Architectural Document

<Contraceptive Timer>

Damir Kusar

*Template Revision:   
5.0 EN, June 2011 (Arc42)  
Modified:  
July 2012 (Damir Kusar)*

**Revision History**

| **Version** | **Date** | **Reviser** | **Description** |
| --- | --- | --- | --- |
| 0.0 | 4. July 2012 | DaKu | Initial Version |
| 0.1 | 4. July 2012 | DaKu | Added tectual descriptions |
|  |  |  |  |

**Related documents**

|  |  |
| --- | --- |
| **Document** | **Description** |
|  |  |
|  |  |

**Table of Contents**

1. Introduction and Goals 5

1.1 Requirements Overview 5

1.2 Quality Goals 5

1.3 Stakeholders 5

2. Architecture Constraints 5

2.1 Technical Constraints 6

2.2 Organizational Constraints 6

2.3 Conventions 6

3. System Scope and Context 6

3.1 Technical- or Infrastructure Context 6

4. Solution Ideas and Strategy 7

5. Building Block View 7

5.1 Level 1 8

5.2 Level 2 9

5.3 Level 3 11

6. Runtime View 11

6.1 Runtime Scenario 1 12

6.2 Runtime Scenario 2 12

6.3 ... 12

6.4 Runtime Scenario n 12

7. Deployment View 12

7.1 Infrastructure Level 1 13

7.2 Infrastructure Level 2 14

8. Recurring or Generic Structures and Patterns 14

8.1 Recurring or Generic Structure 1 14

8.2 Recurring or Generig Structure 2 14

9. Technical Concepts and Architectural Aspects 14

9.1 Persistency 15

9.2 User Interface 15

9.3 Ergonomics 15

9.4 Flow Control 15

9.5 Transaction Procession 15

9.6 Session Handling 15

9.7 Security 16

9.8 Communications and Integration with other Software Systems 16

9.9 Distribution 16

9.10 Exception/Error Handling 16

9.11 System Management and Administration 16

9.12 Logging, Tracing 17

9.13 Business Rules 17

9.14 Configurability 17

9.15 Parallelization and Threading 17

9.16 Internationalization 17

9.17 Migration 17

9.18 Testability 17

9.19 Plausibility and Validity Checks 17

9.20 Code Generation 17

9.21 Build-Management 17

10. Design Decisions 17

11. Quality Scenarios 18

11.1 Quality Tree 18

11.2 Evaluation Scenarios 18

12. Technical Risks 19

13. Glossary 19

Remark: The Microsoft-Word™ variant of this template contains hidden remarks and suggestions. You can toggle display of this text by the appropriate Word-command.

# Introduction and Goals

## Requirements Overview

Diese App ist wichtig, damit Frauen sich nicht mehr darum kümmern müssen, wann Sie das Verhütungsmittel wechseln müssen. Es sollte den Frauen das leben erleichtern.

Der Conctraceptive Timer (später Timer genannt) sollte funktional sein. Das heisst die App sollte so schlank wie möglich sein.

### Functional Requirements

1. Sollte die drei verschiedenen Verhütungsmittel unterstützen
   1. Pille (täglicher Intervall + 1 Woche Pause)
   2. Pflaster (wöchentlicher Intervall + 1 Woche Pause)
   3. Ring (3 Wochen Intervall + 1 Woche Pause)
2. Startzeit der Einnahme muss eingestellt werden können
3. Erinnerungszeit muss gewählt werden können
4. Alarmton und Vibration sollte eingestellt werden können
5. Es sollte eine History der änderungen geben

### Non-Functional Requirements

1. Android
   1. Unterstützte Versionen sollten sein >= 2.2

## Quality Goals

Die Qualität des Produktes sollte hoch sein. Sie muss:

* Testbar sein
* erweiterbar
* einfach benutzbar
* nach absturz der App sollte die App neu gestartet werden und im Hintergrund laufen, damit sicherlich immer der Alarm ertönt
* bei neustart des Gerätes sollte sichergestellt werden, dass die App den Alarm auslöst

## Stakeholders

* Benutzer der App (Werden wohl nur Frauen sein)
* Entwickler (Damir Kusar)
* Tester (Damir Kusar)
* Beta-Tester (Ruzica Grgic)

# Architecture Constraints

All third party functionalities and libraries must be wrapped into Wrapper Classes, so that they are easy exchangeable.

## Technical Constraints

|  |  |
| --- | --- |
| Hardware-Requirements | |
|  | <insert constraint here> |
|  | <insert constraint here> |
| Software-Requirements | |
| DB | Use DB from System |
| Operating System Requirements | |
| Android | Versions from 2.2 and greater must be supported |
| Programming Requirements | |
|  | <insert constraint here> |

## Organizational Constraints

No organizational Contraints.

## Conventions

Use the Java conventions.

See: <http://en.wikipedia.org/wiki/Naming_convention_(programming)#Java>

Examples:

|  |  |
| --- | --- |
| MethodName | lowerCaseStart |
| TestMethodName | methodToTest\_Scenario\_Expectation |
| TestClassName | ClassnameTest |

Use following techniques:

* Test Driven Development (TDD)
* Clean Code Development (CCD)

# System Scope and Context

* //toDo: Kontextdiagramm

## Technical- or Infrastructure Context

* //toDo:

Contents

Specification of the communications channels between your system, its neighboring systems, and the environment.

Motivation

Understanding of the media used for information exchange with neighboring systems, and the environment.

Form

E.g. UML deployment diagram describing channels to neighboring systems

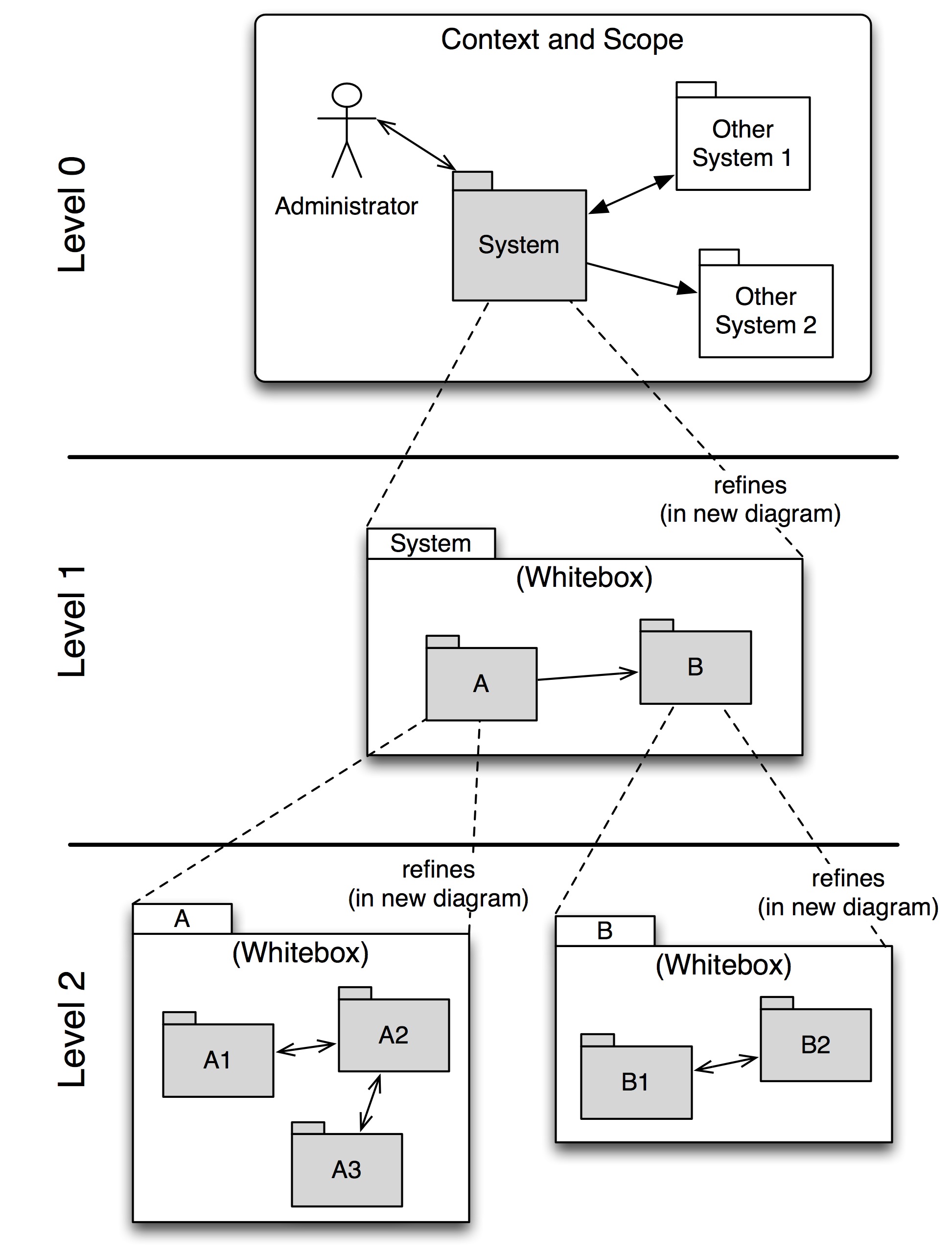
Examples

# Solution Ideas and Strategy

Wrapp all thirdparty libraries.

# Building Block View

* //todo: Bausteinsicht mit levels:



Level 1 contains the white box description of the overall system (system under development / SUD) made up of black box descriptions of the system’s building blocks.

Level 2 zooms into the building blocks of Level 1 and is thus made up of the white box descriptions of all building blocks of Level 1 together with the black box descriptions of the building blocks of Level 2.

Level 3 zooms into the building blocks of Level 2, etc.

The section is structured as follows:

============================

White Box Template:

Contains multiple building blocks with corresponding black box descriptions.

One or more black box templates:

Each building block appearing in the white box template should be described as follows:

1. Purpose / Responsibility:
2. Interface(s):
3. Implemented requirements:
4. Variability:
5. Performance attributes:
6. Repository / Files:
7. Other administrative information: Author, Version, Date, Revision History
8. Open issues:

## Level 1

Here you describe the white box view of level 1 according to the white box template. The structure is given below.

The overview diagram describes the inner structure of the overall system in terms of building blocks 1 – n, as well as their relationships and interdependencies.

It is also useful to list the most important reasons that led to this structure, esp. as relevant to the interdependencies / relationships among the building blocks at this level.

You should also mention rejected alternatives incl. reasons for their rejection.

The following diagram shows the main building blocks of the system and their interdependencies:

<insert overview diagram here>

Comments regarding structure and interdependencies at Level 1:

### Building Block Name 1 (Black Box Description)

Structure according to black box template:

1. Purpose / Responsibility:
2. Interface(s):
3. Implemented requirements:
4. Variability:
5. Performance attributes:
6. Repository / Files:
7. Other administrative information: Author, Version, Date, Revision History
8. Open issues:

<insert the building block’s black box template here>

### Building Block Name 2 (Black Box Description)

<insert the building block’s black box template here>

### ...

<insert the building block’s black box template here>

### Building Block Name n (Black Box Description)

<insert the building block’s black box template here>

### Open Issues

## 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.

### 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.

It is also useful to list the most important reasons that led to this structure, esp. as relevant to the interdependencies / relationships among the building blocks at this level.

You should also mention rejected alternatives incl. reasons for their rejection.

<insert diagram of building block 1 here>

#### Building Block Name 1.1 (Black Box Description)

Structure according to black box template:

1. Purpose / Responsibility:
2. Interface(s):
3. Implemented requirements:
4. Variability:
5. Performance attributes:
6. Repository / Files:
7. Other administrative information: Author, Version, Date, Revision History
8. Open issues:

#### Building Block Name 1.2 (Black Box Description)

Structure according to black box template

#### ...

#### Building Block Name 1.n (Black Box Description)

Structure according to black box template

#### Description of Relationships

#### Open Issues

### Building Block Name 2 (White Box Description)

…

<insert diagram of building block 2 here>

#### Building Block Name 2.1 (Black Box Description)

Structure according to black box template

#### Building Block Name 2.2 (Black Box Description)

Structure according to black box template

#### ...

#### Building Block Name 2.n (Black Box Description)

Structure according to black box template

#### Description of Relationships

#### Open Issues

### Building Block Name 3 (White Box Description)

…

<insert diagram of building block 3 here>

#### Building Block Name 3.1 (Black Box Description)

Structure according to black box template

#### Building Block Name 3.2 (Black Box Description)

Structure according to black box template

#### ...

#### Building Block Name 3.n (Black Box Description)

Structure according to black box template

#### Description of Relationships

#### Open Issues

## 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.

# Runtime View

Contents

alternative terms:

1. Dynamic view
2. Process view
3. 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:

1. How are the most important use cases executed by the architectural building blocks?
2. Which instances of architectural building blocks are created at runtime and how are they started, controlled, and stopped.
3. How do the system’s components co-operate with external and pre-existing components?
4. 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

Esp. 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.

## Runtime Scenario 1

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

## Runtime Scenario 2

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

## ...

## Runtime Scenario n

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

# 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. 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.

## Infrastructure Level 1

### Deployment Diagram Level 1

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

### Processor 1

Structure according to node template:

1. Description
2. Performance attributes
3. Assigned software building blocks
4. Other administrative information
5. Open issues

### Processor 2

Structure according to node template:

### ...

### Processor n

Structure according to node template:

### 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

Use a structure similar to the node template.

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

### Channel 2

### ...

### Channel m

### Open Issues

## Infrastructure Level 2

Contents

Additional deployment diagrams with similar structure as above.

Motivation

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

# Recurring or Generic Structures and Patterns

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 pracital to support understandability by adding specific rruntime views to these recurring structures.

## Recurring or Generic Structure 1

<insert diagram and descriptions here>

## Recurring or Generig Structure 2

<insert diagram and descriptions here>

# Technical Concepts and Architectural Aspects

Contents

The following chapters cover examples of frequent cross-cutting concerns or aspects.

Fill in these chapters if there is NO building block that covers this aspect. If some of the aspects are not relevant for your project mention this fact instead of removing the section.

Motivation

Some aspects 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 cover all concepts for such topics in a central place.

Form

.. can be varied. Some concept articles with free structure, some wide-ranging models/scenarios using notations that are also applied in architecture views.

## Persistency

Use the lite DB on the Mobile device to store all relevant data.

## User Interface

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

## 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.

## Flow Control

Flow control of 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.

## Transaction Procession

A transactions 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).

## 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.

## Security

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

Typical issues are:

1. How can data be protected during transport (e.g. via open networks such as the internet)?
2. How can communicating entities ensure mutual trust?
3. How can communicating entities identify each other and be protected against faked identities?
4. 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.

## Communications and Integration with other Software Systems

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).

## 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).

## 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 kind of errors are forwarded to which external interface and which are handled fully internally?

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

## 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.

## Logging, Tracing

Use log4J to log information

* Tracings for Developer

## Business Rules

## Configurability

## Parallelization and Threading

## Internationalization

Prepare for use with Internationalization. (English / German)

## Migration

## Testability

Develop the app with TDD techniques. If the effort is small, use a CI server.

## Plausibility and Validity Checks

## Code Generation

## Build-Management

Use Ant, the build-in tool for Android in Eclipse.

# Design Decisions

Contents

Document all important design decisions and their reasons!

Motivation

It is advantageous if all important design decisions can be found in one place. It is up to you to decide if a decision should be documented here or rather locally (e.g. in the white box descriptions of building blocks). In any case avoid redundancies.

Form

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

# Quality Scenarios

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

## Quality Tree

Content

The quality tree ( as defined in ATAM) 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 mindmaps to a pure tree-like structure, as mindmaps 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!

## 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:

1. 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.
2. 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:

1. 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.

**Source of**

**the Stimulus**

**System**

**artifact**

**Response measure**

**Stimulus**

**Response**

**Figure: Schematic depiction of scenarios (cf. [Bass+03])**

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

1. 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.
2. Source of the stimulus: Describes where the stimulus comes from. Examples: internal or external, user, operator, attacker, manager.
3. 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?
4. System artifact: Describes the part of the system is affected by the stimulus. Examples: The whole system, the data base, the web server.
5. 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?
6. 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).

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.

# Technical Risks

Alarm: How to use the Alarm functionality of the Android system. And that the Alarm is called when necessary. The app must ensure that the Alarm is called when the app is crashed or after a new start of the device.

# 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>