBlockName Blackbox description

Purpose: *describe here Purpose and Responsibilities of this building block*

Interface 1	Description 1
Interface 2	Description 2
Interface 3	Description 3

Table: Interfaces *BuildingBlockName*

Related Locations/Files: *Point to a relevant starting point in source code, or name packages or modules of the most relevant classes to this building block*

Requirement 1	Remark 1
Requirement 2	Remark 2
Requirement 3	Remark 3

Table: Requirements fulfilled

Variability: *describe hier how is this building block variable, flexible or configurable.*

Topic 1	Remark 1
Topic 2	Remark 2
Topic 3	Remark 3

Table: Open discussion topics

Version	Remark 1
Author	Remark 2
Revision date	Remark 3
(. . .)	. . .

Table: Organisational

CHAPTER

EIGHTEEN

INDICES AND TABLES

- genindex
- modindex
- search