

3rd
Edition



arc⁴²

by Example

Software Architecture
Documentation
in Practice

Gernot Starke

Hendrik Lösch

Michael Simons

Ralf D. Müller

Stefan Zörner

arc42 by Example

Software Architecture Documentation in Practice

Gernot Starke, Michael Simons, Stefan Zörner, Ralf D. Müller and Hendrik Lösch

This book is for sale at <http://leanpub.com/arc42byexample>

This version was published on 2023-04-10



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2016 - 2023 Gernot Starke, Michael Simons, Stefan Zörner, Ralf D. Müller and Hendrik Lösch

Contents

Preface	1
Acknowledgements	1
Conventions	5
Disclaimer	6
I - Introduction	7
I.1 What is arc42?	8
I.2 Why this Book?	11
I.3 What this Book is Not	11
I.4 Overview of the Examples	12
I.5 Table of arc42 Topics	15
II. HTML Sanity Checking	17
II.1. Introduction and Goals	18
II.2 Constraints	24
II.3 System Scope and Context	25
II.4 Solution Strategy	28
II.5 Building Block View	29
II.6 Runtime View	35
II.7 Deployment view	38
II.8 Cross-cutting Concepts	42
II.9 Architecture Decisions	50
II.10 Quality Requirements	52
II.11 Risks and Technical Debt	53
II.12 Glossary	54
III - Mass Market Customer Relationship Management	55

CONTENTS

III.1 Introduction and Goals	56
III.2 Constraints	70
III.3 System Scope and Context	71
III.4 Solution Strategy	77
III.5 Building Block View	78
III.6 Runtime View	86
III.7 Deployment View	89
III.8 Cross-Cutting Concepts	92
III.9 Architecture Decisions	98
III.10 Quality Requirements	99
III.11 Risks	101
III.12 Glossary	102
IV - biking2	104
IV.1 Introduction and Goals	105
IV.2 Constraints	107
IV.3 System Scope and Context	110
IV.4 Solution Strategy	114
IV.5 Building Block View	115
IV.6 Runtime View	128
IV.7 Deployment View	130
IV.8 Cross-cutting Concepts	132
IV.9 Architecture Decisions	144
IV.10 Quality Requirements	146
IV.11 Risks and Technical Debt	147
IV.12 Glossary	148
V - DokChess	151
V.1 Introduction and Goals	152
V.2 Constraints	155
V.3 System Scope and Context	158
V.4 Solution Strategy	161
V.5 Building Block View	166
V.6 Runtime View	178
V.7 Deployment View	181
V.8 Cross-cutting Concepts	183

CONTENTS

V.9 Architecture Decisions	192
V.10 Quality Requirements	199
V.11 Risks and Technical Debts	202
V.12 Glossary	205
VI - docToolchain	210
VI.1 Introduction and Goals	211
VI.2 Constraints	216
VI.3 System Scope and Context	217
VI.4 Solution Strategy	223
VI.5 Building Block View	226
VI.6 Runtime View	239
VI.7 Deployment View	241
VI.8 Cross-cutting Concepts	243
VI.9 Architecture Decisions	244
VI.10 Quality Requirements	249
VI.11 Risks and Technical Debt	251
VI.12 Glossary	253
VII - Foto Max	254
VII.1 About this document	255
VII.2 Introduction and Goals	258
VII.3 Quality Requirements	260
VII.4 Constraints	264
VII.5 System Scope and Context	266
VII.6 Solution Strategy	271
VII.7 Building Block View	277
VII.8 Runtime View	283
VII.9 Deployment View	285
VII.10 Cross-cutting Concepts	287
VII.11 Architecture Decisions	295
VII.12 Risks & Technical Debt	311
VII.13 Glossary	314
VII.14 Organizational Topics	315
VIII - Mac-OS Menubar Application	321

CONTENTS

The Authors	323
Gernot Starke	323
Hendrik Lösch	325
Michael Simons	326
Stefan Zörner	327
Ralf D. Müller	329
Contacting the Authors	330
Further Reading	331

Preface

Acknowledgements

Gernot

Long ago, on a winters' day in 2004, I sat together with Peter Hruschka, a long-time friend of mine and discussed one¹ of our mutual favorite subjects - structure and concepts of software systems.

We reflected about an issue we both encountered often within our work, independent of client, domain and technology: Developers know their way around implementation technologies, managers theirs' around budgets and risk management. But when forced to communicate (or even document) the *architecture* of systems, they often started inventing their own specific ways of articulating structures, designs, concepts and decisions.

Peter talked about his experience in requirements engineering: He introduced me to a *template* for requirements, a pre-structured *cabinet* (or document) called VOLERE² which contains placeholders for everything that *might be* important for a specific requirements document. When working on requirements, engineers therefore needn't think long before they could *dump* their results in the right place - and others would be able to retrieve it later on... (as long as they knew about VOLERE's structure). This requirements template had been in industry use since several years, there even was a [book available³](#) on its usage and underlying methodology.

“If we only had a similar template for software architecture”, Peter complained, and continued “countless IT projects could save big time and money”... My developer-soul added a silent wish: “If this was great, it could even take the ugliness out of documentation”.

¹Peter and Gernot both share a passion for cooking too, but you probably wouldn't share our sometimes exotic taste

²<http://volere.co.uk>

³<http://www.volere.co.uk/book/mastering-the-requirements-process-getting-requirements-right>

We both looked at each other, and within this second decided to create exactly this: A template for software architecture documentation (and communication), that is highly practical, allows for simple and efficient documentation, is usable for all kinds of stakeholders and could facilitate software architecture documentation. And of course, it had to be open source, completely free for organizations to use.

That's how the arc42 journey started.

Since then, Peter and myself have used arc42 in dozens of different IT systems within various domains. It has found significant acceptance within small, medium and large organizations throughout the world. We wrote more many articles around it, taught it to more than 1000 (!) IT professionals and included it in several of our software architecture-related books.

Thanx Peter for starting this wild ride with me. And, of course, for your lectures on cooking.

Thanx to my customers and clients - I learned an incredible lot by working together with you on your complex, huge, hard, interesting and sometimes stressful problems. Due to all these nondisclosure agreements I signed all my life, I'm not officially allowed to mention you all by name.

Thanx to my wife *Cheffe Uli* and my kids Lynn and Per for allowing dad to (once more) sit on the big red chair and ponder about another book project... You're the best, and I call myself incredibly lucky to have you!

Thanx to my parents, who, back in 1985, when the *computer stuff* was regarded to be something between crime and witchcraft, they encouraged my to buy one (an Apple-2, by the way) and didn't even object when I wanted to study computer science instead of something (by that time) more serious. You're great!

Michael

I've met Gernot and Peter (Hruschka) as instructors on a training at the end of 2015. The training was called "Mastering Software Architectures" and I learned an awful lot from both of them, not only the knowledge they shared but how they both shared it. By the end of the training I could call myself "Certified Professional for Software Architecture", but what I really took home was the wish structure, document and communicate my own projects like Peter and Gernot proposed and that's why the current documentation of my pet project [biking2](#) was created.

Since then I used arc42 based documentations several times: As well as for in-house products and also at projects and consultancy gigs. The best feedback was something along the lines: “Wow, now we’ve got an actual idea about what is going on in this module.” What helped that special project a lot was the fact that we set fully on Asciidoc and the “[Code as documentation and documentation as code](#)”⁴ approach I described in depth in my blog.

So here’s to Gernot and Peter: Thanks for your inspiration and the idea for arc42.

StefanZ

Originally, DokChess is a case study from my German [book](#)⁵ on documenting software architecture. Gernot has encouraged me to write it after I had told him about the chess example on a conference in Munich in 2011. Thank you especially for that, Gernot! (besides many valuable discussions and good times since then in Cologne, Zurich ...)

Thanks to Harm Gnoyke and my daughter Katharina for trying to save my miserable English. All mistakes are my fault, of course.

Ralf D. Müller

Quite a while ago, I discovered the arc42 template as MS Word document. It didn’t take long to see how useful it is and I started to use it in my projects. Soon, I discovered that MS Word wasn’t the best format for me as a developing architect. I started to experiment with various text-based formats like Markdown, AsciiDoc but also with Wikis. The JAX conference was the chance to exchange my ideas with Gernot. He told me that Jürgen Krey already created an AsciiDoc version of the arc42 template. We started to consider this template as the golden master and tried to generate all other formats needed (at this time, mainly MS Word and Confluence) from this template. The [arc42-generator](#)⁶ was born, and a wonderful journey was about to start. The current peak of this journey is [docToolchain](#)⁷ - the evolution of the arc42-generator. Read more about its architecture in this book.

⁴<https://info.michael-simons.eu/2018/12/05/documentation-as-code-code-as-documentation/>

⁵<https://www.swadok.de>

⁶<https://github.com/arc42/arc42-generator>

⁷<https://doctoolchain.github.io/docToolchain/>

On this journey, I have met many people who helped me along this way - impossible to name all of them.

However my biggest “Thanx!” goes out to Gernot who always encouraged me to do my next step and helped me along the way.

Thank you, Peter and Gernot for pushing my architectural skills to the next level through their superb training workshops.

Thanx to Jakub Jabłoński and Peter for their review of the architecture - You gave great feedback!

Last but not least, I have to thank my family for their patience while I spent too much time with my notebook!

Hendrik Lösch

Like some of my fellow authors, I met Gernot at conferences and then got to know him better in a workshop. At that time I had already specialized in restructuring legacy software and since I come from the field of cyber-physical systems, I noticed some peculiarities. Bold as I was, I suggested to Gernot that we could write a book together.

Fast-forward some years, I had created a video training about architecture documentation for LinkedIn Learning. Gernot took up the idea and asked me whether I would want to contribute to this book with the example I used in the video training. Lo and behold this is what happened.

My biggest thanks go to both Gernot and Stefan. The work of both has helped me very often in the past years. It is a great honor for me to be able to publish something together with them that helps other people in their work.

I also want to say a big thank you to Attila Bertok who reviewed my part of the book and was an invaluable help in translating it from German into English.

All of us

Thanx to our thorough reviewers that helped us improve the examples, especially Jerry Preissler, Roland Schimmack and Markus Schmitz.

Conventions

Chapter and Section Numbering:

We use roman chapter numbers (I, II, III etc), so we can have the arabic numbers within chapters in alignment with the arc42 sections...

In the sections within chapters, we add the chapter-prefix only for the top-level sections. That leads to the following structure:

Chapter II: HtmlSC

II.1 Introduction and Goals

II.2 Constraints

II.3 Context

...

Chapter III: Mass Market CRM

III.1 Introduction and Goals

III.2 Constraints

III.3 Context

...

Explanations:

The first example ([HTML Sanity Checking](#)) contains short explanations on the arc42 sections, formatted like this one.

In this book, we keep these explanations to a bare minimum, as there are other books extensively covering [arc42 background and foundations](#).

Disclaimer

We like to add a few words of caution before we dive into arc42 examples:



We show you quite pragmatic approaches to software and system architecture documentation, based upon the (also pragmatic) arc42 template. Although fine for many kinds of systems, **this pragmatism is not appropriate** for critical or high-risk systems, developed or operated under strict safety-, security- or similar requirements.

If you're working on safety-critical or otherwise potentially dangerous systems, the methods or techniques demonstrated in this book might **not be appropriate** for you.

We, the authors, cannot take any responsibility if you decide to use arc42 or any other approach shown in this book.

The content of this book has been created with care and to the best of our knowledge. However, we cannot assume any liability for the up-to-dateness, completeness, accuracy or suitability to specific situations of any of the pages.

I - Introduction



This chapter explains the following topics

- I.1 [What is arc42?](#)
- I.2 [Why this book?](#)
- I.3 [What this book is *not*!](#)
- I.4 [Overview of the examples](#)
- I.5 [Table of arc42 sections](#)



This book contains several examples of *Software Architecture Documentation* based upon the practical, economical, well-established and systematic arc42 approach.

It shows how you can *communicate* about software architectures, but it does *not* show how to develop or implement systems!

I.1 What is arc42?

arc42 is a template for architecture documentation.

It answers the following two questions in a pragmatic way, but can be tailored to your specific needs:

- *What* should we document/communicate about our architecture?
- *How* should we document/communicate?

Figure I.1 gives you the big picture: It shows a (slightly simplified) overview of the structure of arc42.

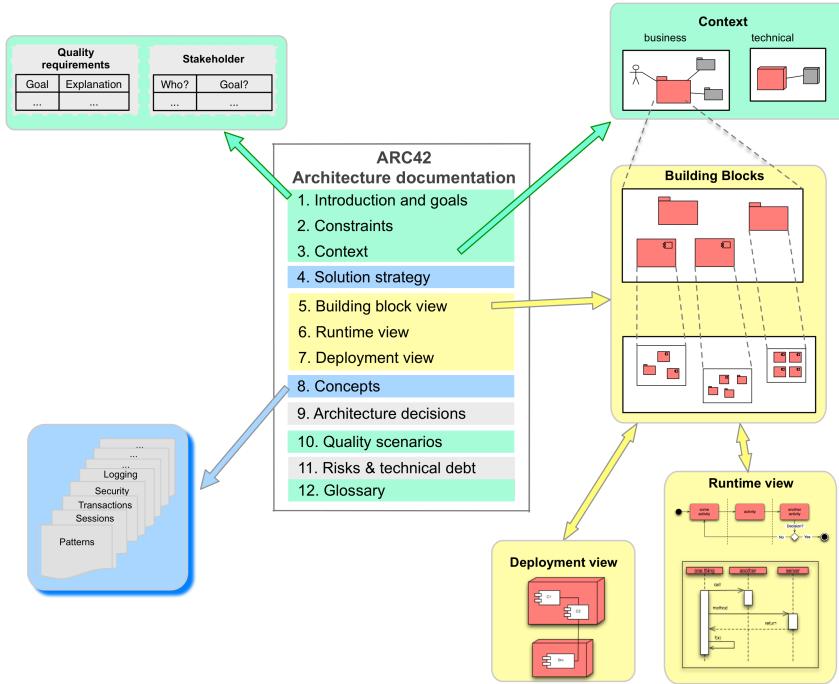
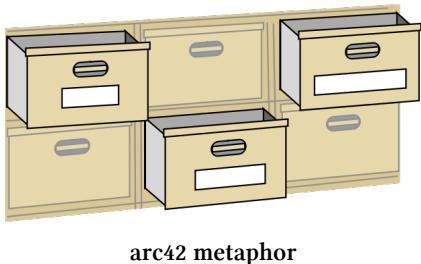


Figure I.1

Compare arc42 to a cabinet with drawers: the drawers are clearly marked with labels indicating the content of each drawer. arc42 contains 12 such drawers (a few more than you see in the picture above). The meaning of these arc42 drawers is easy to understand.



Therefore, arc42 offers you a simple and clear structure to document and communicate your (complex!) system. Starting with the goals and requirements for your system and its embedding into its environment you can provide the important stakeholder of your system with adequate information about the architecture.

arc42 is optimized for understandability and adequacy. It naturally guides you to explain any kind of architecture information or decision in an understandable and reproducible context.

Individuals and organizations using arc42 especially like two things about it:

1. the *understandability* of the documentation resulting from its standardized structure (the *drawers*) and
2. the *manageable effort* to create such documentation. We call it “*painless documentation*”.

Why 42?

You’re kidding, aren’t you? Ever heard of Douglas Adams, the (very British) and already deceased sci-fi writer... his novel “Hitchhikers Guide to The Galaxy” calls 42 the:

Answer to the Ultimate Question of Life, The Universe, and Everything⁸

arc42 aims at providing the answer to *everything* around your software architecture. (Yes, I know it’s a little pretentious, but we couldn’t think of a better name back in 2005.)

⁸https://en.wikipedia.org/wiki/Phrases_from_The_Hitchhiker's_Guide_to_the_Galaxy#Answer_to_the_Ultimate_Question_of_Life.2C_the_Universe.2C_and_Everything_.2842.29

Where to get additional information

With the latest release (V7) of arc42 in January 2017, the project now has extensive online help and FAQ (frequently asked questions) available:

- The [documentation site docs.arc42.org⁹](https://docs.arc42.org) contains the full documentation plus many (more than 100) tips for efficient and effective usage.
- The [FAQ site faq.arc42.org¹⁰](https://faq.arc42.org) answers more than 100 typical questions.

⁹<https://docs.arc42.org>

¹⁰<https://faq.arc42.org>

I.2 Why this Book?

Examples are often better suited to show *how things can work* than lengthy explanations.

arc42 users have often asked for examples to complement the (quite extensive) conceptual documentation of the template that was, unfortunately, only available in German for several years.

There were a few approaches to illustrate how arc42 can be used in real-world applications, but those were (and still are) scattered around numerous sources, and not carefully curated.

After an incredibly successful (again, German only) experiment to publish one single example as a (very skinny) 40-page booklet we decided to publish a *collection of examples* on a modern publishing platform - so we can quickly react to user feedback and add further samples without any hassle.

I.3 What this Book is Not

In this book we focus on examples for arc42 to document and communicate software architectures. We only give a brief introduction to arc42 (in case you like more, see [appendix B](#) for details)

This book is *no* introduction to:

- Software architecture
- Architecture and Design Patterns
- Modeling, especially UML
- Chinese cooking (but probably you didn't expect that here...)

I.4 Overview of the Examples

We encountered exciting software and system architectures during our professional lives. Several of those would have made instructive, interesting and highly relevant examples - too bad they were declared *confidential* by their respective owners. Our *nondisclosure agreements* stated that we're not allowed to even mention clients' names without violating these contracts...

Therefore it has been quite difficult to find practical examples which we can freely write and talk about.

Now let's get an overview of the examples that are waiting in the rest of this book...

HTML Sanity Checking

A tiny open source system to check HTML files for semantic errors, like broken cross references, missing images, unused images and similar stuff.

[Gernot](#) wrote the initial version of this system himself when working on a rather large (100+pages) documentation based upon the [AsciiDoctor](#)¹¹ markup language.

HtmlSanityCheck (or HtmlSC for short) is described in [chapter II](#) of this book.

Mass Market Customer Relationship Management

Gosh, what an awkward name - so let's shorten it to MaMa-CRM at first: *Mass market* refers to consumers of several kinds of enterprises, like health insurance, mobile telecommunication or large real-estate companies.

We are reasonably sure you know these 86×54mm plastic cards (experts call them [ISO/IEC 7810](#)¹²). Chances are you even *own* several of these cards. In case it contains your portrait photo, chances are high it was produced by a system like MaMa-CRM...

MaMa-CRM takes the burden of (usually paper-based) customer contacts from the organizations working in such mass markets. It has been initially build by an

¹¹<https://asciidoc.org>

¹²https://en.wikipedia.org/wiki/ISO/IEC_7810

independent mid-sized data center to support the launch of the German (government-enforced) e-Health-Card - and later on used to support campaigns like telephone billing, electrical-power metering and similar stuff.

The architecture of MaMa-CRM is covered in [chapter III](#).

biking2

biking2 or “*Michis milage*” is a web based application for tracking biking related activities. It allows the user to track the covered milage, collect GPS tracks of routes, convert them to different formats, track the location of the user and publish pictures of bike tours.

The architecture of biking2 is covered in [chapter IV](#)

DokChess - a Chess Engine

DokChess is a fully functional chess engine.

The architecture of DokChess is covered in [chapter V](#)

docToolchain

docToolchain¹³ is a heavily used and highly practical implementation of the [docs-as-code](#)¹⁴ approach: The basic idea is to facilitate creation and maintenance of technical documentation.

The architecture of docToolchain is covered in [chapter VI](#)

Foto Max

Foto Max is a made-up company that offers solutions for ordering photos. The software architecture in this book describes how the software is implemented to be used on the company’s devices.

Foto Max can be found in [chapter VII](#)

¹³<https://doctoolchain.github.io/doctoolchain>

¹⁴<https://docs-as-co.de>

MiniMenu

Surely one of the *leanest* architecture documentations you will ever encounter.
Captures some design and implementation decisions for a tiny Mac-OS menu bar application.

The main goal is to show that arc42 even works for *very* small systems.

Skip to [chapter VIII](#) to see for yourself.

1.5 Table of arc42 Topics

This page helps you to quickly find the arc42 topics you're looking for: For each section of arc42, you find references to each of this books' examples... The digits after the links denote the size of section in the respective example. We marked the most extensive or complete example in **bold**, so you can easily recognize it.

arc42-Section	Where to Find
1. Introduction and Goals	HtmlSC (5) , MaMa-CRM (12) , biking2 (2) , DokChess (2) , docToolchain (5)
2. Constraints	HtmlSC (1) , MaMa-CRM (1) , biking2 (3) , DokChess (2) , docToolchain (1)
3. System Scope and Context	HtmlSC (3) , MaMa-CRM (4) , biking2 (3) , DokChess (2) , docToolchain (4)
4. Solution Strategy	HtmlSC (1) , MaMa-CRM (1) , biking2 (1) , DokChess (3) , docToolchain (3)
5. Building Block View	HtmlSC (6) , MaMa-CRM , biking2 (12) , DokChess , docToolchain (2)
6. Runtime View	HtmlSC (2) , MaMa-CRM (3) , biking2 (2) , DokChess (2) , docToolchain (0)
7. Deployment View	HtmlSC (3) , MaMa-CRM (2) , biking2 (1) , DokChess (2) , docToolchain (4)
8. Cross-cutting Concepts	HtmlSC (6) , MaMa-CRM (5) , biking2 (9) , DokChess (7) , docToolchain (1)
9. Architecture Decisions	HtmlSC (1) , MaMa-CRM (1) , biking2 (2) , DokChess (5) , docToolchain (3)
10. Quality Requirements	HtmlSC (1) , MaMa-CRM (2) , biking2 (1) , DokChess (2) , docToolchain (2)
11. Risks and Technical Debt	HtmlSC (1) , MaMa-CRM (1) , biking2 (1) , DokChess (2) , docToolchain (2)

arc42-Section**Where to Find**

12. Glossary

[HtmlSC \(1\)](#), [MaMa-CRM \(1\)](#), [**biking2 \(4\)**](#), [DokChess \(3\)](#),
[docToolchain \(1\)](#)

II. HTML Sanity Checking

By [Gernot Starke](#).

Convention for this example

At the beginning of each section you find short explanations, formatted in boxes like this.



The system documented here is a small open source tool hosted on [Github](#)¹⁵.

The full sourcecode is available - you might even configure your Gradle build to use this software. Just in case you're writing documentation based on Asciidoc, that would be a great idea!

But enough preamble. Let's get started...

¹⁵<https://github.com/aim42/htmlSanityCheck>

II.1. Introduction and Goals

Content and Motivation

This section shows the driving forces for architecturally relevant decisions and important use-cases or features, summarized in a few sentences. If possible, refer to existing requirements documentation.

The main goal of this section is enabling your stakeholders to understand the solution, which is detailed in arc42-sections 3 to 12.

HtmlSC supports authors creating digital formats by checking hyperlinks, images and similar resources.

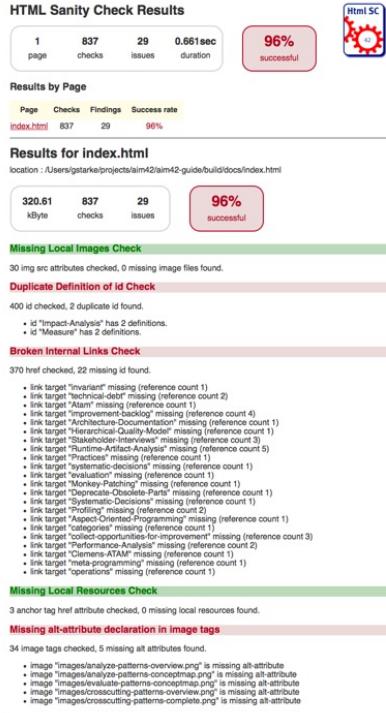
1.1 Requirements Overview

Content and Motivation

You like to briefly explain important goals and requirements, use-cases or features of the system. If available, refer to existing requirements documentation.

Most important: Readers can understand the central tasks of the system, before they encounter the architecture of the system (starting with arc42-section 3).

The overall goal of HtmlSC is to create neat and clear reports, showing errors within HTML files. Below you find a sample report.



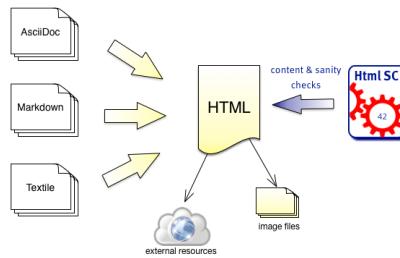
Sample Report

HtmlSanityCheck (HtmlSC) checks HTML for semantic errors, like broken links and missing images. It has been created to support authors who create HTML as output format.

1. Authors write in formats like [AsciiDoc](https://asciidoc.org/docs/what-is-asciidoc/)¹⁶, [Markdown](https://www.daringfireball.net/projects/markdown/syntax)¹⁷ or other formats, which are transformed to HTML by the corresponding generators.
2. HtmlSC checks the generated HTML for broken links, missing images and other semantic issues.
3. HtmlSC creates a test report, similar to the well-known unit test report.

¹⁶<https://asciidoc.org/docs/what-is-asciidoc/>

¹⁷<https://www.daringfireball.net/projects/markdown/syntax>



HtmlSC goal: Semantic checking of HTML pages

Basic Usage

1. A user configures the location (directory and filename) of one or several HTML file(s), and the corresponding images directory.
2. HtmlSC performs various checks on the HTML and
3. reports its results either on the console or as HTML report.

HtmlSC can run from the command line or as Gradle plugin.

Basic Requirements

ID	Requirement	Explanation
G-1	Check HTML for semantic errors	HtmlSC checks HTML files for semantic errors, like broken links.
G-2	Gradle and Maven Plugin	HtmlSC can be run/used as Gradle and Maven plugin.
G-3	Multiple input files	Configurable for a set of files, processed in a single <i>run</i> , HtmlSC produces a joint report.
G-4	Suggestions	When HtmlSC detects errors, it shall identify suggestions or alternatives that might <i>repair</i> the error.
G-5	Configurable	Several features of checks shall be configurable, especially input files/location, output directory, timeouts and status-code behavior for checking external links etc.

Required Checks

HtmlSC shall provide the following checks in HTML files:

Check	Explanation
Missing images	Check all image tags if the referenced image files exist.
Broken internal links	Check all internal links from anchor-tags ('href="#"XYZ') if the link targets "XYZ" are defined.
Missing local resources	Check if referenced files (e.g. css, js, pdf) are missing.
Duplicate link targets	Check all link targets (... id="XYZ") if the id's ("XYZ")are unique.
Malformed links	Check all links for syntactical correctness.
Illegal link targets	Check for malformed or illegal anchors (link targets).

Check	Explanation
Broken external links	Check external links for both syntax and availability.
Broken ImageMaps	Though ImageMaps are a rarely used HTML construct, HtmlSC shall identify the most common errors in their usage.

1.2 Quality Goals

Content and Motivation

You want to understand the quality goals (aka architecture goals), so you can align architecture and design decisions with these goals.

These (usually *long term*) quality goals diverge from the (usually *short term*) goals of development projects. Mind the difference! See also arc42-section 10.

Priority	Quality Goal	Scenario
1	Correctness	Every broken internal link (cross reference) is found.
1	Correctness	Every potential semantic error is found and reported. In case of doubt ¹⁸ , report and let the user decide.
1	Safety	Content of the files to be checked is <i>never</i> altered.
2	Flexibility	Multiple checking algorithms, report formats and clients. At least Gradle and command-line have to be supported.
2	Correctness	Correctness of every checker is automatically tested for positive AND negative cases.
3	Performance	Check of 100kB html file performed under 10 secs (excluding Gradle startup)

¹⁸Especially when checking external links, the correctness of links depends on external factors, like network availability, latency or server configuration, where HtmlSC cannot always identify the root cause of potential problems.

1.3 Stakeholders

Content and Motivation

You want an overview of persons, roles or organizations that affect, are affected by or can contribute to the system and its architecture. Make the concrete expectations of these stakeholders with respect to the architecture and its documentation explicit. Collect these in a simple table.

Remark: For our simple HtmlSC example we have an extremely limited number of stakeholders, in real-life you will most likely have many more stakeholders!

Role	Description	Goal, Intention
Documentation author	writes documentation with HTML output	wants to check that the resulting document contains good links, image references.
arc42 user	uses arc42 for architecture documentation	wants a small but practical example of <i>how to apply arc42</i> .
software developer		wants an example of pragmatic architecture documentation

II.2 Constraints

Content and Motivation

You want to know the constraints that restrict your freedom of design decisions or the development process. Such constraints are often imposed by organizations across several IT systems.

HtmlSC shall be:

- platform-independent and should run on the major operating systems (Windows(TM), Linux, and Mac-OS(TM))
- implemented in Java or Groovy
- integrated with the Gradle build tool
- runnable from the command line
- have minimal runtime and installation dependencies (a Java(TM) runtime may be required to run HtmlSC)
- developed under a liberal open-source license. In addition, all required dependencies/libraries shall be compatible with a CreativeCommons license. |

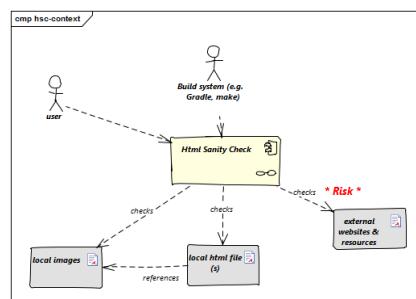
II.3 System Scope and Context

Content and Motivation

You want to know the boundaries and scope of the system to distinguish it from neighboring systems. The context identifies the systems relevant external interfaces.

3.1 Business Context

You want to identify all neighboring systems and the different kinds of (business) data or events that are exchanged between your system and its neighbors.



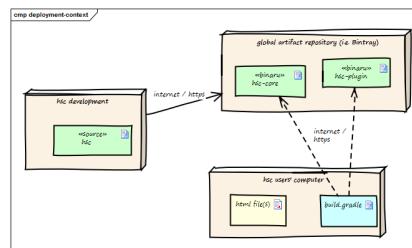
Neighbor	Description
user	documents software with toolchain that generates html. Wants to ensure that links within this HTML are valid.
build system	mostly Gradle ¹⁹
local HTML files	HtmlSC reads and parses local HTML files and performs sanity checks within those.
local image files	HtmlSC checks if linked images exist as (local) files.
external web resources	HtmlSC can be configured to optionally check for the existence of external web resources. Risk: Due to the nature of web systems and the involved remote network operations, this check might need significant time and might yield invalid results due to network and latency issues.

¹⁹<https://gradle.org>

3.2 Deployment Context

You like to know about the technical or physical infrastructure of your system, together with physical channels or protocols.

The following diagram shows the participating computers (nodes) with their technical connections plus the major artifacts of HtmlSC, the hsc-plugin-binary.



Deployment context

Node / Artifact	Description
hsc-development	where development of HtmlSC takes place
hsc-plugin-binary	compiled and packaged version of HtmlSC including required dependencies.
artifact repository	A global public <i>cloud</i> repository for binary artifacts, similar to MavenCentral²⁰ , the Gradle Plugin Portal²¹ or similar. HtmlSC binaries are uploaded to this server.
hsc user computer	where arbitrary documentation takes place with html as output formats.
build.gradle	Gradle build script configuring (among other things) the HtmlSC plugin to perform the HTML checking.

For details see the [deployment-view](#).

²⁰<https://search.maven.org/>

²¹<https://plugins.gradle.com>

II.4 Solution Strategy

Content and Motivation

You need a brief summary and explanation of the fundamental solution ideas and strategies. These key ideas should be familiar to everyone involved in development and architecture.

Briefly explain how you achieve the most important quality requirements.

1. Implement HtmlSC mostly in the Groovy programming language and partially in Java with minimal external dependencies.
2. We wrap this implementation into a Gradle plugin, so it can be used within automated builds. Details are given in the [Gradle userguide²²](#). (The Maven plugin is still under development).
3. Apply the [*template-method-pattern*²³](#) to enable:
 - multiple checking algorithms. See the [concept for checking algorithms](#),
 - both HTML (file) and text (console) output. See the [reporting-concept](#).
4. Rely on standard Gradle and Groovy conventions for configuration, having a single configuration file.
 - For the Maven plugin, this might lead to problems.

²²<https://docs.gradle.org/current/userguide/userguide.html>

²³https://sourcemaking.com/design_patterns/template_method/

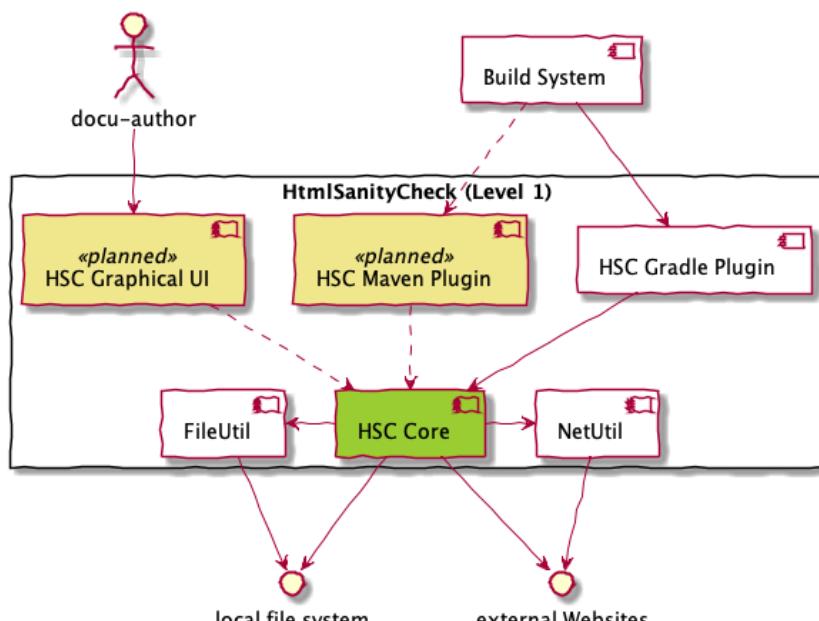
II.5 Building Block View

Content and Motivation

The building block view explains the static decomposition of the system into building blocks (modules, components, subsystems, packages...) and their relationships. It shows the overall structure of the source code.

This view is organized in a top-down hierarchy.

5.1 HtmlSanityChecker (Whitebox)



HtmlSanityCheck
<https://github.com/aim42/htmlSanityCheck>

Whitebox (HtmlSC)

Rationale: We used *functional decomposition* to separate responsibilities:

- HSC Core shall encapsulate checking logic and HTML parsing/processing.
- Plugins and GraphicalUI encapsulate all *usage* aspects

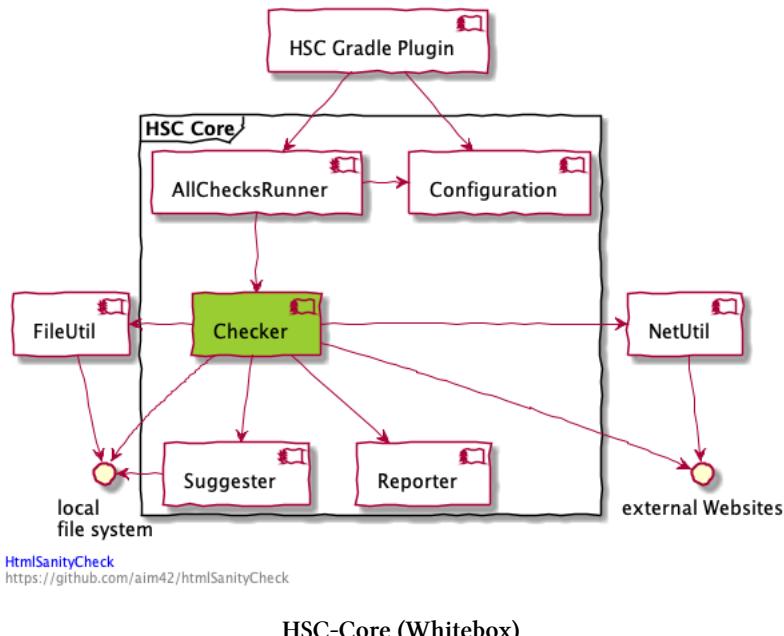
Contained Blackboxes:

Building block	Description
HSC Core	HTML parsing and sanity checking
HSC Gradle Plugin	Expose HtmlSC via a standard Gradle plugin, as described in the Gradle user guide ²⁴ . Source: Package <code>org.aim42.htmlsanitycheck</code> , classes: <code>HtmlSanityCheckPlugin</code> and <code>HtmlSanityCheckTask</code>
NetUtil	package <code>org.aim42.inet</code> , checks for internet connectivity, configuration of http status codes
FileUtil	package <code>org.aim42.filesystem</code> , file extensions etc.
HSC Graphical UI	(planned, not implemented)

²⁴<https://docs.gradle.org/current/userguide/userguide.html>

5.2 Building Blocks - Level 2

5.2.1 HSC Core (Whitebox)



Rationale: The internal structure of HSC Core follows a functional decomposition:

- configuration,
- parsing and handling HTML input,
- checking,
- creating suggestions and
- collecting checking results

Contained Blackboxes:

Building block	Description
Checker	Contains the pure checking functionality. See its blackbox description below.
AllChecksRunner	Facade to the Checkers. Provides a (configurable) interface. Source: <code>org.aim42.htmlsanitycheck.AllChecksRunner</code> . Called by HSC GradlePlugin
Configuration	Handles configuration of input and output location, timeouts, status-code behavior and types of checks to be performed.
Reporter	Reports checking results to either console or file.
Suggester	In case of checking issues, suggests alternatives (<i>did you mean xyz?</i>). Suggestions are included in results.

5.2.1.1 Checker (Blackbox)

The abstract class `Checker` provides the uniform interface (`public void check()`) to different checking algorithms.

Based upon polymorphism, the actual checking is handled by subclasses of the abstract `Checker` class, uses the template-method pattern. It uses the [concept of extensible checking algorithms](#).

5.2.1.2 Suggester (Blackbox)

For a given input (*target*), `Suggester` searches within a set of possible values (*options*) to find the n most similar values. For example:

- Target = “McDown”
- Options = {“McUp”, “McDon”, “Mickey”}
- The resulting suggestion would be “McDon”, because it has the greatest similarity to the target “McDown”.

The implementation is based upon the Jaro-Winkler distance²⁵, one of the algorithms to calculate similarity between strings.

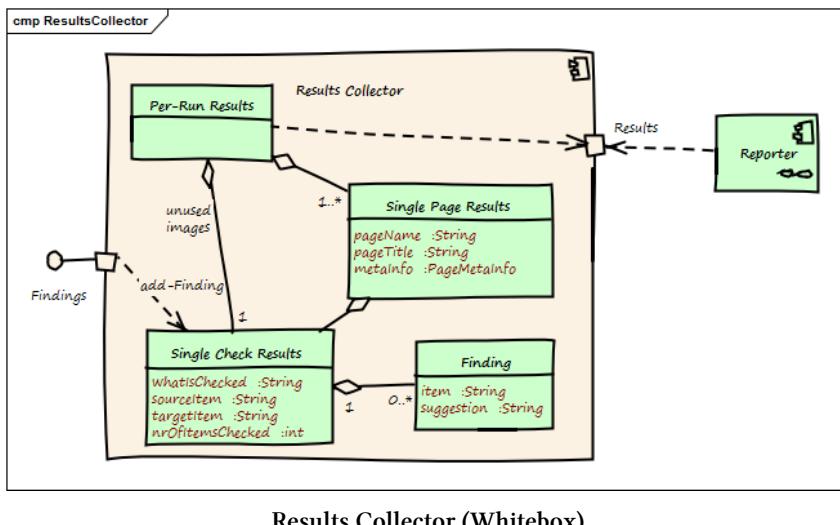
Suggester is used at least in the following cases:

- Broken image links: Compares the name of the missing image with all available image file names to find the closest match.
- Missing cross references (broken internal links): Compares the broken link with all available link targets (anchors).

Source: package org.aim42.htmlsanitycheck.suggest.Suggester

5.3 Building Blocks - Level 3

5.3.1 ResultsCollector (Whitebox)



Results Collector (Whitebox)

Rationale: This structures follows the hierarchy of checks, managing results for:

1. a number of pages/documents (`PerRunResults`),

²⁵https://en.wikipedia.org/wiki/Jaro%20-%20Winkler_distance

2. a single HTML page (`SinglePageResults`) and finally
3. the results of a single check, e.g. the `MissingImagesChecker` (`SingleCheckResults`).

Contained Blackboxes:

Building block	Description
Per-Run Results	Aggregated results for potentially many HTML pages/documents.
<code>SinglePageResults</code>	Aggregated results for a single HTML page
<code>SingleCheckResults</code>	Results for a single type of check (e.g. missing-images check or broken-internal-link check)
<code>Finding</code>	A single finding, (e.g. “image ‘logo.png’ missing”). Can contain suggestions.

II.6 Runtime View

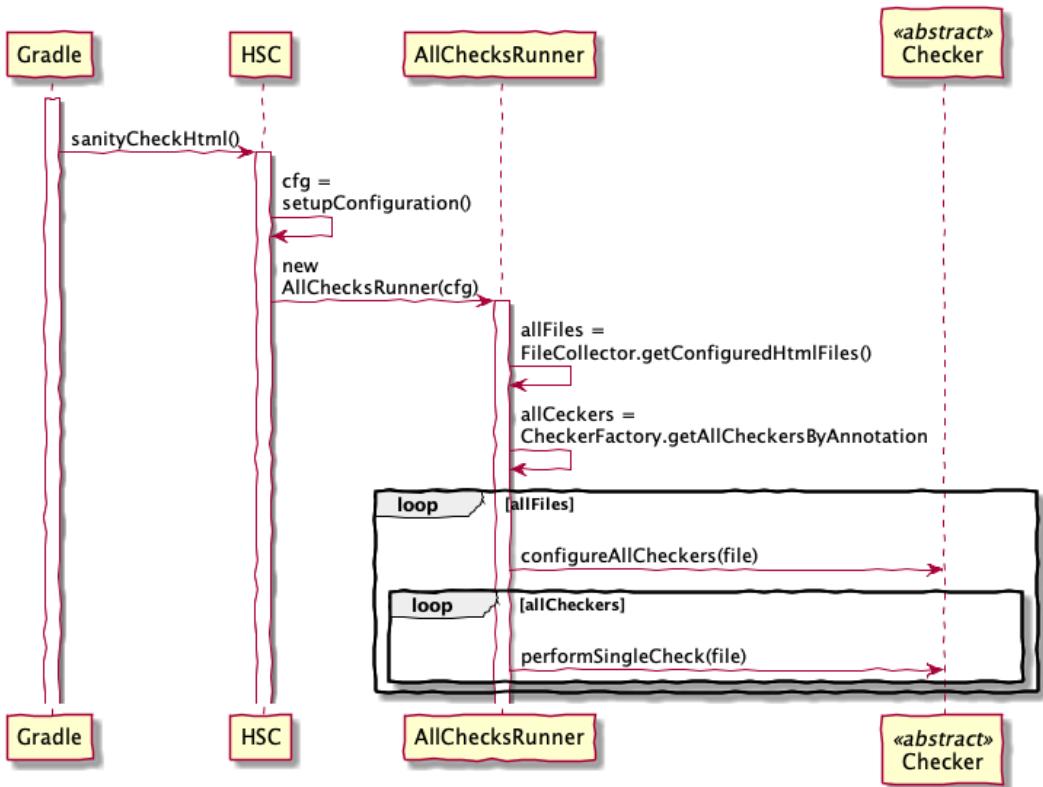
Content and Motivation

The runtime view shows behavior, interactions and runtime dependencies of the building blocks in form of concrete scenarios.

It helps you to understand *how* the building blocks of your systems fulfill their respective tasks at runtime, and *how* they communicate/interact with each other at runtime.

II.6.1 Execute all checks

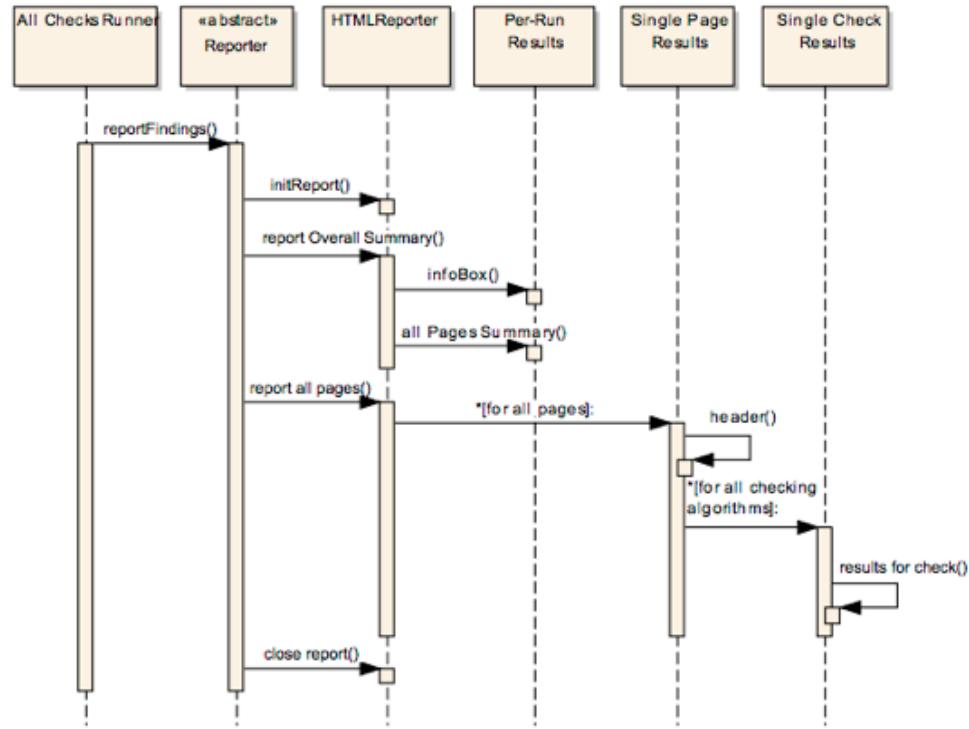
A typical scenario within HtmlSC is the execution of *all* available checking algorithms on a set of HTML pages.



Explanation:

1. User or build calls `htmlSanityCheck` build target.
2. Gradle (from within build) calls `sanityCheckHtml()`
3. HSC configures input files and output directory
4. HSC creates an `AllChecksRunner` instance
5. gets all configured files into `allFiles`
6. (planned) get all available Checker classes based upon annotation
7. perform the checks, collecting the results

II.6.2 Report checking results



Sequence diagram: Report results

Reporting is done in the natural hierarchy of results (see the corresponding concept in [section 8.2.1](#) for an example report).

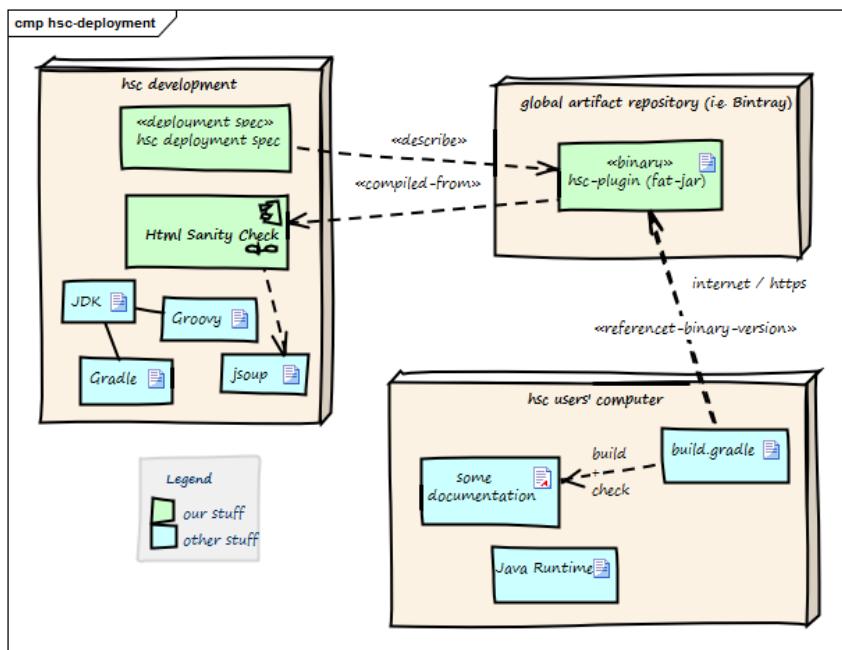
1. per “run” (`PerRunResults`): date/time of this run, files checked, some configuration info, summary of results
2. per “page” (`SinglePageResults`):
3. create page result header with summary of page name and results
4. for each check performed on this page create a section with `SingleCheckResults`
5. per “single check on this page” report the results for this particular check

II.7 Deployment view

Content and motivation

For stable operation, you need to know the technical infrastructure in which your system will run. This is especially important if your software is distributed or deployed on several different machines, application servers or containers.

Sometimes you need to know about different environments (e.g. development, test, production). For large commercial or web systems, aspects such as scalability, clustering, automatic provisioning, firewalls and load balancing also play an important role.



HtmlSC deployment (for use with Gradle)

Node / Artifact	Description
hsc plugin binary	Compiled version of HtmlSC, including required dependencies.
hsc-development	Development environment
artifact repository	Global public <i>cloud</i> repository for binary artifacts, similar to mavenCentral ²⁶ HtmlSC binaries are uploaded to this server.
hsc user computer	Where documentation is created and compiled to HTML.
build.gradle	Gradle build script configuring (among other things) the HtmlSC plugin.

The three nodes (*computers*) shown in the diagram above are connected via Internet.

Prerequisites:

- HtmlSC developers need a Java development kit, Groovy, Gradle plus the JSoup HTML parser.
- HtmlSC users need a Java runtime (> 1.6) plus a build file named `build.gradle`. See below for a complete example.

Example for `build.gradle`

```

1 buildscript {
2     repositories {
3         mavenLocal()
4         maven {
5             url "https://plugins.gradle.org/m2/"
6         }
7     }
8     dependencies {
9         // in case of mavenLocal(), the following line is valid:
10        classpath(group: 'org.aim42',
11
12        // in case of using the official Gradle plugin repository:

```

²⁶<https://search.maven.org/>

```
13      //classpath (group: 'gradle.plugin.org.aim42',
14      name: 'htmlSanityCheck', version: '1.0.0-RC-3')
15  }
16 }
17
18 plugins {
19     id 'org.asciidoc.convert' version '1.5.8'
20 }
21
22
23 // ===== path definitions =====
24 ext {
25     srcDir = "$projectDir/src/docs/asciidoc"
26
27 // location of images used in AsciiDoc documentation
28     srcImagesPath = "$srcDir/images"
29
30 // (input for htmlSanityCheck)
31     htmlOutputPath = "$buildDir"
32
33     targetImagesPath = "$buildDir/images"
34 }
35
36 // ===== asciidoctor ======
37 apply plugin: 'org.asciidoc.convert'
38
39 asciidoctor {
40     outputDir = file(buildDir)
41     sourceDir = file(srcDir)
42
43     sources {
44         include "many-errors.adoc", "no-errors.adoc"  }
45
46     attributes = [
47             doctype   : 'book',
48             icons     : 'font',
```

```
49         sectlink : true,
50         sectanchors: true ]
51
52     resources {
53         from(srcImagesPath) { include '**' }
54         into "./images" }
55 }
56
57 // =====
58 apply plugin: 'org.aim42.htmlSanityCheck'
59
60 htmlSanityCheck {
61     // ensure asciidoctor->html runs first
62     // and images are copied to build directory
63
64     dependsOn asciidoctor
65
66     sourceDir = new File("${buildDir}/html15")
67
68     // files to check, in Set-notation
69     sourceDocuments = ["many-errors.html", "no-errors.html"]
70
71     // fail the build if any error is encountered
72     failOnErrors = false
73
74     // set the http connection timeout to 2 secs
75     httpConnectionTimeout = 2000
76
77     ignoreLocalHost = false
78     ignoreIPAddresses = false
79 }
80
81 defaultTasks 'htmlSanityCheck'
```

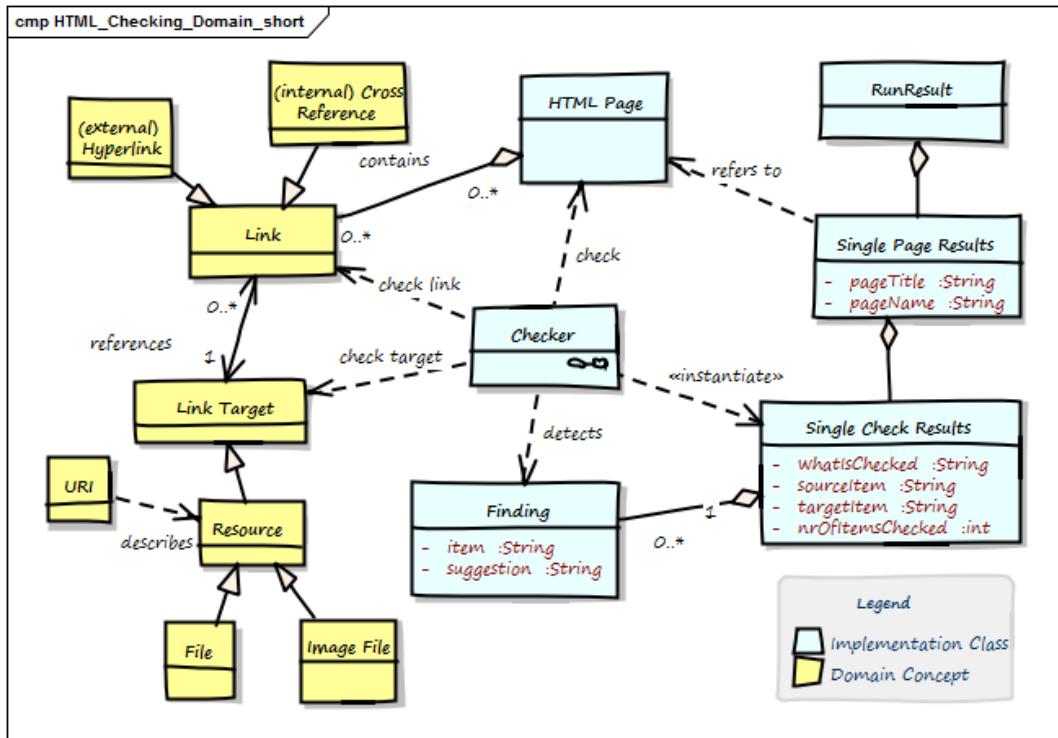
II.8 Cross-cutting Concepts

Content and Motivation

You should explain cross-cutting and ubiquitous rules of your system. arc42 calls them *concepts*: They often affect multiple building blocks and are relevant in several parts of the system and its implementation. Examples include:

- Rules for the usage of technologies and/or frameworks
- Implementation rules, design or architecture patterns used

8.1 Domain Model



HTML Checking Domain Model

Properties of the implementation classes are private, as we manipulate these via getter/setter methods.

Term	Description
Anchor	Html element to create ->Links. Contains link-target in the form <code></code>
Cross Reference	Link from one part of the document to another part within the same document. Special form of ->Internal Link, with a ->Link Target in the same document.
External Link	Link to another page or resource at another domain.

Term	Description
Finding	Description of a problem found by one ->Checker within the ->Html Page.
Html Element	HTML pages (documents) are made up by HTML elements .e.g., , ‘ and others. See the definition from the W3-Consortium²⁷
Html Page	A single chunk of HTML, mostly regarded as a single file. Shall comply to standard HTML syntax. Minimal requirement: Our HTML parser can successfully parse this page. Contains ->Html Elements. Synonym: <i>Html Document</i> .
id	Identifier for a specific part of a document, e.g. <h2 id="#someHeader">.Often used to describe ->Link Targets.
Internal Link	Link to another section of the same page or to another page of the same domain. Also called ->Cross Reference or <i>Local Link</i> .
Link	Any a reference in the ->Html Page that lets you display or activate another part of this document (->Internal Link) or another document, image or resource (can be either ->Internal (local) or ->External Link). Every link leads from the <i>Link Source</i> to the <i>Link Target</i> .
Link Target	Target of any ->Link, e.g. heading or any other a part of ->Html Documents, any internal or external resource (identified by URI). Expressed by ->id.
Local Resource	Local file, either other Html files or other types (e.g. pdf, docx)
Run Result	The overall results of checking a number of pages (at least one page).
Single Page Result	A collection of all checks of a single ->Html Page.
URI	Universal Resource Identifier. Defined in RFC-2396²⁸ , the ultimate source of truth concerning link syntax and semantic.

²⁷https://www.w3schools.com/html/html_elements.asp

²⁸<https://www.ietf.org/rfc/rfc2396.txt>

8.2 Structure of HTML Links

Remark: For many web developers or HTML experts the following information on URI syntax might be completely evident. As we wrote this book also for different kind of people, we included this information anyhow.

HtmlSC performs various checks on HTML links (hyperlinks), which usually follow the URI syntax specified by [RFC-2396²⁹](#). URIs are generally used to link to arbitrary resources (documents, files or parts within documents).

Their general structure is depicted in the following figure - you also find a unit test below.

[protocol://][host][:port][path]?query[#ref]

http://example.com:42/docs/index.html?name=aim42#INTRO

Figure: Generic URI structure

Test showing generic URI syntax

```

1  @Test
2  public void testGenericURISyntax() {
3      // based upon an example from the Oracle(tm) Java tutorial:
4      // https://docs.oracle.com/javase/tutorial/networking/urls/urlInfo.\n
5      html
6          def aURL = new URL(
7              "https://example.com:42/docs/tutorial/index.html?name=aim42#INT\\
8              RO");
9          aURL.with {
10              assert getProtocol() == "http"
11              assert getAuthority() == "example.com:42"
12              assert getHost() == "example.com"
13              assert getPort() == 42
14              assert getPath() == "/docs/tutorial/index.html"
15              assert getQuery() == "name=aim42"
16              assert getRef() == "INTRO"

```

²⁹<https://www.ietf.org/rfc/rfc2396.txt>

```
17     }
18 }
```

8.3 Multiple Checking algorithms

HtmlSC uses the [template-method-pattern³⁰](#) to enable flexible checking algorithms:

“The Template Method defines a *skeleton of an algorithm* in an operation, and defers some steps to subclasses”.

We achieve that by defining the skeleton of the checking algorithm in one operation (`performCheck`), deferring the specific checking algorithm steps to subclasses. The invariant steps are implemented in the abstract base class, while the variant checking algorithms have to be provided by the subclasses.

Template method for performing a single type of checks

```
1 /**
2  * Prerequisite: pageToCheck has been successfully parsed,
3  * prior to constructing this Checker instance.
4 */
5 public CheckingResultsCollector performCheck() {
6     // assert prerequisite
7     assert pageToCheck != null
8     initResults()
9     return check() // subclass executes the actual checking algorithm
10 }
```

Template Method (excerpt)

³⁰https://sourcemaking.com/design_patterns/template_method/

Component	Description
Checker	<i>abstract</i> base class, containing the template method <code>check()</code> plus the public method <code>performCheck()</code>
<code>ImageFileExistChecker</code>	checks if referenced local image files exist
<code>InternalLinksChecker</code>	checks if cross references (links referenced within the page) exist
<code>DuplicateIdChecker</code>	checks if any id has multiple definitions

8.4 Reporting

HtmlSC supports the following output (== reporting) formats and destinations:

- formats (HTML and text) and
- destinations (file and console)

The reporting subsystem uses the template method pattern to allow different output formats (e.g. Console and HTML). The overall structure of reports is always the same.

The (generic and abstract) reporting is implemented in the abstract Reporter class as follows:

Report findings using TemplateMethod pattern

```

1  /**
2   * main entry point for reporting - to be called when a report is requested
3   * Uses template-method to delegate concrete implementations to subclasses
4   */
5
6
7  public void reportFindings() {
8      initReport()           // (1)
9      reportOverallSummary() // (2)
10     reportAllPages()       // (3)

```

```
11     closeReport()           // (4)
12 }
13
14 private void reportAllPages() {
15     pageResults.each { pageResult ->
16         reportPageSummary( pageResult )           // (5)
17         pageResult.singleCheckResults.each { resultForOneCheck ->
18             reportSingleCheckSummary( resultForOneCheck ) // (6)
19             reportSingleCheckDetails( resultForOneCheck ) // (7)
20             reportPageFooter()
21     }
22 }
```

1. initialize the report, e.g. create and open the file, copy css-, javascript and image files.
2. create the overall summary, with the overall success percentage and a list of all checked pages with their success rate.
3. iterate over all pages
4. write report footer - in HTML report also create back-to-top-link
5. for a single page, report the number of checks and problems plus the success rate
6. for every singleCheck on that page, report a summary and
7. all detailed findings for a singleCheck.
8. for every checked page, create a footer, page break or similar to graphically distinguish pages between each other.

The sample report below illustrates this.

HTML Sanity Check Results

Results by Page

Page	Checks	Findings	Success rate
index.html	1364	42	96%

RunResults

Results for index.html

location : /Users/gstarke/projects/aim42/aim42-guide/build/docs/html5/index.html

kByte	checks	issues
370.16	1364	42

PageResults

96% successful

Missing Local Images Check

32 img src attributes checked, 0 missing image files found.

Duplicate Definition of id Check

432 id checked, 1 duplicate id found.

- id "Measure" has 2 definitions.

SingleCheckResults

Broken Internal Links Check

858 href checked, 37 missing id found.

- link target "Invariant" missing

Sample report showing run/page/check hierarchy of results

II.9 Architecture Decisions

Content and Motivation

You like to understand important, huge, expensive, risky or otherwise special architecture and design decisions.

It's especially interesting to keep the *reasons* for these decisions.

9.1 Checking of external links postponed

In the current version of HtmlSC we won't check external links. These checks have been postponed to later versions.

9.2 HTML Parsing with jsoup

To check HTML we parse it into an internal (DOM-like) representation. For this task we use **Jsoup**³¹, an open-source parser without external dependencies.

To quote from their website:

[quote] jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jQuery-like methods.

Goals of this decision: Check HTML programmatically by using an existing API that provides access and finder methods to the DOM-tree of the file(s) to be checked.

Decision Criteria:

- Few dependencies, so the HtmlSC binary stays as small as possible.
- Very easy to use: Simple and elegant (accessor and finder) methods to easily locate images, links and link-targets within the DOM tree.

³¹<https://jsoup.org>

- Highly flexible: Can parse files and strings (and other) input.

Alternatives:

- jsoup: a plain HTML parser without any dependencies (!) and a rich API to access all HTML elements in DOM-like syntax. Clear winner!
- HTTPUnit: a testing framework for web applications and -sites. Its main focus is web testing and it suffers from a large number of dependencies.
- [HtmlCleaner³²](#)

³²<http://htmlcleaner.sourceforge.net/>

II.10 Quality Requirements

Content and motivation

You want to know specific and operational quality requirements, at best in form of a specific *quality tree*. This section completes, augments or refines the top-quality goals already given in arc42-section 1.2.

Remark: For our small example, such a quality tree is overly extensive... whereas in real-live systems we've seen quality trees with more than 100 scenarios. Therefore, we stick to (repeating) a few scenarios here.

Quality Scenarios

Attribute	Description
Correctness	Every broken internal link will be found.
Correctness	Every missing (local) image will be found.
Correctness	Correctness of all checks is ensured by automated positive and negative tests.
Completeness	The results-report must contain <i>all</i> results (aka findings)
Flexibility	HtmlSC shall be extensible with new checking algorithms and new usage scenarios (i.e. from different build systems)
Safety	HtmlSC leaves its source files completely intact: Content of files to be checked will <i>never</i> be modified.
Performance	HtmlSC performs all checks on a 100kByte HTML file in less than 10 seconds.

II.11 Risks and Technical Debt

Content and Motivation

You want to know the technical risks of your system, so you can address potential future problems.

In addition you want to support your management stakeholders (i.e. project management, product owner) by identifying technical risks.

Remark: In our small example we don't see any *real* risks for architecture and implementation. Therefore the risks shown below are a bit artificial...

11.1 Technical risks

Risk	Description
Bottleneck with access rights on public repositories	Currently only one single developer has access rights to deploy new versions of HtmlSC on public servers like Bintray or Gradle plugin portal.
High effort required for new versions of Gradle	Upgrading Gradle from v-3.x to v-4.x required configuration changes in HtmlSC. Such effort might be needed again for future upgrades of the Gradle API.

11.2 Business or domain risks

Risk	Description
System might become obsolete	In case AsciiDoc or Markdown processors implement HTML checking natively, HtmlSC might become obsolete.

II.12 Glossary

Content and Motivation

You need to understand the most important domain terms and expressions, that stakeholders use when communicating about the system, its interfaces, goals and requirements.

The glossary is one manifestation of our pledge for “*explicit, not implicit*”, as it associates words or terms with sense, meaning and semantics.

In the case of our small example, the terms given here should be good friends to most developers. You find a more interesting version of the glossary in [section II-8.1](#).

Term	Definition
Link	A reference within an →HTMLPage. Points to →LinkTarget
Cross Reference	Link from one part of a document to another part within the same document.
External Hyperlink	Link to another HTML-page or to a resource within another domain or site.
Run Result	Combined checking results for multiple pages (→HTMLPages)
SinglePageResults	Combined results of all Checker instances for a single HTML page.

III - Mass Market Customer Relationship Management

By [Gernot Starke](#).

MaMa-CRM takes the burden of (usually paper-based) customer contacts from organizations working in mass markets, like insurance, credit-card providers, mobile telecommunication providers, energy and water providers or large real-estate companies (in MaMa speak these are called «Mandator»)



It has been initially ordered by an independent mid-sized data center to support the launch of the German (government-enforced) e-Health-Card - and later on used to support *campaigns* like telephone billing, electrical-power metering and similar stuff.

For every mandator, there is at least one completely independent MaMa-CRM instance running, which is specifically configured for its mandator and a campaign.

MaMa-CRM architecture documentation is quite heavy in the requirements part, describing several *aspects of flexibility* that triggered many central architecture decisions.

The team that built the system consisted of 7-10 persons working in 2-4 week iterations for about 15 months.

Me (Gernot Starke) had the honor to be part of that team in a responsible role. The original client allowed me to talk and write about the system without disclosing the company name. I was not allowed to use any of the original documentation or source code.

Thanx to Thorsten, Sven, Arno and a few unnamed other guys for great cooperation and a successful finish.

III.1 Introduction and Goals

This document describes the MaMa-CRM platform, a software product family to coordinate processes between mass-market-service-providers and their clients. MaMa-CRM provides the foundation for individual systems, which have to be customized or configured for specific CRM domains.

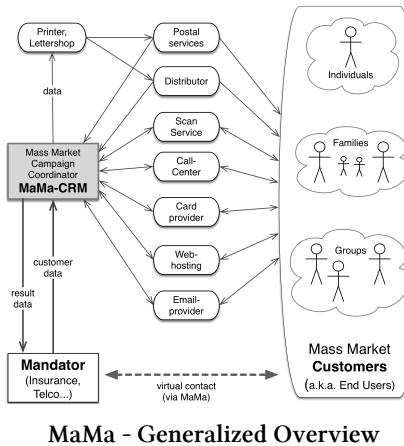
MaMa-CRM has been built and is being operated by an innovative datacenter (let's call it InDAC³³) – a company providing a variety of hosted software services to its clients. Within business process outsourcing initiatives, InDAC is operating specific instances of MaMa-CRM for its customers.

Let's clarify MaMa-CRM's scope with a few brief examples (you'll find more details in the [requirements overview](#)):

- Telecommunication providers offer new or updated tariffs to their customers. These have several possibilities, like email, phone, letter or fax, to respond or issue support queries regarding such offers.
- Retail organizations send specific advertising or marketing material to certain parts of their customers. Again, these can react on several different communication channels. These kinds of advertisements are called *target group marketing*.
- Insurance companies propose modifications or enhancements of existing contracts to specific customer groups. Insured persons or organizations can react over different communication channels, like phone, mail, email or even in person. In case they accept this proposal, they have to submit a legally binding signature.

The following figure shows a generalized overview:

³³The real company behind the MaMa-CRM system prefers not to be disclosed here. The name InDAC (= Innovative data center) is therefore pure fantasy. The original system wasn't called MaMa, but had a different name – I hope you don't mind.



In that figure, the mandator represents an arbitrary company or organization serving mass market customers. MaMa, our system, can host CRM campaigns for many different mandators from different industries.



For an explanation of the terms campaign, activity, mandator and partner, please refer to the glossary in [chapter 12](#)

1.1 Requirements Overview

MaMa-CRM controls customer relationship CRM campaigns, which InDAC (the contractor of MaMa) conducts for his mandators. InDACS' mandators are companies like insurance or telco enterprises – which themselves offer services or products to mass market customers. These mandators outsource their CRM related business processes to InDAC.

MaMa-CRM shall support the following campaign types:

- Changes to tariffs or contracts for insurance companies, telecom and internet service providers and energy suppliers. You'll find a detailed example for this process below.
- Processing of billing information for call-by-call telephone service providers: Mandator submits billing data to MaMa, which coordinates printing, submission and support for the resulting invoices. Management or handling of payments is out-of-scope for MaMa-CRM.

- Management of binary images for credit card and insurance companies, which print cardholders' images onto the cards to prevent misuse. The German e-Health card belongs to this type.
- Meter reading for either energy or water providing companies or large-scale real-estate enterprises.

As MaMa is the foundation for a whole family of CRM systems, it shall be adaptable to a variety of different input and output interfaces and channels without any source code modification! This interface flexibility is detailed in [section 1.1.2](#).

1.1.1 Campaign Example: Mobile Phone Contract Modification

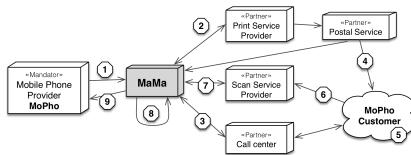
An example shall clarify the complex business and technical requirements to MaMa.

MoPho is a (hypothetical) mobile phone provider with a large customer base and a variety of different tariff options (flat fee, time-based, volume-based etc.). Some of these tariffs or tariff combinations are not marketable any longer (a shorthand for “MoPho does not earn its’ desired profit from them... - but that’s another story). Others are technically outdated.

In their ongoing effort to optimize their business, MoPho management decided to streamline their tariff landscape. Within a large campaign they contact every customer with outdated tariffs and offer upgrades to new and mostly more beneficial tariff options. Some customers get to pay less, but have to accept a longer contract duration. Others will have to pay a little more, but receive additional services, increased capacity, bandwidth or other benefits. In all cases where customers accept the new tariff offer, the formal contract between MoPho and the customer has to be updated, which requires a valid signature to be legally binding³⁴.

MoPho intends to inform certain customers via printed letters of the new tariff offerings. Customers can react via letter, fax, phone or in person at one of MoPho’s many sales outlets. As MoPho’s core competency are *phone services*, it doesn’t want to bother with printing letters, scanning and processing replies, answering phone inquiries and dealing with other out-of-scope activities. Therefore they employ MaMa-CRM as an all-around carefree solution. Look at MaMa’s approach to this scenario:

³⁴In some countries or for some contract types there might be simpler solutions than a written signature. Please ignore that for the moment.



Telecommunication Example Scenario

The 9 steps depicted in that scenario need some explanation:

1. MoPho selects pertained customers from its internal IT systems. We're talking about a 30+ million customer base and approximately 10 million of those customers will be part of this campaign. MoPho exports their address-, contract- and tariff data and transmits these to MaMa-CRM. MaMa imports this customer data.
2. MaMa forwards only the relevant parts of customer data to the print service provider (a «Partner»). This company creates personalized letters from address and tariff data, prints those letters and finally delivers them to the postal service (which ensures the letters finally end up in customers' mailboxes).
3. MaMa now informs the call center (also a «Partner») participating in this campaign, so they can prepare for the customers reaction.
4. In the meantime the print service has delivered the letters to the postal service, which delivers letters to the customers. This is more difficult than it sounds: 1-5% of addresses tend to change within 12 month, depending on the customer base. Some customers refuse to accept marketing letters. The postal service informs MaMa about all the problematic cases (address unknown, forwarding request etc.), so MaMa can decide on further activities.
5. The customers decide what to do with the letter: There's a multitude of options here:
 - Customer fills out the enclosed reply form and signs it.
 - Customer fills out the reply form, but forgets to sign it.
 - Customer does not understand this offer and sends written inquiry by letter.
 - Customer inquires by phone call.
 - Customer ignores letter and does not react at all.
 - There are additional special cases (customer is not contractually capable, has a custodian or legal guardian, is under-age, deceased or temporarily unavailable...).

6. Let's assume the customer accepts and sends the letter back via postal service. These letters will be forwarded to the scan service provider (again, a «Partner»), which scans them and performs optical character recognition.
7. MaMa imports the scan-results from the scan service provider. Again, several cases are possible.
8. MaMa distinguishes between all possible cases and takes appropriate actions:
 - The form is completely filled and signed.
 - The form is completely filled, but customer forgot to sign.
 - Customer signed, but forgot to fill other important fields in the form...
 - If the form is not returned within an appropriate period of time, MaMa might decide to resend the letter, maybe even with a different wording, layout or even a different contractual offer.
 - Surely you can imagine several other possible options...
9. Finally, for every customer who accepted the changed tariff agreement, MaMa sends back the appropriate information, so the mandator MoPho can internally change the corresponding master data, namely contracts and tariffs and take all other required actions (activities in MaMa).

For the phone service provider MoPho, the primary advantage is the centralized interface to all customer interaction: All processes required to conduct this crucial campaign are handled by MaMa-CRM, without intervention by MoPho. Very practical for MoPho is the nearly unlimited flexibility of MaMa-CRM to import and export a variety of different data formats. This flexibility enables MaMa-CRM to easily incorporate new campaign partners (like a second scan service provider, an additional email-service provider or web hosting partner).

1.1.2 Campaign Configuration

MaMa provides the software foundation for *campaign instances* that have to be extensively configured to operate for a specific mandator with a number of specific partners.

The following section gives an overview of such a configuration for the MoPho campaign described in the previous [section](#).

1. Configure client master data

Define and configure required data attributes and additional classes, usually this information is completely provided by the mandator.

This data (e.g. contract-, tariff-, offer- and other business-specific entities or attributes) are *modeled* as extensions of the MaMa base model in UML.

In the MoPho telecommunications example, this data includes:

- Existing tariff
- List of potential new tariffs
- Contract number
- Validity period of existing tariff
- Validity period of new tariff

2. Configure communication channels

- Define and exchange security credentials (passwords, certificates, user-ids).
- Determine contact address, especially for handling technical and process issues.

3. Define campaign meta data

- start- and end dates,
- target dates for specific activities,
- reply address for letters, reply email address,
- phone number for call center,

4. Define campaign activities

Campaign activities³⁵ can be either input-, output- and MaMa-internal activities.

- Output-activities *export* data to partners or the mandator, e.g:
 - PrintLetter
 - DataToCallCenter
 - DataToScanServiceProvider
 - ResultsToMandator

³⁵See the [MaMa-CRM Glossary](#) for definitions of these term.

- ProductionOrderToCardProvider
- Input-activities *import* data provided by the mandator or partners, e.g.:
 - MasterDataFromMandatorImport
 - ScanDataImport
 - CallCenterImport
 - SalesOutletImport
- Internal activities define data maintenance operations, e.g.:
 - Delete customer data 60 days after completion.
 - Delete all campaign data 120 days after campaign termination.

5. Model campaign flows

What is the required flow of activities, what needs to happen under which conditions (including plausibility checks).

1.1.3 Activities Subject to Charge

Some activities result in charges, for example:

- sending letters via postal service results in postal charges
- producing e-Health cards results in card production charges

As we are talking about potentially millions of clients respectively activities, these charges will amount to a significant flow of money:

MaMa-CRM does NOT handle billing and payment processes.

Instead these are handled parallel to MaMa-campaigns by InDACS' finance department³⁶.

³⁶In the real system, billing and payment was usually processed by the mandators (!) - but these 3-party processes are not significant for the software architecture, and therefore left out of this documentation.

1.1.4 Additional Requirements

Status and Financial Reporting

MaMa has to campaign reports containing all campaign activities and their results (e.g. how many letters were printed, how many calls from clients were processes in the call center, how many clients sent their return letter etc.)

It's especially important to report about activities subject to a charge (aka having financial consequences). Although MaMa-CRM is not responsible for the resulting billing and payment processes, mandators and partners need reliable and trustworthy reports about such activities.

MaMa shall offer adequate standard reports covering these cases.

Client Data Belongs To Mandators

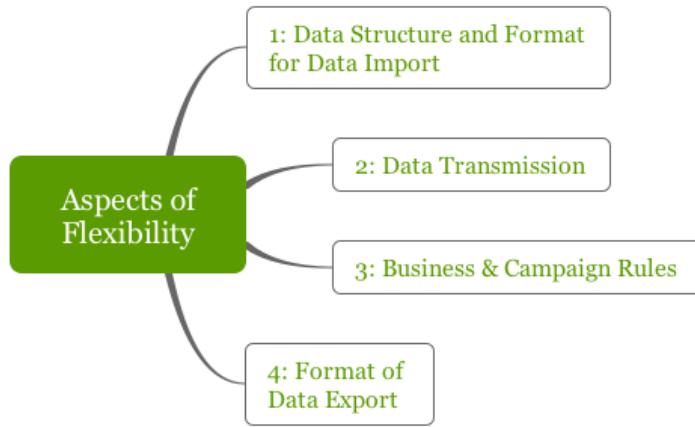
MaMa-CRM follows a very strict data protection policy: All data processes in campaigns is somehow related to clients and needs to be kept strictly confidential. Mandators always remain owners of all such data, from whatever partner it is transferred to MaMa by whatever activities.

After a campaign officially ends, all (!) data including backups, configurations, data and metadata needs to be deleted³⁷.

³⁷Effectively this requirement forbade the use of the well-known version control systems "subversion" (svn), as by development time of MaMa-CRM, svn could not reliably *purge* files from version history! Therefore a campaign-spanning svn repository was not allowed.

1.2 Quality Goals

1.2.1 Flexibility First



Simply stated, MaMa is a data hub with flexibility in several aspects: It imports data (aspect 1) coming over various transmission channels (aspect 2) executes a set of specific business rules (aspect 3) to determine further activities to be performed with this data. Such activities consist of data exports (aspect 4) to certain business partners (like print-, scan- or fax service providers, call centers and the like).

MaMa has to be flexible concerning all these aspects.

Aspect 1: Flexibility in Data Structure and Format

MaMa always deals with variants of clientdata - the most simple version could look like an instance of the following class definition:

Generic Definition of Client Record

```
1 public class Client
2     String primaryKey
3     String lastName
4     String firstName
5     Date birthDate
6     String title
7     Address postalAddress
```

MaMa needs to import such data from various campaign partners. These partners operate their own IT systems with their own data formats. They will export the affected data records in some format (see below) of their choice. MaMa needs to handle many such formats:

- Comma separated value (CSV): attributes are separated by a (configurable) delimiter, e.g.: “id01”, “starke”, “gernot”, “23-jul-1963”, “willi-lauf”, “D-50858”, “cologne”, “germany”
- Field order and delimiter can vary.
- Elements could be enclosed in either “ or ‘ or even «».
- Fix length formats, where every data record has a fixed length. Shorter entries are padded with empty strings or zeroes.
- XML dialects, either conforming to an appropriate schema or DTD.

Regardless of format, data elements need to comply to validation or plausibility rules. Good news: No serialized objects from programming language runtimes (e.g. from a java virtual machine) have to be processed.

Aspect 2: Flexibility in Transmission Formats and Modes

MaMa needs to handle the following transmission-related aspects:

- Standard transmission protocols (ftp, sftp, http, https) as both client and server
- Compressed,
- Encrypted
- Creation of checksums, at least with counters, MD5 and SHA-1
- Mandator- or campaign specific credentials for secure transmission
- Transmission metadata, i.e.: Count the number of successful transmissions and send the number of the last successful transmissions as a prefix to the next transmission.

Aspect 3: Flexibility in Business / Campaign Rules

MaMa needs to support several categories of campaign rules (sometimes called business rules)

- What are required and possible activities in this campaign?
- In what order shall these activities be executed?
- Certain types of activities (especially data import activities) are sometimes triggered by «Partner» organizations. What kind of event triggers what activities?
- How often to remind clients/customers by letter?
- When to call clients/customers by phone?
- What data is mandatory from the customer and what is optional?
- How long to keep data records, when to delete or archive data records?
- Whom to contact when certain (data) conditions arise?

Aspect 4: Flexibility in Data Export

Now that MaMa has received and processed data (by import, transmission and rule processing), it needs to send data to the campaign partners. Analogous to data import, these receiving partners have very specific requirements concerning data formats.

It is pretty common for MaMa to receive data from a partner company or mandator in CSV format, import it into its own relational database, process it record-by-record, export some records in XML to one partner and some other records in a fix format to another partner.

1.2.2 Quality Goals (Scenarios)

Prio 1: Flexibility

MaMa-CRM can:

- import and export (configurable) CSV and fix-record-length data.
- import and export data via ftp, sftp, http and https, as client and server.
- handle compressed data. Compression algorithm is configurable per activity, standard lossless have to be supported.
- handle encrypted data. Security credentials need to be exchanged between parties prior to campaigns during campaign setup. PGP or derivatives are recommended.
- handle hashes or similar transmission metadata, configurable per campaign.

For all these topics, InDAC administrators can fully configure all campaign-specific aspects within one workday.

Prio 2: Security

MaMa will keep all its client and campaign related data safe. It shall never be possible for campaign-external parties to access data or metadata.

InDAC Administrators need to sign nondisclosure contracts to be allowed to administer campaign data. This organizational issue is out-of-scope for MaMa.

Prio 3: Runtime Performance

MaMa can fully process 250.000 scanned return letters within 24 hours.

1.2.3 Non-Goals (Out of Scope)

- MaMa shall be exclusively operated by InDAC. It shall not evolve into a marketable product.
- MaMa shall not replace conventional CRM systems, like Siebel™, salesforce.com™ or others.
- MaMa does not handle billing and payment of any charges generated by its activities, e.g. sending letters or others. See [section 1.1.3](#)

1.3 Stakeholder

Role	Description	Goal, Intention
Mandator	The organizations paying InDAC for conducting one or more campaigns	Minimal effort to get data <i>into and out of</i> MaMa-CRM, minimal effort to configure campaign rules etc.
Management of contractee (InDAC)	Requires flexible and performant foundation for current and future business segments.	Periodical cooperation on current and changed requirements, solution approaches.
Software developers participating in MaMa implementation	Design MaMa internal structure, implement building blocks.	Active participation in technical decisions.
Partner	Companies supporting parts of the campaign processes, like print- or scan service providers, postal service, card producers, call centers. Partner need to send and receive data from MaMa, need to negotiate interface contracts, organizational and operational details.	Effective, productive and robust interfaces between MaMa and partner.
Government agency	Several agencies regulate MaMa campaigns, for example: Insurance regulation body or governmental IT-security body.	Ensure that MaMa fulfills all data security regulations

Role	Description	Goal, Intention
Eclipse-RCP (rich client platform)	Open source project (provider of UI framework)	MaMa uses Eclipse rich client platform to implement the graphical user interface. Eclipse changes its API quite often and therefore (unpredictably) influences UI development.

1.3.1 Special Case: German e-Health Card



InDAC hosted several campaigns in the evaluation phase of the German e-Health card, which had to comply to business and technical standards issued by the corresponding standardization body. That was a company especially founded for this purpose by the German Government.

Interesting fact for the MaMa-CRM development team: That company wasn't operational when MaMa development started - therefore the standards that had to be adhered to hadn't even been created or published...

1.3.2 «Partner» or «Mandator» Specify Interface Details

MaMa-CRM offers detailed interface specification documents, which partners (like print- or scan service providers) can use to implement, configure and operate their interfaces to MaMa. Due to MaMas great flexibility it usually works the other way round: Mandators and Partners provide MaMa with *their* interface descriptions, which MaMa administrators only configure to make it operational.

III.2 Constraints

General Constraints

Setup Campaign without programming

To setup a campaign, no *programming* shall be necessary. *Configuration* in various forms is allowed.

Implementation based upon Java

MaMa-CRM shall work on a recent Java runtime (>1.6)

Use Oracle(tm) as database

InDAC holding company has negotiated a favorable deal with Oracle Inc. considering license and maintenance fee, therefore Oracle-DB has to be used for data storage.

Software Infrastructure Constraints

- Linux operating system (preferably RedHat Enterprise Linux, as there exist *hardened* editions certified by several security organizations).
- Open source frameworks with liberal licenses are possible (esp.: GNU and FSF licenses not allowed)
- Code generation / MDSD preferred for development
- Use of (UML) modeling tool recommended
- InDAC prefers iterative development processes but does not impose them.
- Sound technical documentation: InDAC emphasizes long-lasting, robust and cost-effective software systems and therefore strongly requires maintainable, understandable and expressive technical documentation (part of which you are currently reading).

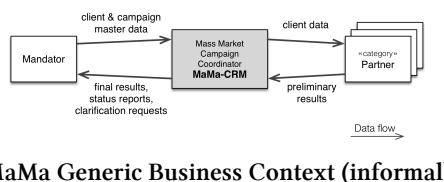
Operational constraints

- Every MaMa-CRM instance shall be operable in its own virtual machine
- MaMa shall run in batch/background mode to minimize operation overhead
- Complete configuration shall be possible from custom Eclipse plugin (alternatively: via browser)

III.3 System Scope and Context

3.1 (Generic) Business Context

Every MaMa instance communicates with a single mandator and one or more partner organizations, like shown in the diagram below. Partners are external service providers, for example printer, mail delivery services, scan services, call-center or internet hosting providers.



MaMa Generic Business Context (informal)

Interface / Neighbor system	Exchanged Data
Client and campaign master data	(inbound, from mandator) Mandator transfers campaign and client master data to MaMa-CRM.
Final results (outbound, to mandator)	MaMa transfers final campaign results back to mandator. This is the ultimate goal of the campaign.
Status reports	MaMa periodically sends status reports to the mandator and interested partners.
Clarification requests	Sometimes client data is wrong, outdated or corrupted, so that certain activities within the campaign cannot be executed for this client. In such cases, MaMa sends clarification requests to the mandator: The corresponding client data has to be checked - and returned to MaMa in corrected way or the client is revoked and will not be processed any further by MaMa.

<u>Interface / Neighbor system</u>	<u>Exchanged Data</u>
Client data (outbound, to partner)	MaMa sends client data to partners, depending on campaign business rules and processing results. MaMa has a distinct interface configured for every partner.
Preliminary results (inbound, from partner)	Partners send results of their respective work back to MaMa. This data is called “preliminary results”, as it requires processing and evaluation by MaMa before it can be marked as final. Process logs and partner status report are also transmitted to MaMa via this interface.

Client data (outbound)

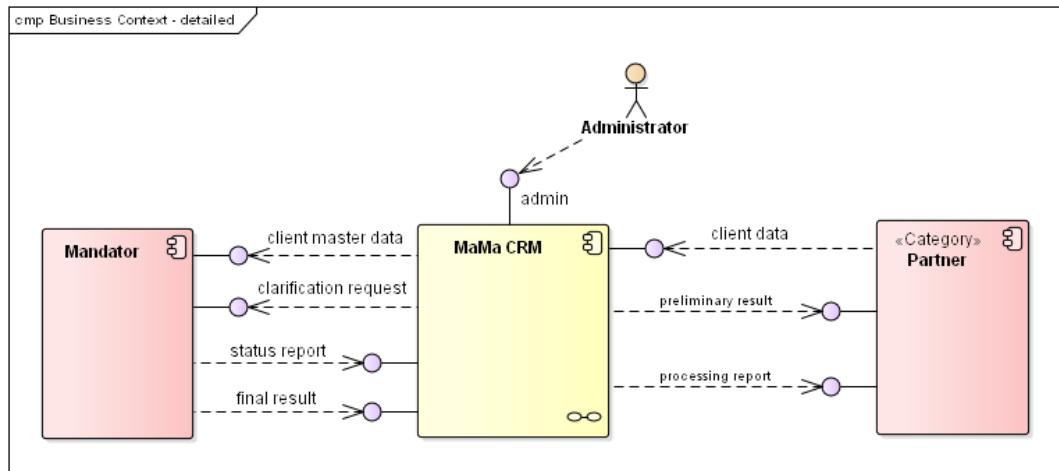
Client data is sent to partners on a “need-to-know” basis to achieve data minimality: Every partner organization gets only the data they absolutely require to fulfill their campaign tasks.

For example, MaMa will not disclose clients’ street address to call centers (they usually get to know name, phone contact and sometimes one or two additional attributes for verification purposes.)

On the other hand, print service providers usually don’t get to know the phone numbers of clients, as the latter is not required to deliver printed letters via postal services.

3.1.1 Formal Business Context

The diagram below contains a more formal version of the context diagram. It includes an `admin` interface, which was left out in the informal version above.

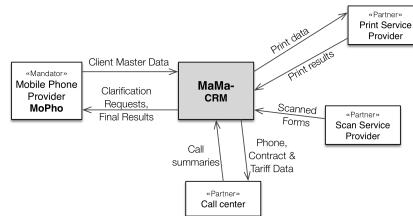


MaMa Generic Business Context (informal)

The `admin` interface enables MaMa and campaign administrators to perform all required administrative tasks needed to init, configure and operate campaigns.

3.1.2 Specific Business Context: Mobile Phone Contract Modification

The following diagram details the example already shown in [section 1.1.1](#).



Mobile Phone Example Context

The data flows are detailed (in excerpts!) in the following table:

Context Details for Mobile Phone Contract Modification Campaign

Neighbor System	Exchanged Data	Format
Mandator (inbound)	Client Master Data: Name, Address, Contact, Contract, Tariff. Once for every client in the campaign, second as response to clarification requests.	Zip-compressed CSV, via sftp (mandator uploads)
Mandator (outbound)	Final results: ID, tariff and contract details for every client who accepted the contract modification proposal	Zip-compressed CSV over sftp, MaMa uploads
Mandator (outbound)	Clarification request	-- " --
Print Service Provider (outbound)	Print Data: Name, Address, parts of contract and tariff.	Zip-compressed, PGP-encrypted CSV via http upload
...

Mapping of Attributes to CSV-Fields

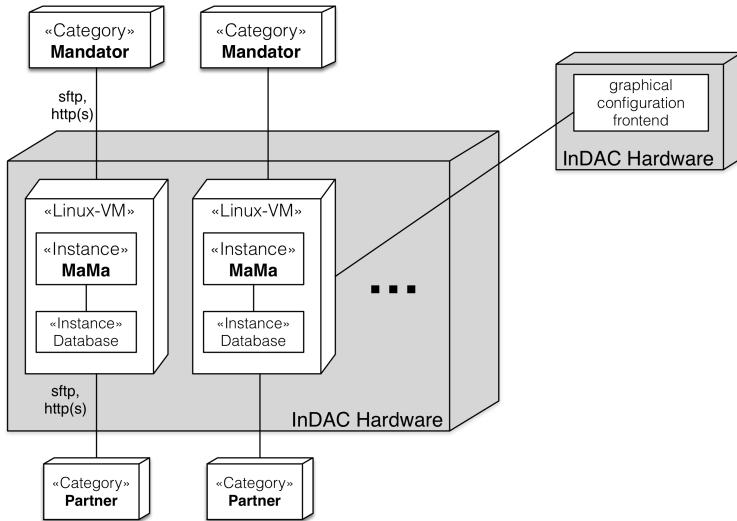
For every instance of MaMa, the mapping of data attributes to fields/records in data transmissions has to be specified in detail. This is done by a domain specific language, details are described in [section 8.3 on CSV-Import/Export](#)

3.2 Technical / Deployment Context

MaMa instances are supposed to run distinct virtual machines (whereas certain³⁸ mandators or campaigns require instances to be deployed on their own physical hardware - which results in significantly higher campaign costs.)

Details of the MaMa deployment are explained in the [deployment view in section 7](#)

The following diagram gives a schematic overview of the typical MaMa deployment setup.



Typical MaMa Deployment Context

³⁸Some mandators with extremely high security requirements negotiated their own distinct physical hardware for their MaMa instance(s).

Element	Description
«Instance» MaMa	A distinct instance of MaMa, running a specific campaign (connected to a single mandator and a number of campaign-specific partner organizations)
InDAC Hardware	Physical server (Dell, HP or similar), located on InDAC premises. Running RHE Linux and a virtualization environment (not shown in diagram)
«Category» Mandator	For every MaMa instance there is one distinct mandator.
«Category» Partner	For every MaMa instance there might be several different partner organizations, each one having a distinct communication channel.
«Instance» Database	Every MaMa instance has its own database instance, usually within the same virtual machine.
Linux VM	Virtualized (RHE) Linux environment. Configured to disallow unwanted external access (e.g. ssh only allowed from within InDAC)

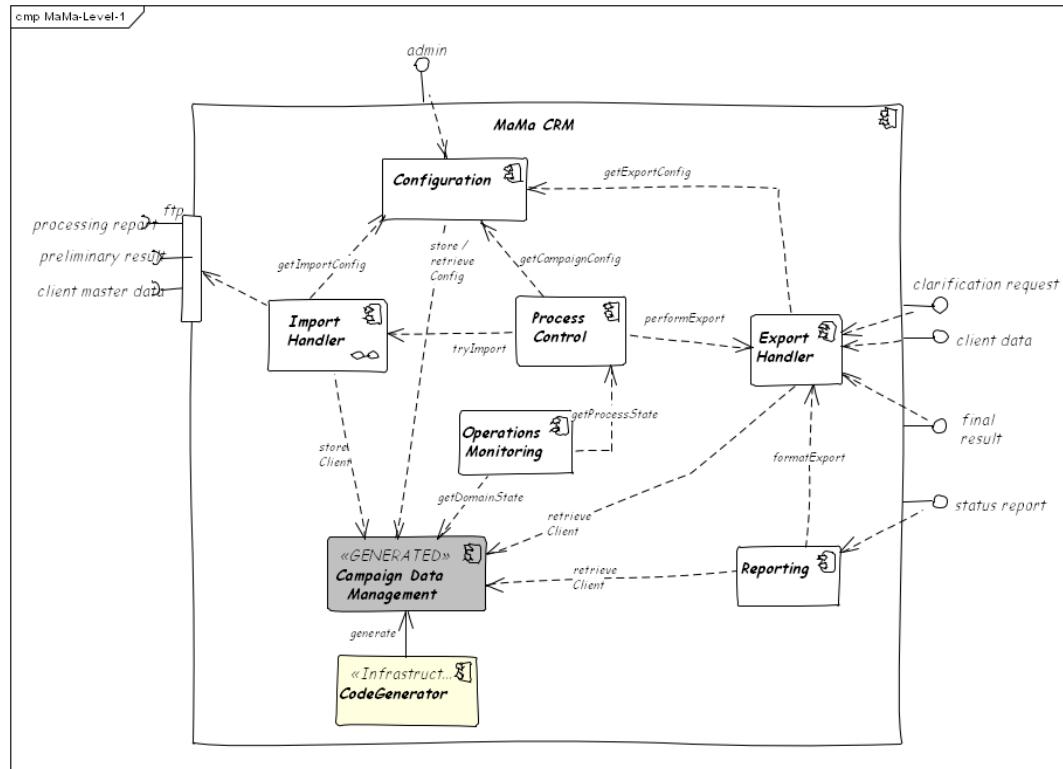
III.4 Solution Strategy

Here you just find the shorthand form of the architectural approaches to the most important (quality) requirements, plus the links to the detailed description in section 8 (crosscutting concepts).

Goal/Requirement	Architectural Approach	Details
Flexible Data Structure	Database structure + persistence code is completely (100%) generated from UML-model	Section 8.1
Flexibility in Transmission Formats (CSV and fix-record-formats)	Create domain-specific languages for CSV and fix-format import/export configurations. Build an ANTLR based parser for these languages plus the corresponding interpreters.	Section 8.2
Flexibility (Configurable CSV/fix formats)	Implement customized editor for CSV/fix DSL as Eclipse plugin	Section 8.2
Performance (import/process 250k images/24hrs)	Treat images as special case, store images in filesystem instead of database, create unique path/filename based upon client-ID, include load-testing in automatic build, create test-data generator	Include special case for image persistence in code generator, Section 8.1

III.5 Building Block View

5.1 MaMa Whitebox Level 1



MaMa Whitebox (Level1)

Rationale: The structure of building blocks within MaMa is based upon *functional decomposition* and the concept of *generated persistence* (see section 8.1).

Contained Blackboxes:

Element	Description
Import Handler	Imports data from Partners or Mandator via external interfaces
Export Handler	Exports data to Partners or Mandator via external interfaces
Configuration	Maintains configuration of all import- and export activity types, import- and export filters and campaign business rules. Includes syntax driven editors for configuration.
Reporting	Reports campaign state to Mandator and Partners, as configured.
Process Control	Responsible for management and execution processes within a campaign, especially for execution of campaign specific business rules.
Campaign Data Management	Completely generated. Stores all client- and campaign data.
Operations Monitoring	Monitors (and reports) all import and export processes plus database and application state.
Code Generator	Generates the (complete) <code>CampaignDataManagement</code> from a campaign specific UML model. See persistence concept for details.

Import Handler (Blackbox)

Intent/Responsibility: Import Handler contains the core functions to imports data from Partners or Mandator via external interfaces. It handles csv, fix-format or xml input, either or both encrypted and compressed of configurable structure.

Interfaces:

Interface (From-To)	Description
getImportConfig	Read all required configuration information to perform imports, especially details about data structures (like csv formats) and filter chains.
storeClient	Sends an imported instance of Client to CampaignDataManagement to be either updated or inserted.
tryImport(from ProcessControl)	ProcessControl either calls or schedules a specific imports activity.
ImportHandler -> external port	ImportHandler needs to access various external entities, like ftp server, file system or even remote access to Mandator or Partner hosts.

Quality of Service:

ImportHandler implements extensive failure handling mechanisms and can therefore deal with a large number of error categories (e.g. communication errors, data format errors, compression and encryption issues and so forth.)

Details:

For details see the [Import Handler \(Whitebox\)](#).

Configuration (Blackbox)

Intent/Responsibility: Configuration is responsible to provide deploy-time flexibility to all MaMa subsystems. It handles the following kinds of configuration information:

- Data import and export configuration
 - csv, fix-format and xml formats
 - transmission and routing information, endpoints, network configuration
 - configuration for compression, encryption and similar filter operations
 - account and security information required to communicate with the campaign-specific external systems.
- Campaign configuration
 - validation rules
 - activities: what kind of imports, exports and maintenance activities are required for this campaign?
- Configuration for archiving of imported data

Interfaces:

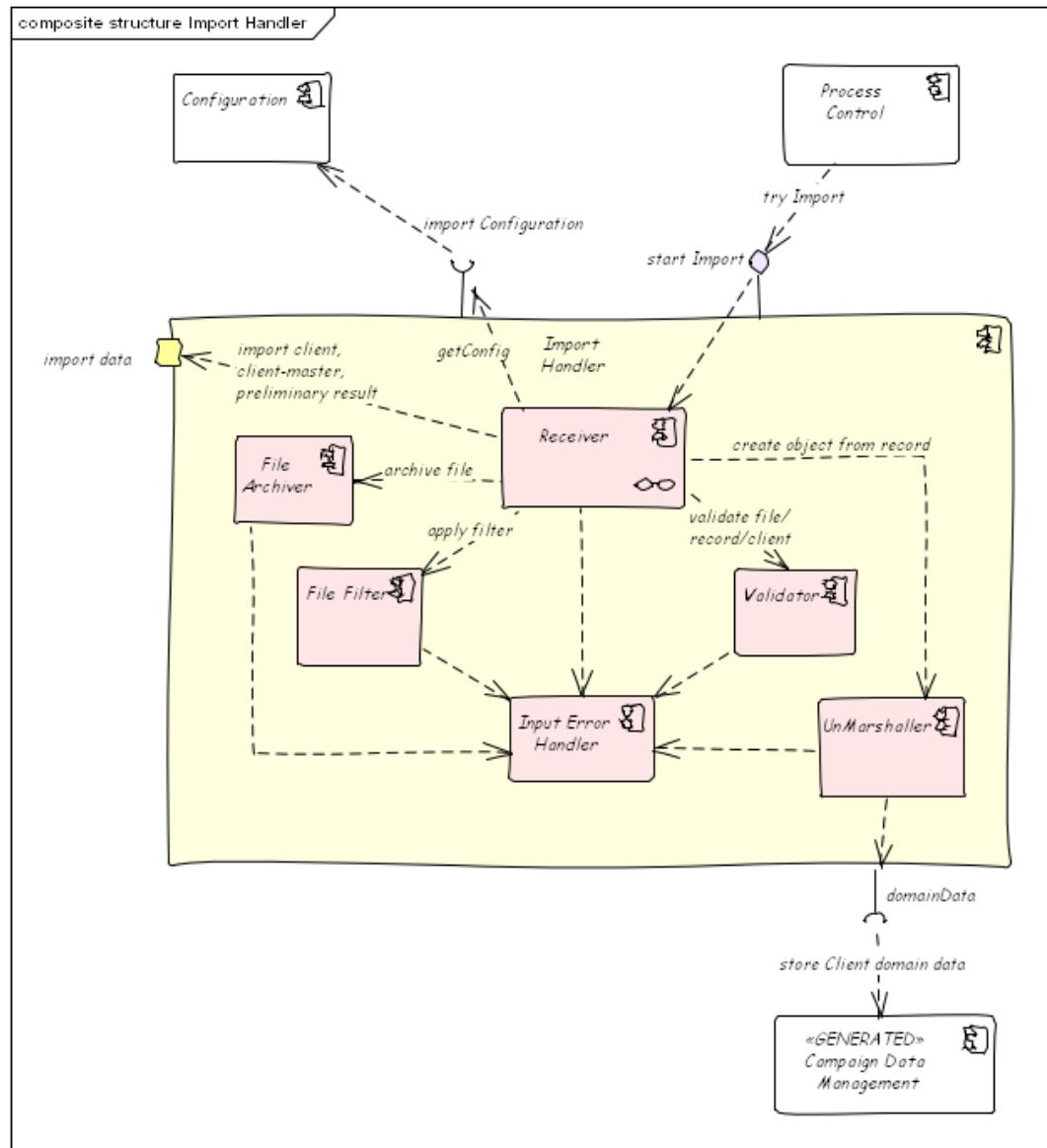
- For all configuration methods, the campaignID and mandatorID need always be input parameters.
- Configuration information is always subclass of the (abstract) superclass Configuration.

Interface (From-To)	Description
getImportConfig	Methods to get import configurations for a specific campaign.
getExportConfig	Methods to get export configurations for a specific campaign.
getCampaignConfig	
store/retrieveConfig	Calls DataManagement to store/retrieve configuration data.

Quality of Service: (not documented)

MaMa Level 2

Import Handler (Whitebox)



Rationale: This is (again) based upon *functional decomposition* of the generic import process. [Section 6.1](#) describes the runtime behavior of this component.

Contained Blackboxes:

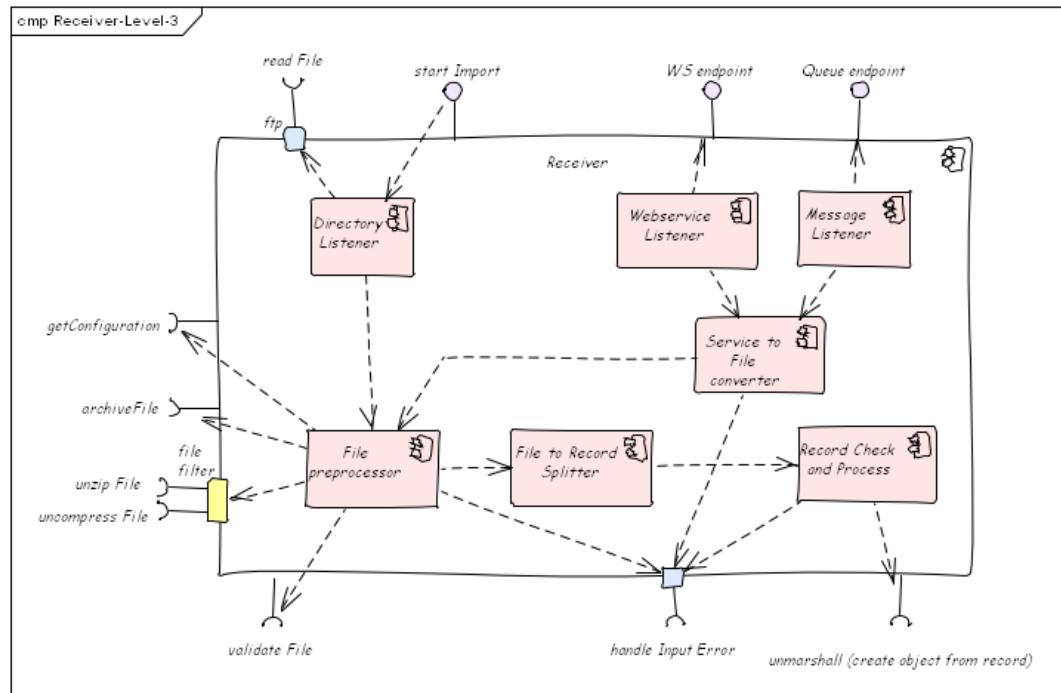
Element	Description
Receiver	Receives data from partners or mandators via the ImportData port.
ImportErrorHandler	Handles the various possible errors during import. With severe errors, import is stopped. Many (especially record or object level) errors are recoverable - these will be logged, eventually the administrator is notified.
ImportData (Port)	Connection to the outside world - via ftp and http, usually transmitted via VPN.
FileArchiver	Non-erasable archive where all imported files are kept for auditability.
FileFilter	Various filter operations, like decrypt, unzip etc. Explained in the filter concept in section 8
Validator	Checks files, records (collections of strings) and client objects for validity.
UnMarshaller	Creates Java objects from collections of strings by using reflection magic. You don't want to know all the dirty details of this component.

Important Interfaces:

Not documented.

MaMa Level 3

Receiver (Whitebox)



Rationale: We have to admit that this structure just evolved out of a number of prototypes. A more functional oriented design would most likely improve understandability, but we never refactored the code into that direction due to different priorities.

Contained Blackboxes:

Element	Description
Directory or WebService or Message - Listener	Components that listen for input of specific kinds, e.g. the DirectoryListener watches for new files to appear in certain directories, (configurable) either in a local or remote file system.

Element	Description
FileProcessor	Completely handles input files, calls all required operations to be performed on the file (archive, unzip, decrypt etc.). A big mess of spaghetti code - you don't want to look at it...
FileToRecordSplitter	Depending on configuration, creates a collection of records from the imported file. Most often a record is represented by a single row/line within the file, but sometimes several lines from the file have to be combined.

III.6 Runtime View

6.1 Import File

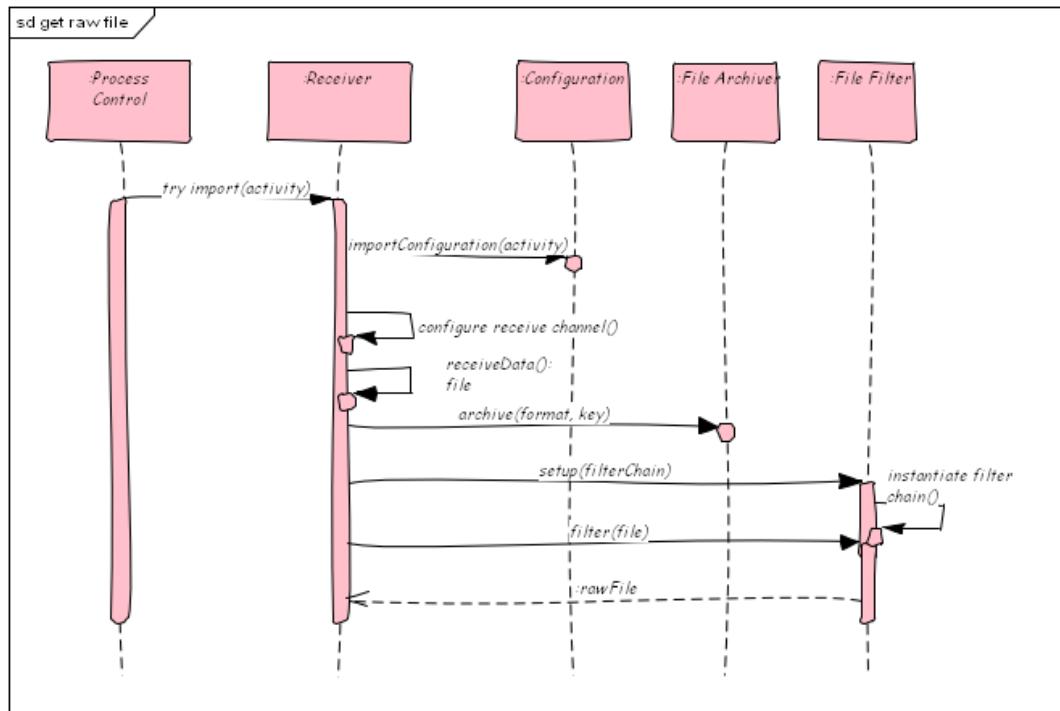
One of the major use cases is *Import File*, which can be from both mandator and partner. Such files contain always contain Client related data in configurable formats (CSV, fix-formats or XML).

We split the explanation of *import file* into two phases:

1. Import Raw File Generic (from an external source)
2. Validate the imported data and update the internal Client database

6.1.1 Import Raw Generic

At first we explain the *generic* import, where no campaign-specific activities are executed. This concerns `configureReceiveChannel` and especially the `instantiateFilterChain()` activities.



(First part of data import:) Import Raw File

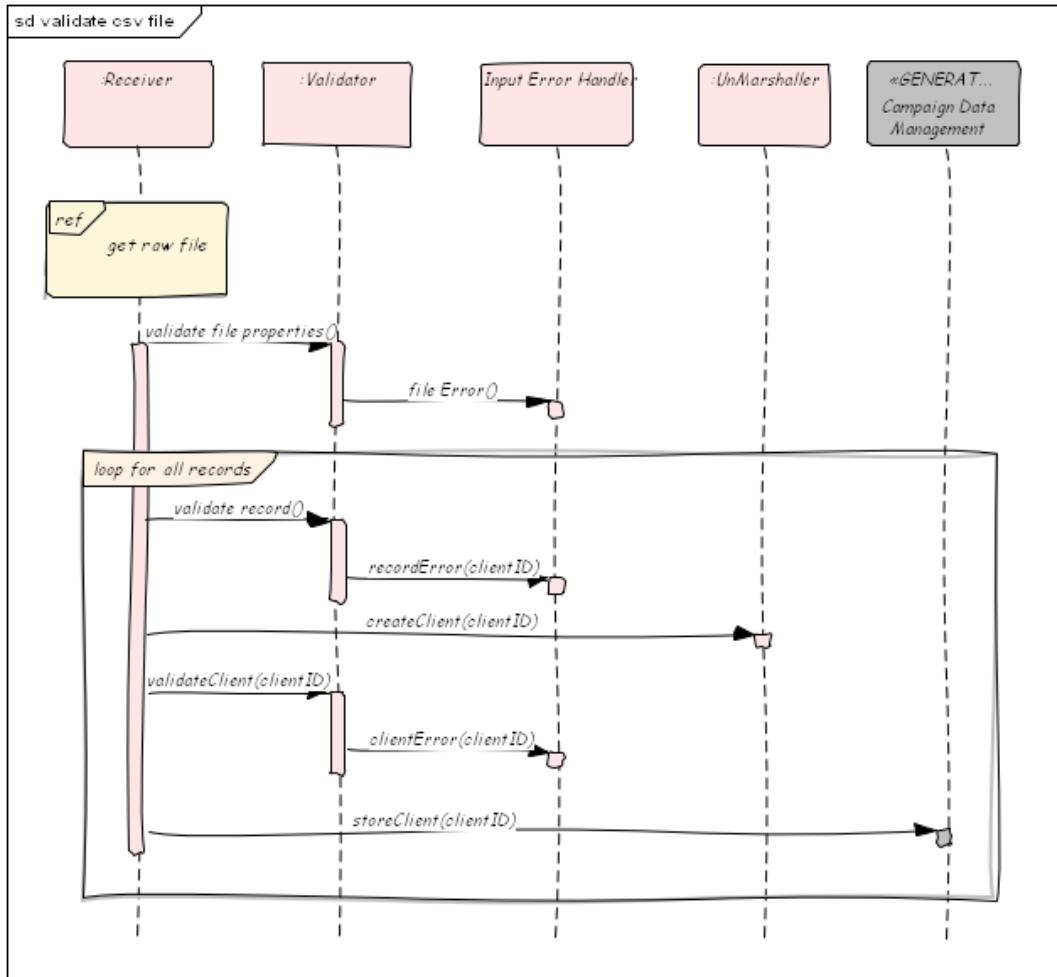
1. `tryImport`: `ProcessControl` starts the import. The `activity` is a unique ID identifying the mandator, the campaign and the activity.
2. `importConfiguration` gets all required configuration information
3. `configureReceiveChannel` prepares everything needed to get data from an external source. For example, URL, filenames and authentication credentials for an external ftp server need to be configured here.
4. `archive` sends the file to the (configured) archive system, usually an optical write-once non-erasable backup archive.
5. `setup` initializes the required filters, e.g. unzip or decrypt.
6. `filter` executes all the filters

The steps 5+6 are a dynamically configured pipes-and-filter dataflow subsystem. You find some more info in the [filter concept](#).

6.1.2 Validate File

Prerequisite: Data has been imported from external source, has been successfully filtered (i.e. decrypted and decompressed). See previous section (Import Raw).

The diagram below contains error handling. In *good cases* there will be no errors. Calls to `ImportErrorHandler` are only executed if errors occur!



(Second part of data import:) Validate imported data

III.7 Deployment View

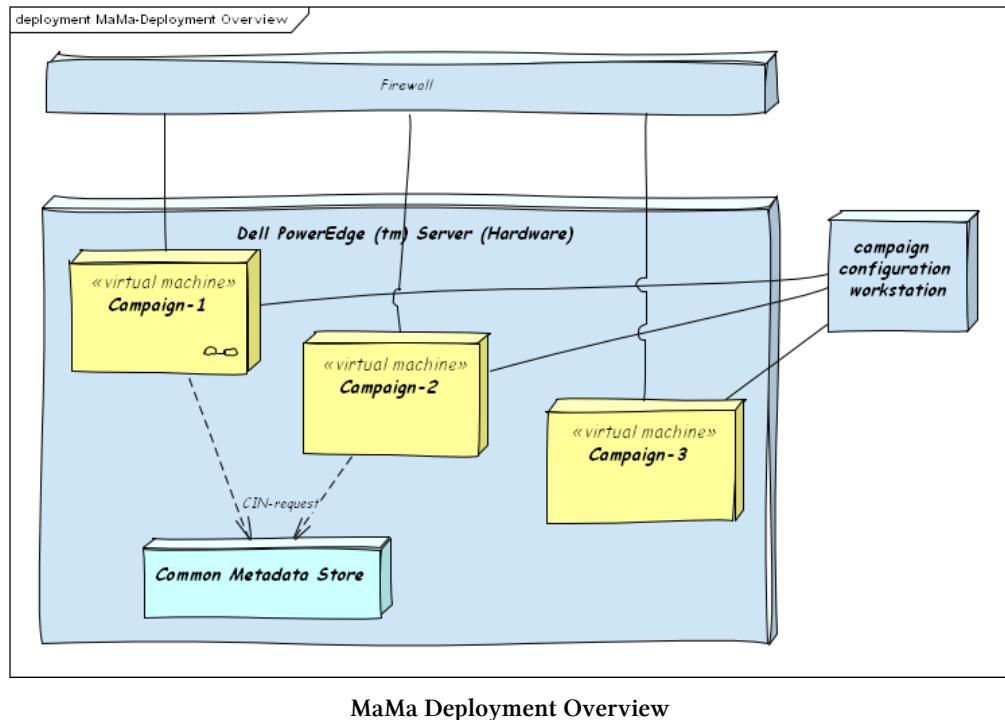
7.1 Deployment Overview

It was a longstanding goal of MaMa to deploy and operate each MaMa campaign on a dedicated virtual machine, to clearly separate mandator specific data from other instances.

The operating-system level configuration and operation mode of these virtual machines and their host machine directly influences the level of security the campaigns have. These topics have to be subject of regular security inspections and reviews.



Due to the sensitive nature of data handled by the original MaMa system the owner required strict nondisclosure in that aspect. Therefore we are not allowed to go into any detail of security.



Element	Description
Campaign-i	Virtual machine for one single campaign.
Common Metadata Store	Used only in eHealth campaigns to synchronize generation of CIN IDs (see below)
Campaign Configuration Workstation	Workstation (standard PC running Java-enabled OS) used to configure campaigns.
CIN request	Request for Common Insurance Number (see below)

7.2 Campaign Specific Virtual Machine

For every campaign operated by InDAC there will be a single dedicated virtual machine containing a database instance and all required MaMa code (except the graphical configuration UI).

7.3 Common Metadata Store (CoMeS)

The German government regulations for the eHealth card contained a very specific process to generate the “Common Insurance ID” (CID) for persons: This ID could only be generated by a single government entity (formerly the GPFunds, “Deutsche Rentenanstalt”, since 2012 the [ISTG³⁹](#)).

Requests for calculating the 10-digit CID have to be wrapped in a *request envelope* containing the following metadata:

- Unique ID of the requesting entity (usually the tax ID number of the organization/company issuing the request.) MaMa needed to use the tax ID of the InDAC data center.
- request purpose (for MaMa, a constant)
- request sequence number (RSN)

This RSN needed to be an uninterrupted sequence of numbers, as GPFunds wanted to make sure it did not miss any request. For MaMa that implied some synchronization mechanism between otherwise independent virtual machines. We decided to implement the Common Metadata Store for this reason.

For security reasons MaMa did not use a real database for this purpose, but this custom-build synchronization solution.

7.4 Campaign Configuration Machine

One (or several) operator workstations (standard PC's) will be used to configure MaMa instances after they have been physically deployed on their respective VMs.

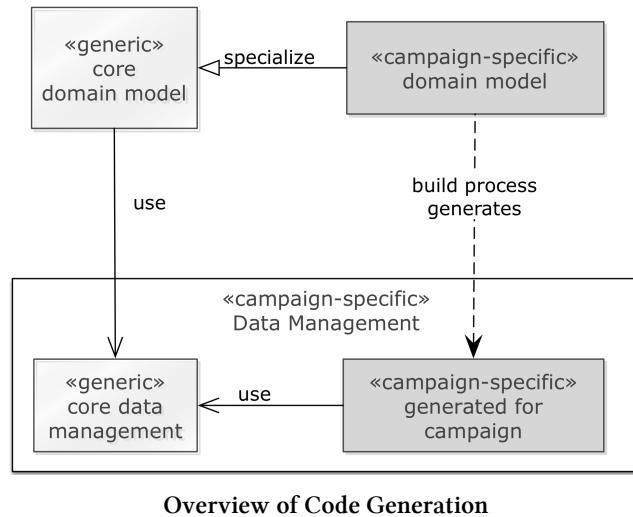
The configuration UI is build as Eclipse RCP plugin.

³⁹https://de.wikipedia.org/wiki/Informationstechnische_Servicestelle_der_gesetzlichen_Krankenversicherung

III.8 Cross-Cutting Concepts

8.1 Generated Persistence based upon Domain Model

MaMa uses a code generator to generate all (!) required persistence code from a UML entity model. The overall concept of this generation is depicted in the following diagram. Generic and campaign-specific parts are stereotyped in this diagram. The «campaign-specific» Data Management component is automatically created and packaged by the build process.



Element	Applicability	Description
Core Domain Model	«generic»	UML model containing the generic classes and relations every MaMa instance needs.
Core Domain Data Management	«campaign-specific»	Handwritten code, some Hibernate stuff, some generic finder and repository methods.
Specific Domain Model	«campaign-specific»	UML model enhancing the Core Domain Model.

Element	Applicability	Description
Data Management	«campaign-specific»	Java jar archive generated by the code generator specifically for a campaign. Name and ID of campaign is contained in metadata, so different instances of this element are distinguishable, e.g. for audit or revision purposes.
Code Generator	«generic»	not shown in diagram.

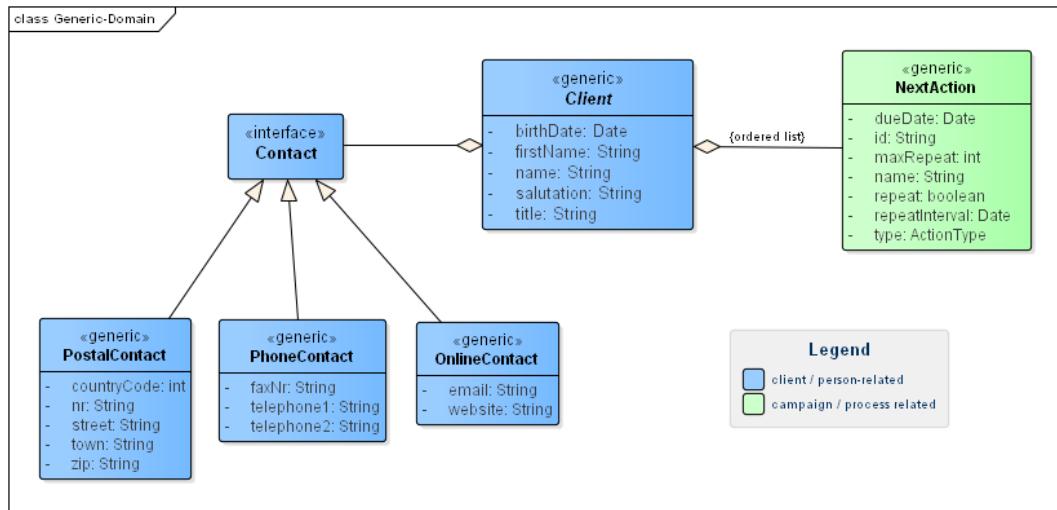
This concept relies on the following prerequisites:

Prerequisites:

- Every MaMa instance will handle data related to individual people - called *clients* in MaMa domain terminology.
- All clients will have a small number of common attributes.
- For all productive campaigns MaMa needs to handle an arbitrary number of additional attributes.
- Every mandator will *add* several campaign-specific attributes to the client, and/or will add campaign specific types (like insurance-contract or mobile-phone-contract)
- Once configured prior to campaign start, these campaign specific data structures will rarely change⁴⁰

⁴⁰In several years of MaMa operation, data structures within an active campaign always remained fix, therefore MaMa did never need any data migration utilities...

8.1.1 Generic Domain Model (“MaMa-Core-Domain”)

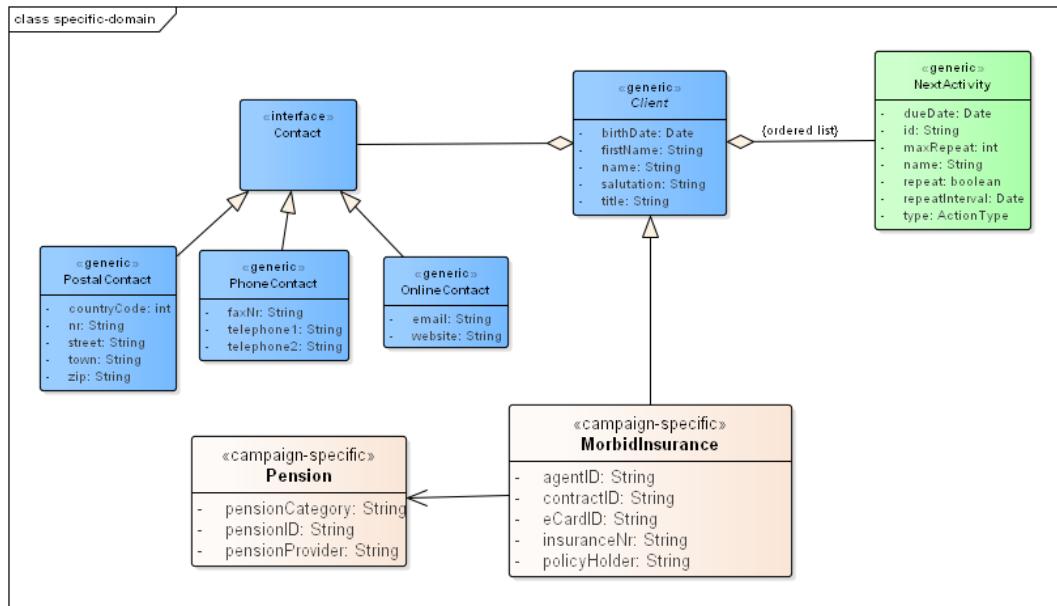


MaMa Generic Domain Model

Element	Description
Client	Abstract class, representing a person plus corresponding contact information.
Contact	Contact information that will be used for contacting the client instances during campaign execution.
Next Action	Generic class describing campaign activities. Central to the concept of campaign process control and business rule execution

8.1.2 Example for a Campaign Specific Domain Model

Specific campaign models always contain a (physical) copy of the complete core domain. The abstract Client class always need to be subclassed, and might be 1:n associated with additional classes.



Specific Comain Model (for hypothetic insurance campaign)

8.1.3 Generator Details

- MaMa uses OpenArchitectureWare to generate the complete persistence code.
- Generation relies on the open source Hibernate O/R mapping and persistence framework.
- The generator generated the following elements:
 - DDL code to create database, schema, tables and indices.
 - Hibernate mapping files
 - Campaign specific implementations of `findClientByXY`, `findNextActivity` and similar methods
 - A number of campaign- and configuration specific methods for reporting and monitoring
- m:n associations are not supported
- modifications of data structures are not possible when campaigns are already active.

Due to nondisclosure agreements with InDAC we cannot show example source code for the persistence concept.

Alternatives

- Initially MaMa had started with AndroMDA code generation framework, but that open source project lost popularity, could not deliver the required support and ceased working with newer Maven releases - so MaMa switched to OAW.
- MaMa uses the commercial MagicDraw UML (in version 9.0) modeling tool, which can in principle generate code based upon models, but proved to be too inflexible for the desired Hibernate integration. The contracting entity (InDAC) refused to upgrade to newer versions or alternative tools.

8.2 CSV Import / Export

- TODO: Describe (some) details of the CSV configuration DSL
- TODO: Describe (customized) Eclipse editor for these DSLs

8.3 Configurable File Filters

As explained in the runtime scenario “[import raw](#)”, every file read during import from an external source needs to be transformed via configurable filters. We most often have two kinds of filters, encryption and compression.

The names and required parameter settings for every filter is managed as part of the activity configuration within a campaign.

Encryption filters

Encryption filters are compliant to the Java Cryptography Architecture ([JCA⁴¹](#)), interfaces.

We urge mandators and partners to use crypto providers from the [BouncyCastle⁴²](#) portfolio. Encryption and decryption filters need credentials or certificates as part of their configuration.



Due to the sensitive nature of data handled by the original MaMa system the owner required strict nondisclosure in that aspect. Therefore we are not allowed to go into any detail of security.

⁴¹<https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html>

⁴²<https://www.bouncycastle.org/java.html>

Compression filter

MaMa supports only lossless compression algorithms. Compression can be configured to be either DEFLATE (as used in zip or gzip) or varieties of Lempel-Ziv compression. Compression filters have no parameters.

8.4 Rule Engine for Process and Flow Control

8.4.1 Flow of Action

8.4.2 Drools as Rule Engine

MaMa uses the open source rule engine [Drools⁴³](#) for definition, implementation and execution of business rules. Rules are defined as text files, which is interpreted at runtime by the rule engine. This enables modification and maintenance of rules without recompilation and redeployment of the whole system.

On the other hand, faulty rules can seriously hamper an active campaign – therefore modification of business rules shall always be thoroughly tested!

Rules always have a simple “when <A> then ” format, where <A> and are Java expressions.

You find a complete reference and many examples of the rule language at [Drools documentation home⁴⁴](#).

⁴³www.drools.org

⁴⁴<http://www.drools.org/learn/documentation.html>

III.9 Architecture Decisions

No Commercial CRM Tool

Do not use any of the commercial CRM tools as foundation for MaMa. The main reason was the incredible amount of flexibility required to quickly setup campaigns. This decision proved to be correct, as several early competitors (which operated upon slightly customized standard CRM tools) failed to enter the market MaMa operated in.

JBoss Drools for Rule Processing

Use JBoss Drools as rule processing engine. We evaluated Python (Jython) as an alternative, but that proved to be incredibly slow for our kind of processing.

No ETL Tool for Data import

Do not use an ETL tool for importing data. The contractor, InDACP, refused to consider the license fee of commercial ETL tools - therefore the development team had no chance to even evaluate those as data import solutions.

III.10 Quality Requirements

(for a brief overview of quality requirements, please see [section 1.2.2](#)).

Flexibility Scenarios

ID	Scenario
F1	New CSV import format shall be configurable at CCT within 2 hours.
F2	New fix-field import format shall be configurable at CCT within 2 hours.
F3	New XML based import format shall be configurable at CCT within 2 hours.
F4	New CSV export format shall be configurable at CCT within 2 hours.
F5	New fix-field export format shall be configurable at CCT within 2 hours.
F6	New XML based export format shall be configurable at CCT within 2 hours.

CCT: Campaign configuration time

In all cases we require both a documentation of the desired format, plus a minimum of 10 different test data records.

Runtime Performance Scenarios

ID	Scenario
P1	Import and fully process 250.000 scanned documents (including images) within 24hrs. That's an average processing rate of approximately 3 complete documents per second. Import format will be a combination of csv file plus images as single files.
P2	Import and fully process 100.000 records of csv file within 30 minutes

Security Scenarios

ID	Scenario
S1	Client and campaign data from one mandator shall never be accessible for another mandator.
S2	MaMa is required to preserve all incoming data from mandators and partners for the appropriate timeframe (usually 90-180 days after the end of a campaign). Such archived data (e.g. files or messages) needs to be made completely accessible for an auditor or inspection within 90 minutes at most.
S3	In case campaigns involve financial data of clients (e.g. credit card, bank account or similar information), these have to be processed and managed compliant to PCIDSS ⁴⁵ regulations.

⁴⁵https://en.wikipedia.org/wiki/Payment_Card_Industry_Data_Security_Standard

III.11 Risks

- The Receiver component suffers from overly complicated source code, created by a number of developers without consent. Since early days, most production bugs resulted from this part of MaMa-CRM.
- The runtime flexibility of import/export configurations and campaign processes might lead to incorrect and *undetected* behavior at runtime, as there are no configuration checks. Mischievous administrators can misconfigure any MaMa-CRM instance at any time.
- Configuration settings are not archived and therefore might get lost (so there might be no fallback to the last working configuration in case of trouble).
- The ‘Common-Metadata-Store’ is an overly trivial and resource-wasting synchronization mechanism and should be replaced with a decent async / event-based system asap.

III.12 Glossary

Term	Definition
Activity	Process step of campaign. For MaMa-CRM: either inbound, outbound or internal.
Activity, internal	Scheduled data maintenance activities, i.e. removing some data 90 days after its last usage. In Germany, data security law requires some kinds of data to be deleted after certain intervals.
Activity, inbound	Read data (i.e. files) delivered by either a ->partner or ->mandator.
Activity, outbound	Send data to either a partner or mandator.
Branch office	Business organization directly associated with mandator, serves a subset of one mandators' consumers.
Campaign	Coordinated set of activities, initiated by a ->mandator towards a potentially large number of ->clients. MaMa-CRM campaigns usually aim at acquiring certain kinds of data, like passport photographs, social security numbers, signatures etc.
CID	Common Insurance ID, 10-digit number to uniquely identify an health insured person in Germany. Created by a central agency.
Client	aka: Consumer. Single person or company with a business relationship to mandator. The end user of the services or products provided by the mandator.
CSV	Comma Separated Value
End user	Synonym for consumer.
InDAC	Contractee of MaMa-CRM, the data-center that initiated and payed for its development. Provides CRM campaign services for several mandators.

Term	Definition
Instance	MaMa-CRM is a family of systems, where a single instance is configured and operated for exactly one mandator and one or more campaigns.
Mandator	Organization responsible for a campaign. Conducts business in a mass-market domain like insurance, telecommunication, retail or energy.
Partner	<i>Category</i> name for service providers: Organization or enterprise providing services for MaMa, like printing, scanning, phone and call-center, mail delivery, and so forth.
PCIDSS	Payment Card Industry Data Security Standard⁴⁶ , a collection of proprietary standards for secure handling and management of credit card related data.
RSN	Request sequence number, 0-padded 8-digit string containing the sequence number of CID requests.

⁴⁶https://en.wikipedia.org/wiki/Payment_Card_Industry_Data_Security_Standard

IV - biking2

By [Michael Simons](#).

biking2 or “Michis milage” is primarily a web based application for tracking biking related activities. It allows the user to track the covered milage, collect GPS tracks of routes, convert them to different formats, track the location of the user and publish pictures of bike tours.

The secondary goal if this application is to have a topic to experiment with various technologies, for example Spring Boot on the server side and AngularJS, JavaFX and others on the client side.

As such biking2 has been around since 2009 and in it’s current Java based form since 2014. Defining production as full filling the primary goal it’s been in production ever since.

The project is published under *Apache License* on [GitHub](#)⁴⁷, use it however you like. Though I’ve been blogging regularly about this pet project, the documentation in its current form was created after I met Gernot and Peter at an awesome workshop in Munich.

⁴⁷<https://github.com/michael-simons/biking2>

IV.1 Introduction and Goals

1.1 Requirements Overview

What is biking2?

The main purpose of *biking2* is keeping track of bicycles and their milages as well as converting *Garmin Training Center XML (tcx)*⁴⁸ files to standard *GPS Exchange Format (GPX)*⁴⁹ files and storing them in an accessible way.

In addition *biking2* is used to study and evaluate technology, patterns and frameworks. The functional requirements are simple enough to leave enough room for concentration on quality goals.

Main features

- Store bikes and their milages
- Convert tcx files to GPX files and provide them in a library of tracks
- Visualize those tracks on a map and provide a way to embed them in other webpages
- Visualize biking activities with images
- Optional, near real time, tracking of a biker

The application must only handle exactly one user with write permissions.

Most bike aficionados have problems understanding the question “why more than one bike?”, the system should be able to keep track of everything between 2 and 10 bikes for one user, storing 1 total milage per bike and month. All milages per month, year and other metrics should be derived from this running total, so that the user only need to look at his odometer and enter the value.

The application should store an “unlimited” number of tracks.

The images should be collected from *Daily Fratze*⁵⁰, the source are all images that are tagged with “Radtour”. In addition the user should be able to provide an “unlimited” number of “gallery pictures” together with a date and a short description.

⁴⁸https://en.wikipedia.org/wiki/Training_Center_XML

⁴⁹https://en.wikipedia.org/wiki/GPS_Exchange_Format

⁵⁰<https://dailyfratze.de>

1.2 Quality Goals

#	Quality	Motivation
1	Understandability	The functional requirements are simple enough to allow a simple, understandable solution that allows focus on learning about new Java 8 features, Spring Boot and AngularJS.
2	Efficiency	Collecting milage data should be a no brainer: Reading the milage from an odometer and entering it.
3	Interoperability	The application should provide a simple API that allows access to new clients.
4	Attractiveness	Collected milages should be presented in easy to grasp charts.
5	Testability	The architecture should allow easy testing of all main building blocks.

1.3. Stakeholders

The following lists contains the most important personas for this application:

Role/Name	Goal/Boundaries
Developers	Developers who want to learn about developing modern applications with Spring Boot and various frontends, preferable using <i>Java 8</i> in the backend.
Bikers	Bike aficionados that are looking for a non-excel, self-hosted solution to keep track of their bikes and milages.
Software Architects	Looking for an <i>arc42</i> example; want to get ideas for their daily work.
Michael Simons	Improving his skills; wants to blog about Spring Boot; looking for a topic he can eventually hold a talk about; needed a project to try out new <i>Java 8</i> features.

IV.2 Constraints

The few constraints on this project are reflected in the final solution. This section shows them and if applicable, their motivation.

2.1 Technical Constraints

Software and programming constraints

Constraint	Background
TC1 Implementation in Java	The application should be part of a Java 8 and Spring Boot show case. The interface (i.e. the api) should be language and framework agnostic, however. It should be possible that clients can be implemented using various frameworks and languages.
TC2 Third party software must be available under an compatible open source license and installable via a package manager	The interested developer or architect should be able to check out the sources, compile and run the application without problems compiling or installing dependencies. All external dependencies should be available via the package manager of the operation system or at least through an installer.

Operating System Constraints

Constraint	Background
TC3 OS independent development	The application should be compilable on all 3 mayor operation systems (Mac OS X, Linux and Windows)
TC4 Deployable to a Linux server	The application should be deployable through standard means on a Linux based server

Hardware Constraints

Constraint	Background
TC5 Memory friendly	Memory can be limited (due to availability on a shared host or deployment to cloud based host). If deployed to a cloud based solution, every megabyte of memory costs.

2.2 Organizational Constraints

Constraint	Background
OC1 Team	Michael Simons
OC2 Time schedule	Start in early 2014 with Spring Boot beta based prototypes running on Java 8 early access builds, first “release” version March 2014 together with the initial release of Java 8. Upgrade to a final Spring Boot release when they are available.
OC3 IDE independent project setup	No need to continue the editor and IDE wars. The project must be compilable on the command line via standard build tools. Due to OC2 there is only one IDE supporting Java 8 features out of the box: <i>NetBeans 8</i> beta and release candidates.
OC4 Configuration and version control / management	Private git repository with a complete commit history and a public master branch pushed to GitHub and linked a project blog.
OC5 Testing	Use JUnit to prove functional correctness and integration tests and JaCoCo to ensure a high test coverage (at least 90%).
OC6 Published under an Open Source license	The source, including documentation, should be published as Open Source under the Apache 2 License.

2.3 Conventions

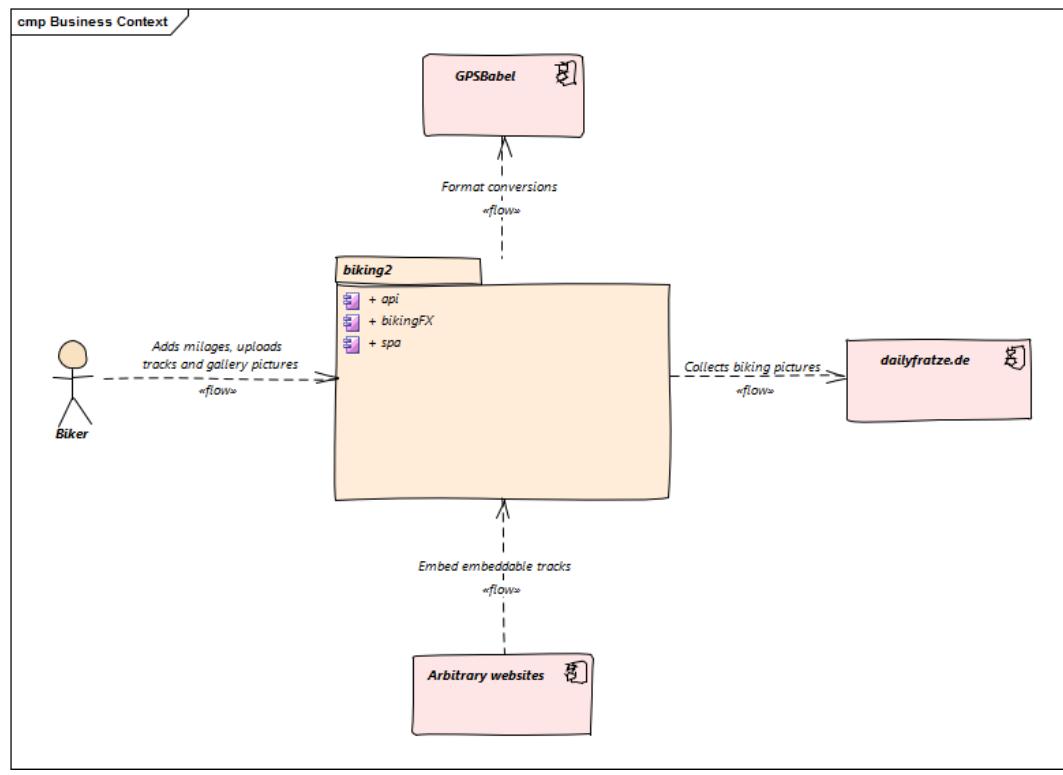
Conventions	Background
C1 Architecture documentation	Structure based on the english arc42-Template in version 6.5
C2 Coding conventions	The project uses the Code Conventions for the Java TM Programming Language⁵¹ . The conventions are enforced through Checkstyle .
C3 Language	English. The project and the corresponding blog targets an international audience, so English should be used throughout the whole project.

⁵¹<https://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

IV.3 System Scope and Context

This chapter describes the environment and context of *biking2*: Who uses the system and on which other system does *biking2* depend.

3.1 Business Context



Business Context

Biker

A passionate biker uses *biking2* to manage his bikes, milages, tracks and also visual memories (aka images) taken on tours etc. He also wants to embed his tracks as interactive maps on other websites.

Daily Fratze

[Daily Fratze⁵²](#) provides a list of images tagged with certain topics. *biking2* should collect all images for a given user tagged with “Theme/Radtour”.

GPSBabel

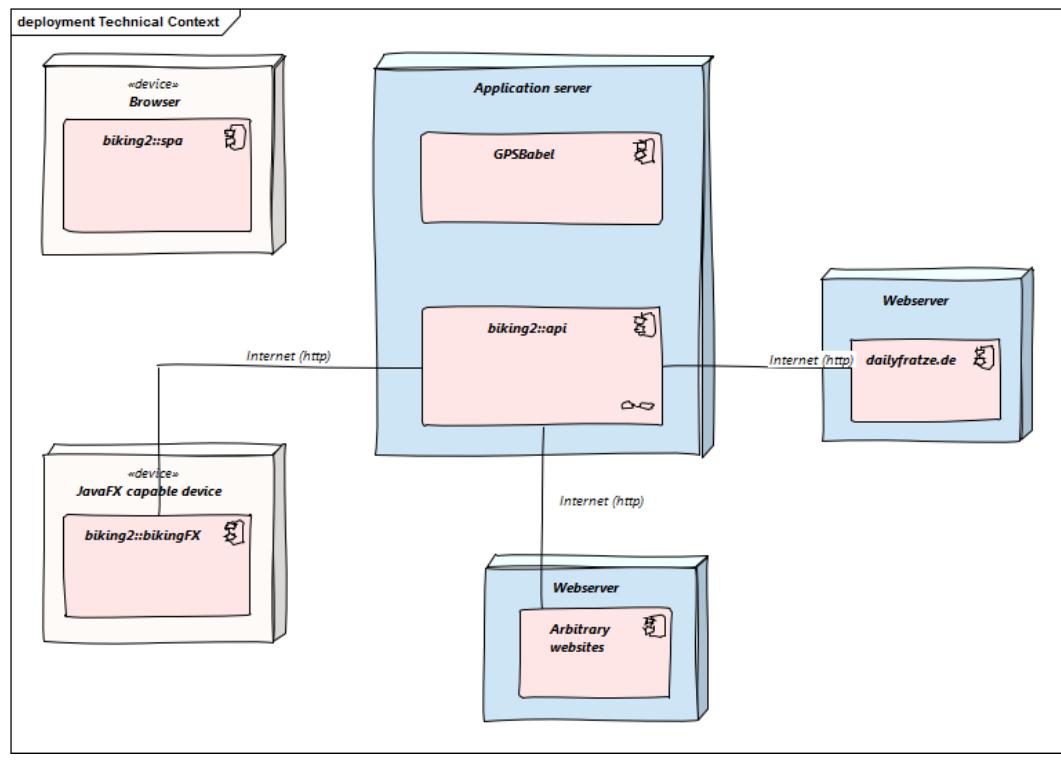
GPSBabel is a command line utility for manipulating GPS related files in various ways. *biking2* uses it to convert TCX into GPX files. The heaving lifting is done by GPSBabel and the resulting file will be managed by *biking2*.

Arbitrary websites

The user may want to embed (or brag with) tracks on arbitrary websites. He only wants to paste a link to a track on a website that supports embedded content to embed a map with the given track.

⁵²<https://dailyfratze.de>

3.2 Technical Context



Technical Context

biking2 is broken into 2 main components:

Backend (**biking2::api**)

The api runs on a supported application server, using either an embedded container or an external container. It communicates via operating system processes with GPSBabel on the same server.

The connection to *Daily Fratze* is http based RSS-feed. The feed is paginated and provides *all* images with a given tag but older images may not be available any more when the owner decided to add a digital expiry.

Furthermore *biking2* provides an oEmbed⁵³ interface for all tracks stored in the

⁵³<https://oembed.com>

system. Arbitrary websites supporting that protocol can request embeddable content over http knowing only a link to the track without working on the track or map apis themselves.

Frontend (biking2::spa and biking2::bikingFX)

The frontend is implemented with two different components, the biking2::spa (Single Page Application) is part of this package. The spa runs in any modern web browser and communicates via http with the api.

Business interface	channel
Format conversions	System processes, command line interface
Collection of biking pictures	RSS feed over Internet (http)
Embeddable content	oEmbed format over Internet (http)
API for business functions	Internet (http)

IV.4 Solution Strategy

At the core of *biking2* is a simple yet powerful domain model based on a few entities of which a “Bike” and it’s “Milage” are the most important.

Although data centric, the application resigns from using too much SQL for creating reports, summaries and such but tries to achieve that with new Java 8 features around streams, lambdas and map/reduce functions.

Building the application with Spring Boot is an obvious choice as one of the main [quality goals](#) is learning about it. But furthermore using Spring Boot as a “framework” for Spring Framework allows concentration on the business logic. On the one hand there is no need to understand a complex XML configuration and on the other hand all building blocks are still put together using dependency injection.

Regarding dependency injection and testability: All injection should be done via constructor injection, setter injection is only allowed when there’s no other technical way. This hopefully prevents centralized “god classes” that control pretty much every other aspect of the application.

Spring Boot applications can be packaged as single, “fat jar” files. Since Spring Boot 1.3 those files contain a startup script and can be directly linked to /etc/init.d on a standard Linux systems which serves [TC4](#).

Interoperability will be achieved by using JSON over simple http protocol for the main API. Security is not the main focus of this application. It should be secure enough to prevent others from tempering with the data, confidentiality is not a main concern (read: passwords can be transmitted in plain over http).

The internal single page application shall be implemented using AngularJS. The deployable artifact will contain this application so there is no need for hosting a second webserver for the static files.

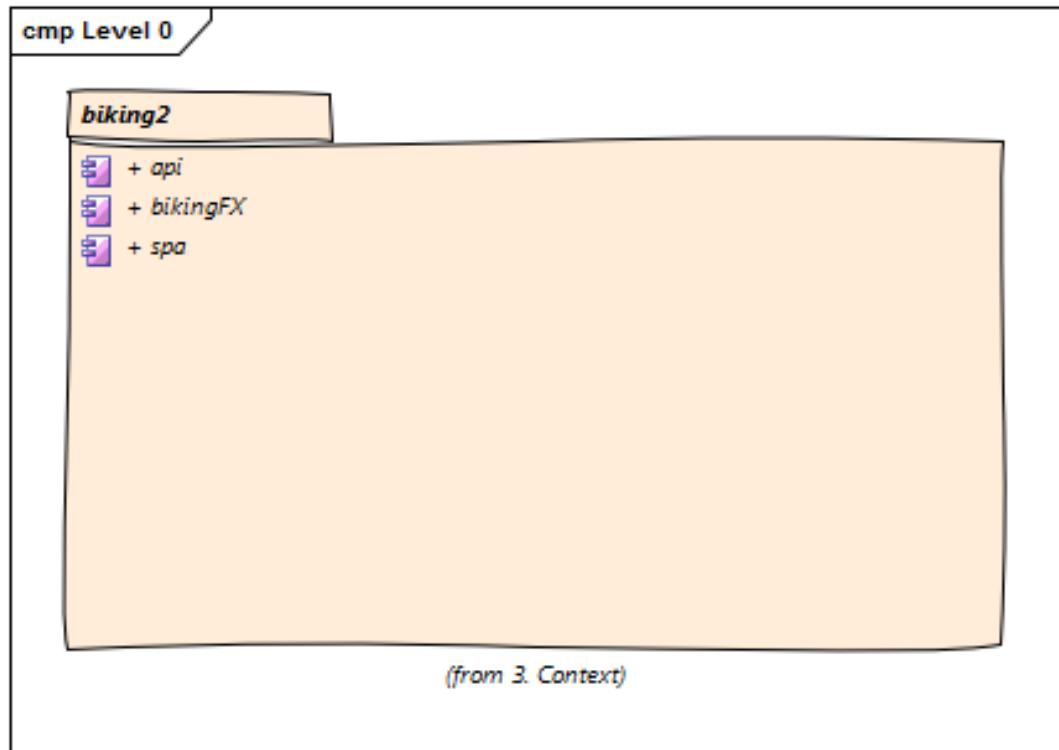
For real time tracking the MQTT protocol will be used which is part of Apache ActiveMQ, supported out of the box by Spring Messaging.

Graphing should not be implemented here but instead the [Highcharts⁵⁴](#) library should be used. The configuration for all charts should be computed server side.

⁵⁴<https://www.highcharts.com>

IV.5 Building Block View

The application packaged as *biking2.jar* contains two (the api and the spa) of three main parts, as shown in the [Business Context](#):



Building Blocks Level 0

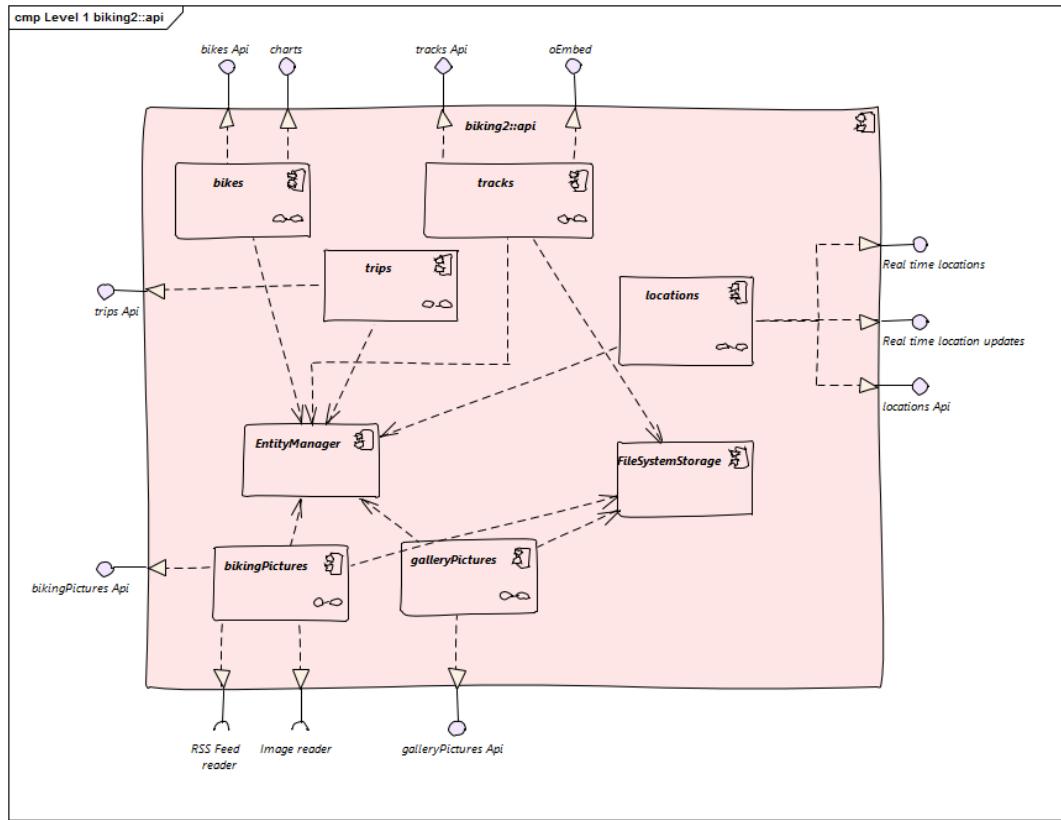
From those two we have a closer look at the api only. For details regarding the structure of an AngularJS 1.2.x application, have a look at their [developers guide](#)⁵⁵.

NOTE: To comply with the Java coding style guidelines, the modules “bikingPictures” and “galleryPictures” reside in the Java packages “bikingpictures” and “gallerypictures”.

⁵⁵<https://code.angularjs.org/1.2.28/docs/guide>

5.1 Whitebox biking2::api

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



Building Blocks Level 1

I used *functional decomposition* to separate responsibilities. The single parts of the api are all encapsulated in their own components, represented as Java packages.

All components depend on a standard JPA EntityManager and some on local file storage. I won't go into detail for those blackboxes.

Contained blackboxes

Blackbox	Purpose
bikes (Blackbox)	Managing bikes, adding monthly milages, computing statistics and generating charts.
tracks (Blackbox)	Uploading tracks (TCX files), converting to GPX, providing an oEmbed interface.
trips (Blackbox)	Managing assorted trips.
locations (Blackbox)	MQTT and STOMP interface for creating new locations and providing them in real time on websockets via stomp.
bikingPictures (Blackbox)	Reading biking pictures from an RSS feed provided by <i>Daily Fratze</i> and providing an API to them.
galleryPictures (Blackbox)	Uploading and managing arbitrary pictures

Interfaces

Interface	Description
bikes Api	REST api containing methods for reading, adding and decommissioning bikes and for adding milages to single bikes.
charts	Methods for retrieving statistics as fully setup chart definitions.
tracks Api	REST api for uploading and reading TCX files.
trips Api	REST api for adding new trips.
oEmbed	HTTP based oEmbed interface, generating URLs with embeddable content.
Real time locations	WebSocket / STOMP based interface on which new locations are published.
Real time location updates	MQTT interface to which MQTT compatible systems like OwnTracks ⁵⁶ can offer location updates.

⁵⁶<https://owntracks.org>

Interface	Description
RSS feed reader	Needs an <i>Daily Fratze</i> OAuth token for accessing a RSS feed containing biking pictures which are than grabbed from <i>Daily Fratze</i> .
galleryPictures Api	REST api for uploading and reading arbitrary image files (pictures related to biking).

5.1.1 bikes (Blackbox)

Intent/Responsibility

bikes provides the external API for reading, creating and manipulating bikes and their milages as well as computing statistics and generating charts.

Interfaces

Interface	Description
REST interface /api/bikes/*	Contains all methods for manipulating bikes and their milages.
REST interface /api/charts/*	Contains all methods for generating charts.

Files

The bikes module and all of its dependencies are contained inside the Java package ac.simons.biking2.bikes.

5.1.2 tracks (Blackbox)

Intent/Responsibility

tracks manages file uploads (TCX files), converts them to GPX files and computes their surrounding rectangle (envelope) using *GPSBabel*. It also provides the *oEmbed* interface that resolves URLs to embeddable tracks.

Interfaces

Interface	Description
REST interface /api/tracks/*	Contains all methods for manipulating tracks.
/api/oembed	Resolve track URLs to embeddable tracks (content).
/tracks/*	Embeddable track content.

Files

The `tracks` module and all of its dependencies are contained inside the Java package `ac.simons.biking2.tracks`.

5.1.3 trips (Blackbox)

Intent/Responsibility

`trips` manages distances that have been covered on single days without relationships to bikes.

Interfaces

Interface	Description
REST interface /api/trips/*	Contains all methods for manipulating trips.

Files

The `trips` module and all of its dependencies are contained inside the Java package `ac.simons.biking2.trips`.

5.1.4 locations (Blackbox)

Intent/Responsibility

`locations` stores locations with timestamps in near realtime and provides access to locations for the last 30 minutes.

Interfaces

Interface	Description
REST interface /api/locations/*	For retrieving all locations in the last 30 minutes.
WebSocket / STOMP topic /topic/currentLocation	Interface for getting notification on new locations.
MQTT interface	Listens for new locations coming in via MQTT in OwnTracks format ⁵⁷

Files

The locations module and all of its dependencies are contained inside the Java package ac.simons.biking2.tracker. The module is configured through the class ac.simons.biking2.config.TrackerConfig.

5.1.5 bikingPictures (Blackbox)

Intent/Responsibility

bikingPictures is used for regularly checking a RSS feed from *Daily Fratze* collecting new images and storing them locally. It also provides an API for getting all collected images.

Interfaces

Interface	Description
RSS Feed reader	Provides access to the <i>Daily Fratze</i> RSS Feed.
Image reader	Provides access to images hosted on <i>Daily Fratze</i> .
REST interface /api/bikingPictures/*	Contains all methods for accessing biking pictures

⁵⁷<https://owntracks.org/booklet/tech/json/>

Files

The `bikingPictures` module and all of its dependencies are contained inside the Java package
`ac.simons.biking2.bikingpictures.`

5.1.6 galleryPictures (Blackbox)

Intent/Responsibility

`galleryPictures` manages file uploads (images). It stores them locally and provides an RSS interface for getting metadata and image data.

Interfaces

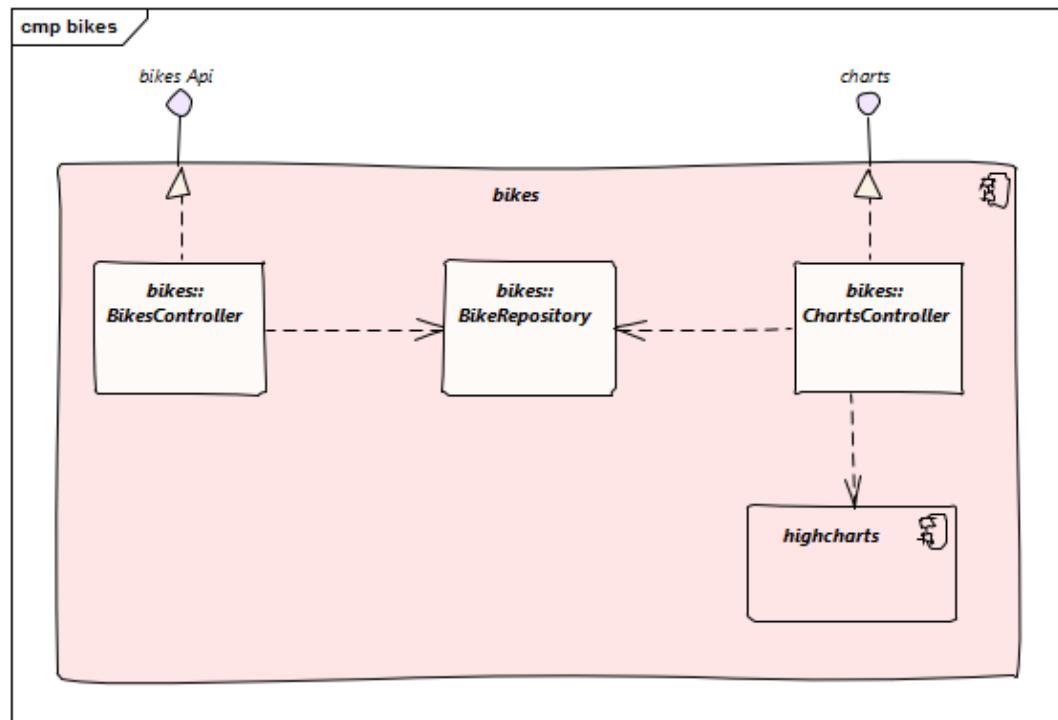
Interface	Description
REST interface <code>/api/galleryPictures/*</code>	Contains all methods for adding and reading arbitrary pictures.

Files

The `galleryPictures` module and all of its dependencies are contained inside the Java package `ac.simons.biking2.gallerypictures.`

5.2 Building Blocks - Level 2

5.2.1 bikes (Whitebox)



Building block: bikes

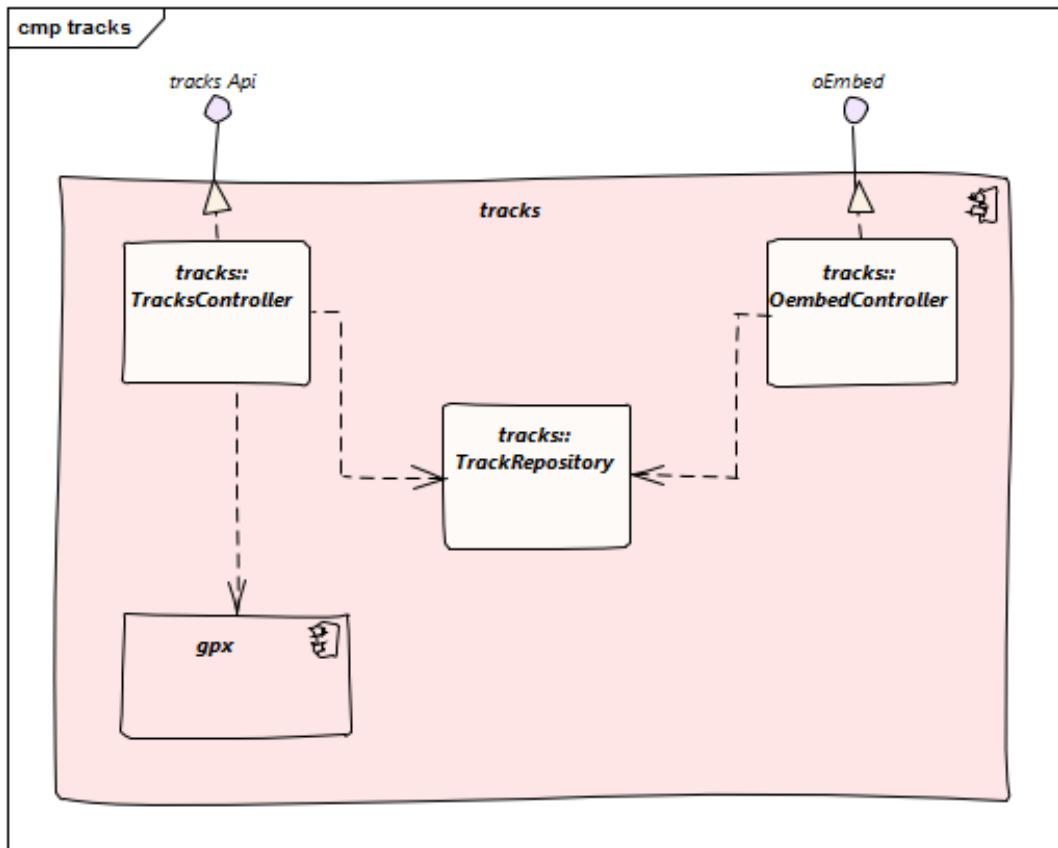
The BikeRepository is a Spring Data JPA based repository for BikeEntities. The BikeController and the ChartsController access it to retrieve and store instances of BikeEntity and provide external interfaces.

Contained blackboxes

Blackbox	Purpose
highcharts	Contains logic for generating configurations and definitions for Highcharts ⁵⁸ on the server side.

⁵⁸<https://www.highcharts.com>

5.2.2 tracks (Whitebox)



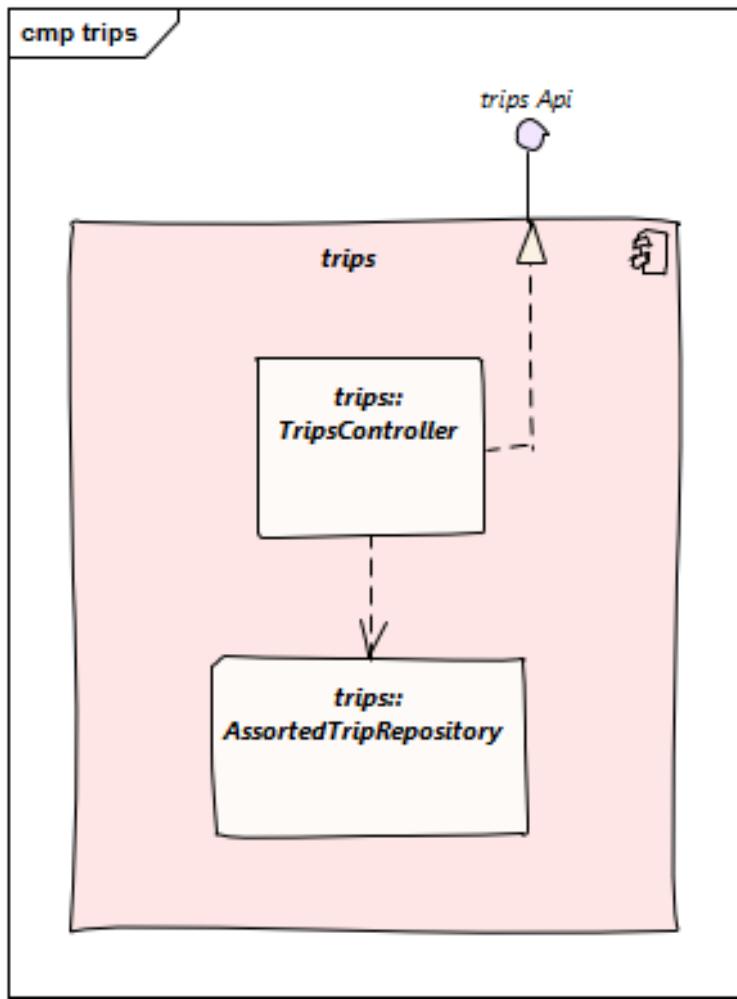
Building block: tracks

The `TrackRepository` is a Spring Data JPA based repository for `TrackEntities`. The `TracksController` and the `OembedController` access it to retrieve and store instances of `TrackEntity` and provide external interfaces.

Contained blackboxes

Blackbox	Purpose
gpx	Generated JAXB classes for parsing GPX files. Used by the TracksController to retrieve the surrounding rectangle (envelope) for new tracks.

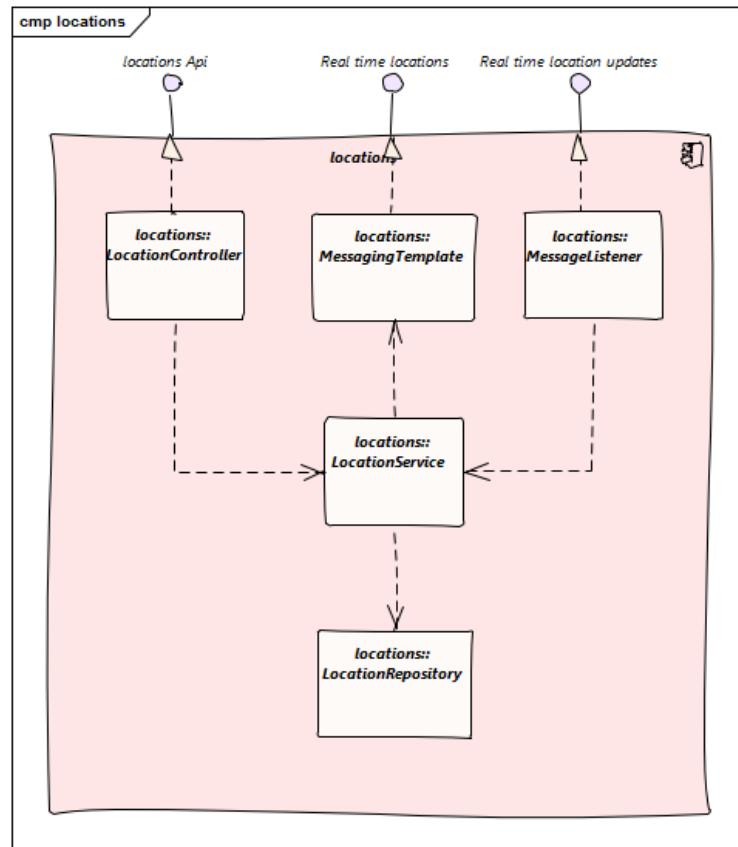
5.2.3 trips (Whitebox)



Building block: trips

The AssortedTripRepository is a Spring Data JPA based repository for AssortedTripEntities. The TripsController accesses it to retrieve and store instances of TrackEntity and provide external interfaces.

5.2.4 locations (Whitebox)

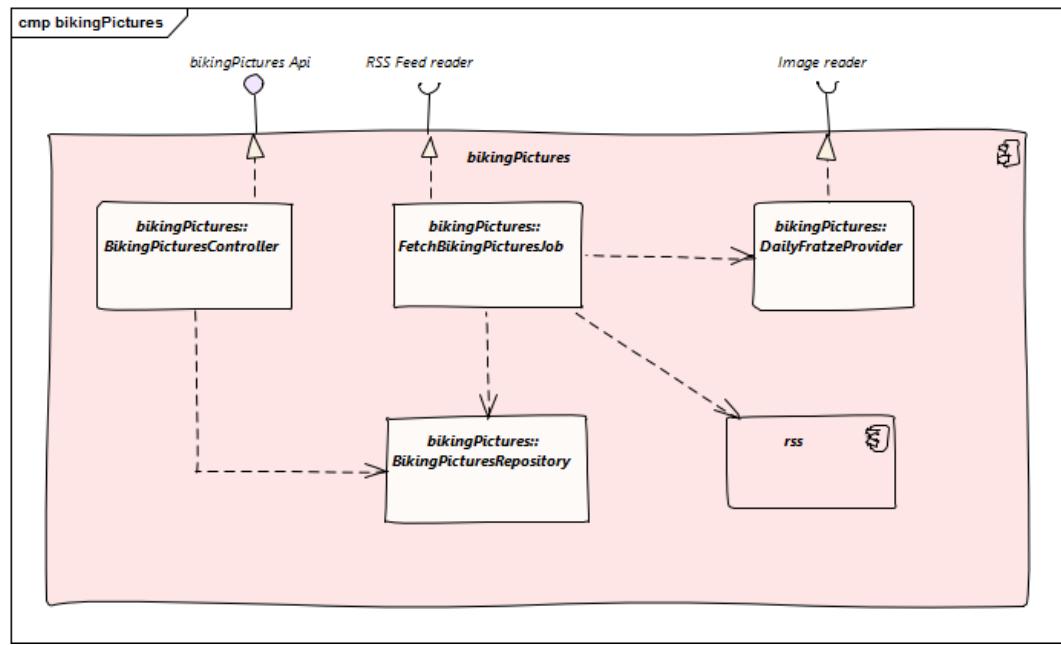


Building block: locations

Locations are stored and read via a Spring Data JPA based repository named LocationRepository. This repository is only accessed through the LocationService. The LocationService provides real time updates for connected clients through a SimpMessagingTemplate and the LocationController uses the service to provide access to all locations created within the last 30 minutes.

New locations are created by the service either through a REST interface in form of the LocationController or via a MessageListener on a MQTT channel.

5.2.5 bikingPictures (Whitebox)



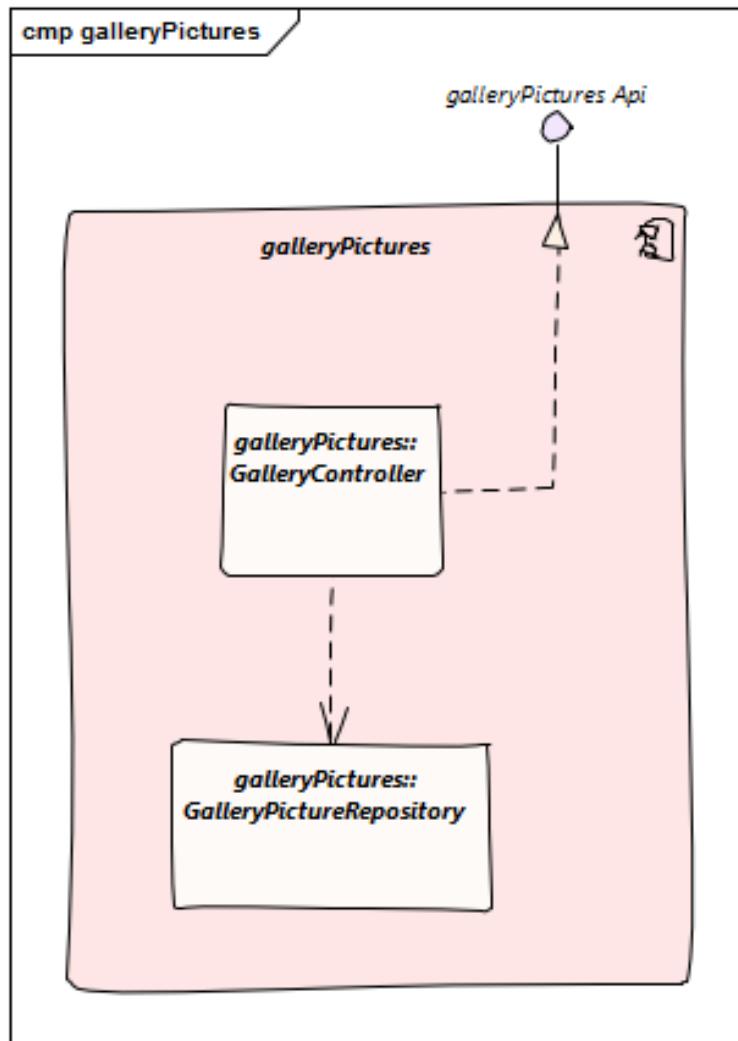
Building block: `bikingPictures`

A Spring Data JPA repository named `BikingPicturesRepository` is used for all access to `BikingPictureEntities`, the external REST api for reading pictures is implemented with `BikingPicturesController`. The RSS feed is read from `FetchBikingPicturesJob` by using a JAXBContext “rss”. The URLs to the image files which may be protected by various means are provided to the job via a `DailyFratzeProvider`.

Contained blackboxes

Blackbox	Purpose
rss	Generated JAXB classes for parsing RSS feeds. Used by the <code>FetchBikingPicturesJob</code> to read the contents of an RSS feed.

5.2.6 galleryPictures (Whitebox)



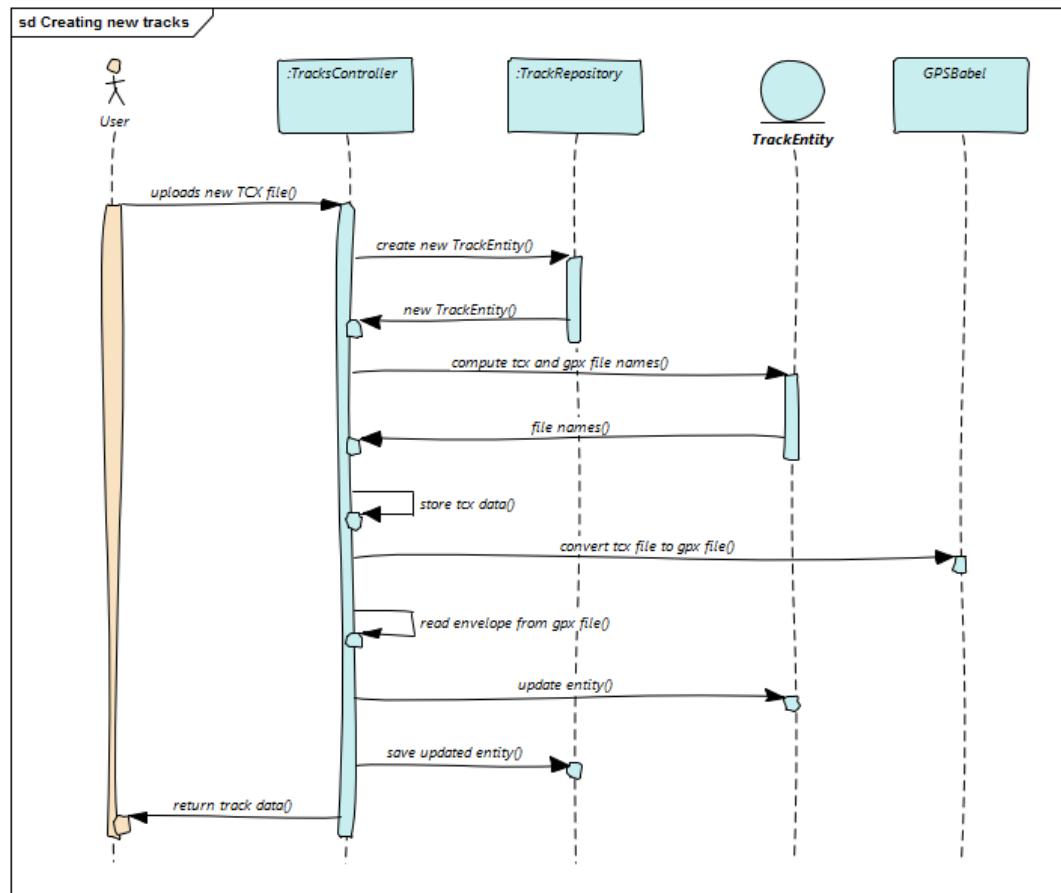
Building block: `galleryPictures`

The `GalleryPictureRepository` is a Spring Data JPA based repository. It is responsible for `GalleryPictureEntities`. The `GalleryController` accesses it to retrieve and store instances of `GalleryPictureEntity` and provide external interfaces.

IV.6 Runtime View

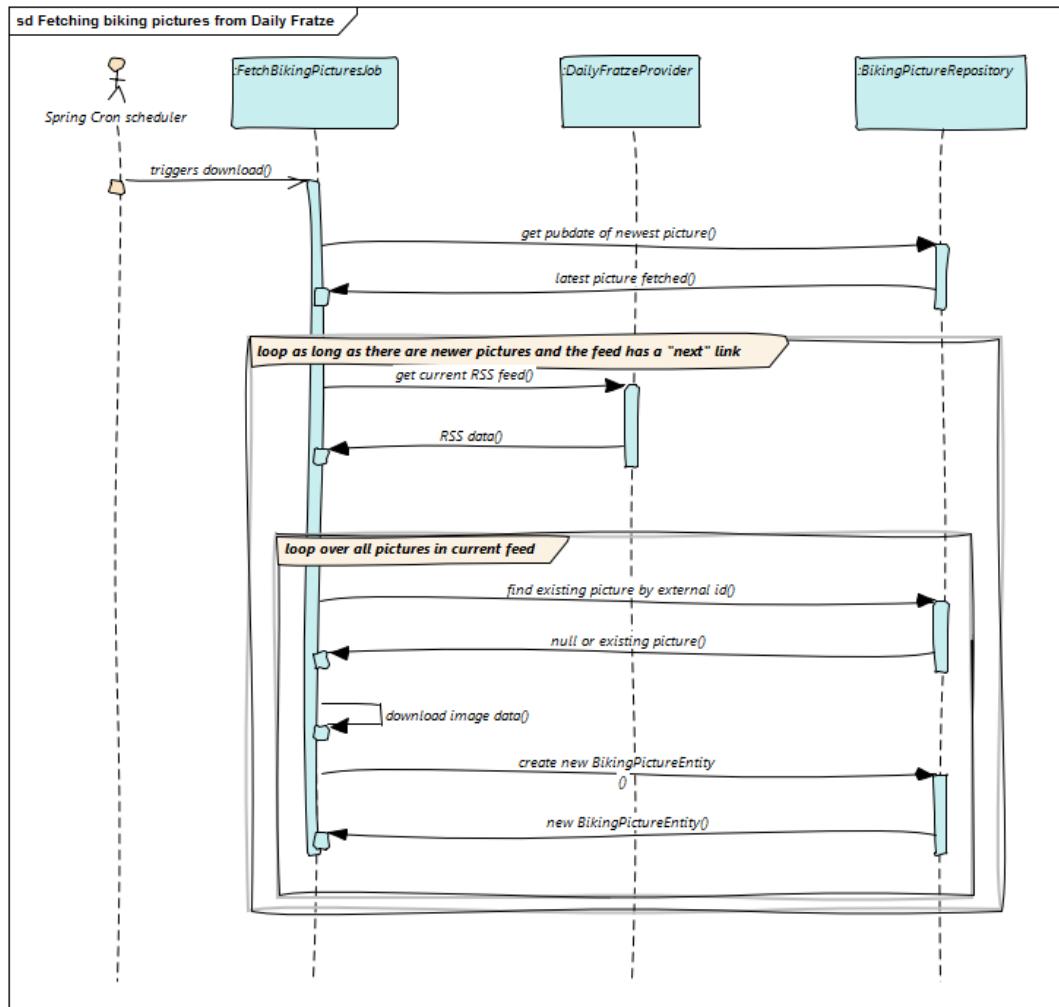
User interaction with *biking2* and error handling is pretty basic and simple. I picked up two uses cases where an actual runtime view is interesting:

Creating new tracks



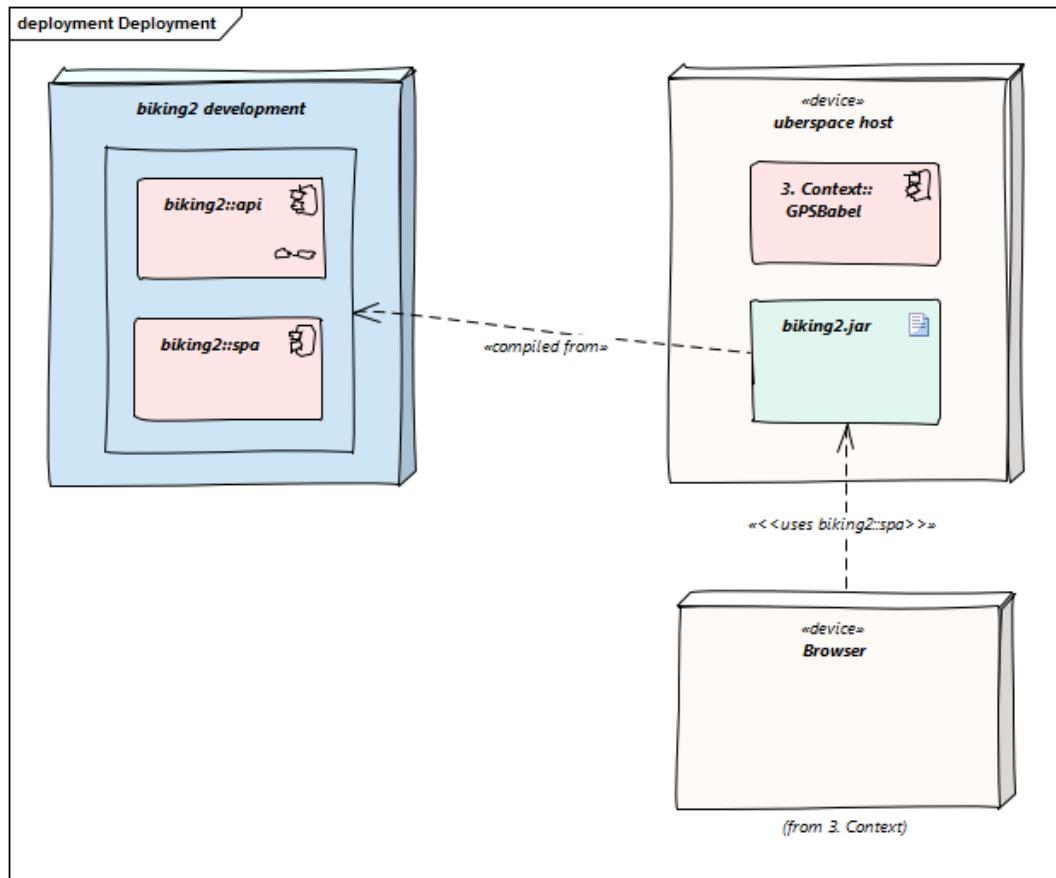
Creating new tracks

Fetching biking pictures from *Daily Fratze*



Fetching biking pictures

IV.7 Deployment View



Deployment View

Node / artifact	Description
biking2 development	Where <i>biking2</i> development takes place, standard computer with JDK 8, Maven and GPSBabel installed.
uberspace host	A host on Uberspace ⁵⁹ where <i>biking2.jar</i> runs inside a Server JRE ⁶⁰ with restricted memory usage.
biking2.jar	A “fat jar” containing all Java dependencies and a loader so that the Jar is runnable either as jar file or as a service script (on Linux hosts).
Browser	A recent browser to access the AngularJS <i>biking2</i> single page application. All major browsers (Chrome, Firefox, Safari, IE / Edge) should work.

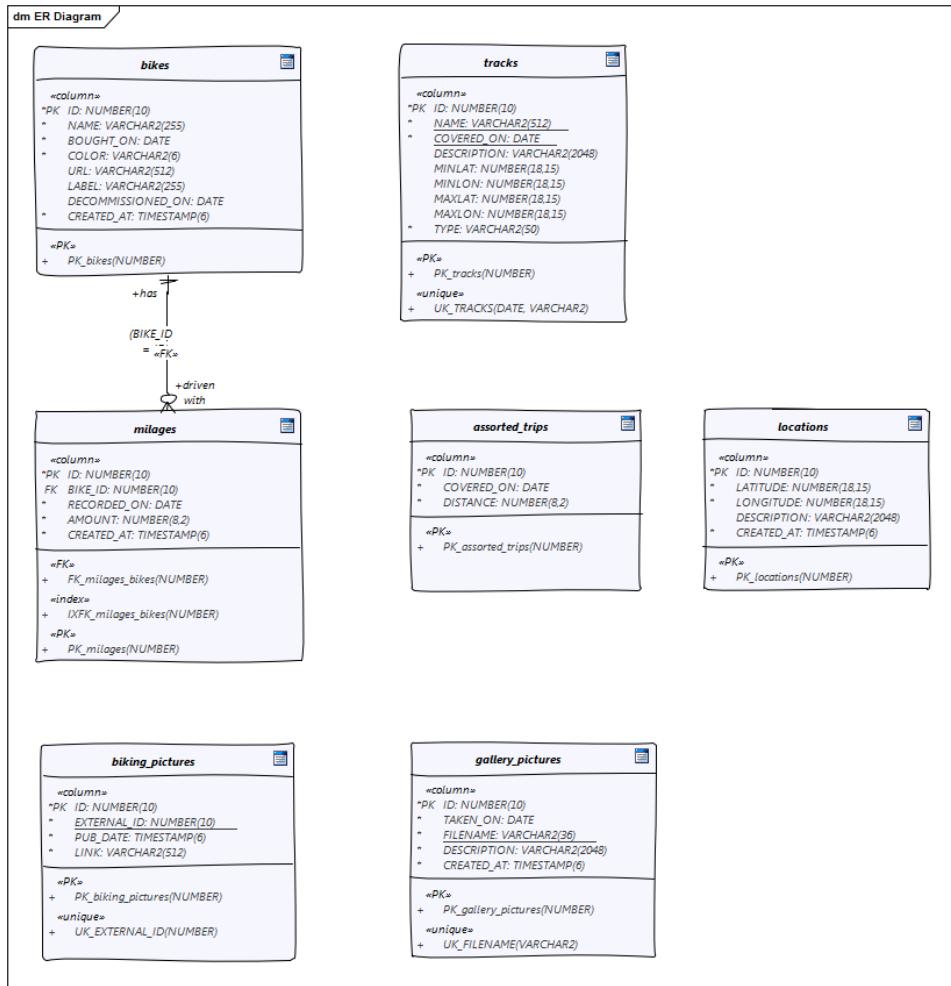
⁵⁹<https://uberspace.de>

⁶⁰<https://www.oracle.com/technetwork/java/javase/downloads/server-jre8-downloads-2133154.html>

IV.8 Cross-cutting Concepts

Domain Models

biking2 is a datacentric application, therefore everything is based around those entities that manifest as tables.



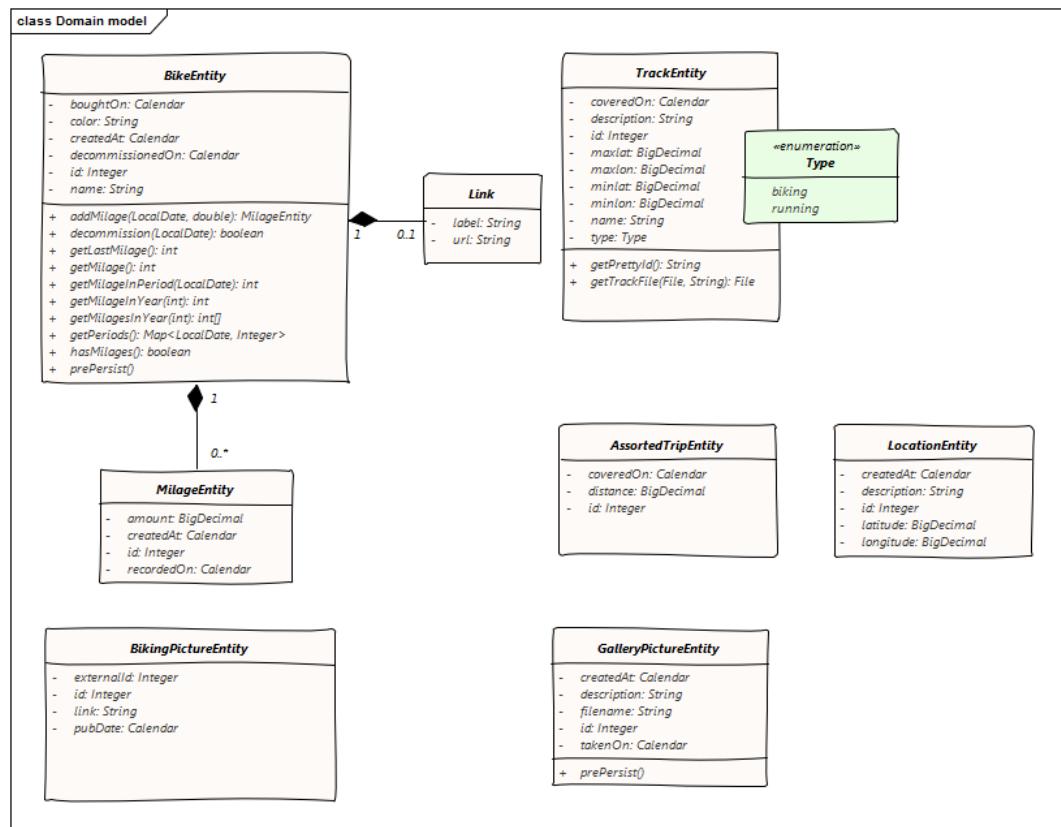
Entities in *biking2*

Tables

Name	Description
bikes	Stores the bikes. Contains dates when the bike was bought and decommissioned, an optional link, color for the charts and also an auditing column when a row was created.
milages	Stores milages for a bike (when and how much).
tracks	Stores GPS tracks recorded and uploaded with an optional description. For each day the track names must be unique. The columns <i>minlat</i> , <i>minlon</i> , <i>maxlat</i> and <i>maxlon</i> store the encapsulating rectangle for the track. The <i>type</i> column is constrained to “biking” and “running”.
assorted_trips	Stores a date and a distance on that day. Multiple distances per day are allowed.
locations	Stores arbitrary locations (latitude and longitude based) for given timestamp with an optional description.
biking_pictures	Stores pictures collected from <i>Daily Fratze</i> together with their original date of publication, their unique external id and a link to the page the picture originally appeared.
gallery_pictures	Stores all pictures uploaded by the user with a description and the date the picture was taken. The <i>filename</i> column contains a single, computed filename without path information.

Domain model

Those tables are mapped to the following domain model:



Domain model (JPA Entities)

Name	Description
BikeEntity	A bike was bought on a given date and can be decommissioned. It has a color and an optional link to an arbitrary website. It may or may not have milages recorded. It has some important functions .
MilageEntity	A milage is part of a bike. For each bike one milage per month can be recored. The milage is the combination of it's recording date, the amount and the bike.
TrackEntity	The representation of <i>tracks</i> contents. The type is an enumeration. Notable public operations are <code>getPrettyId</code> (computes a “pretty” id based on the instances id) and <code>getTrackFile</code> (generates a reference to the GPS track file in the passed data storage directory).
BikingPictureEntity	For handling pictures collected from <i>Daily Fratze</i> . The BikingPictureEntity parses the image link on construction and retrieves the unique, external id.
GalleryPictureEntity	A bean for handling the pictures uploaded by the user. <code>prePersist</code> fills the <code>createdAt</code> attribute prior to inserting into the database.
AssortedTripEntity	This entity captures a distance which was covered on a certain date and can be used for keeping track of trips with bikes not stored in this application for example.
LocationEntity	Used in the tracker module for working with real time locations.

Important business methods on BikeEntity

Name	Description
decommission	Decommissions a bike on a given date.
addMilage	Adds a new milage for a given date and returns it. The milage will only be added if the date is after the date the last milage was added and if the amount is greater than the last milage.
getPeriods	Gets all monthly periods in which milages have been recorded.
getMilage	Gets the total milage of this bike.
getLastMilage	Gets the last milage recorded. In most cases the same as getMilage.
getMilageInPeriod	Gets the milage in a given period.
getMilagesInYear	Gets all milages in a year as an array (of months).
getMilageInYear	Gets the total milage in a given year.

Persistency

biking2 uses an H2⁶¹ database for storing relational data and the file system for binary image files and large ascii files (especially all GPS files).

During development and production the H2 database is retained and not in-memory based. The location of this file is configured through the `biking2.database-file` property and the default value during development is `./var/dev/db/biking-dev` relative to the working directory of the VM.

All access to the database goes through JPA using Hibernate as provider. See the [Domain Models](#) for all entities used in the application.

The JPA Entity Manager isn't accessed directly but only through the facilities offered by Spring Data JPA, that is through repositories only.

All data stored as files is stored relative to `biking2.datastore-base-directory` which defaults to `./var/dev`. Inside are 3 directories:

⁶¹<https://www.h2database.com/html/main.html>

- `bikingPictures`: Contains all pictures collected from *Daily Fratze*
- `galleryPictures`: Contains all uploaded pictures
- `tracks`: Contains uploaded GPS data and the result of converting TCX files into GPX files

User Interface

The default user interface for *biking2* which is packaged within the final artifact is a Single Page Application written in JavaScript using *Angular JS* together with a very default *Bootstrap* template.

For using the realtime location update interface, choose one of the many MQTT clients out there.

There is a second user interface written in Java called [bikingFX⁶²](#).

JavaScript and CSS optimization

JavaScript and CSS dependencies are managed through Maven dependencies in form of [webjars⁶³](#) wherever possible without the need for brew, npm, bower and the like.

Furthermore *biking2* uses [wro4j⁶⁴](#) together with a small [Spring Boot Starter⁶⁵](#) to optimize JavaScript and CSS web resources.

wro4j provides a model like this:

⁶²<https://info.michael-simons.eu/2014/10/22/getting-started-with-javafx-8-developing-a-rest-client-application-from-scratch/>

⁶³<https://www.webjars.org>

⁶⁴<https://alexo.github.io/wro4j/>

⁶⁵<https://github.com/michael-simons/wro4j-spring-boot-starter>

Wro4j configuration

```
1 <groups xmlns="http://www.isdc.ro/wro">
2     <!-- Dependencies for the full site -->
3     <group name="biking2">
4         <group-ref>osm</group-ref>
5
6             <css minimize="false">/webjars/bootstrap/@bootstrap.version@/css/b\
7 ootstrap.min.css</css>
8             <css>/css/stylesheet.css</css>
9
10            <js minimize="false">/webjars/jquery/@jquery.version@/jquery.min.j\
11 s</js>
12            <js minimize="false">/webjars/bootstrap/@bootstrap.version@/js/boo\
13 tstrap.min.js</js>
14            <js minimize="false">/webjars/momentjs/@momentjs.version@/min/mome\
15 nt-with-locales.min.js</js>
16            <js minimize="false">/webjars/angular-file-upload/@angular-file-up\
17 load.version@/angular-file-upload-html5-shim.min.js</js>
18            <js minimize="false">/webjars/angularjs/@angularjs.version@/angula\
19 r.min.js</js>
20            <js minimize="false">/webjars/angularjs/@angularjs.version@/angula\
21 r-route.min.js</js>
22            <js minimize="false">/webjars/angular-file-upload/@angular-file-up\
23 load.version@/angular-file-upload.min.js</js>
24            <js minimize="false">/webjars/angular-ui-bootstrap/@angular-ui-boo\
25 tstrap.version@/ui-bootstrap.min.js</js>
26            <js minimize="false">/webjars/angular-ui-bootstrap/@angular-ui-boo\
27 tstrap.version@/ui-bootstrap-tpls.min.js</js>
28            <js minimize="false">/webjars/highcharts/@highcharts.version@/high\
29 charts.js</js>
30            <js minimize="false">/webjars/highcharts/@highcharts.version@/high\
31 charts-more.js</js>
32            <js minimize="false">/webjars/sockjs-client/@sockjs-client.version\
33 @/sockjs.min.js</js>
34            <js minimize="false">/webjars/stomp-websocket/@stomp-websocket.ver\
35 sion@/stomp.min.js</js>
```

```
36
37      <js>/js/app.js</js>
38      <js>/js/controllers.js</js>
39      <js>/js/directives.js</js>
40    </group>
41 </groups>
```

This model file is filtered by the Maven build, version placeholders will be replaced and all resources, in webjars as well as inside the filesystem, will be available as `biking.css` and `biking.js`.

How those files are optimized, minimized or otherwise processed is up to wro4js configuration, but minification can be turned off during development.

Transaction Processing

biking2 relies on Spring Boot to create all necessary beans for handling local transactions within the JPA EntityManager. *biking2* does not support distributed transactions.

Session Handling

biking2 only provides a stateless public API, there is no session handling.

Security

biking2 offers security for its API endpoints only via [HTTP basic access authentication](#)⁶⁶ and in case of the MQTT module with MQTTs default security model. Security can be increased by running the application behind a SSL proxy or configuring SSL support in the embedded Tomcat container.

For the kind of data managed here it's an agreed tradeoff to keep the application simple. See also [Safety](#).

⁶⁶https://en.wikipedia.org/wiki/Basic_access_authentication

Safety

No part of the system has life endangering aspect.

Communications and Integration

biking2 uses an internal Apache ActiveMQ broker on the same VM as the application for providing STOMP channels and a MQTT transport. This broker is volatile, messages are not persisted during application restarts.

Plausibility and Validity Checks

Datatypes and ranges are checked via [JSR-303](#)⁶⁷ annotations on classes representing the [Domain Models](#). Those classes are directly bound to external REST interfaces.

There are three important business checks:

1. Bikes which have been decommissioned cannot be modified (i.e. they can have no new milages): Checked in `BikesController`.
2. For each unique month only one milage can be added to a bike. Checked in the `BikeEntity`.
3. A new milage must be greater than the last one. Also checked inside `BikeEntity`.

Exception/Error Handling

Errors handling to inconsistent data (in regard to the data models constraint) as well as failures to [validation](#) are mapped to HTTP errors. Those errors are handled by the frontends controller code. Technical errors (hardware, database etc.) are not handled and may lead to application failure or lost data.

⁶⁷<https://beanvalidation.org/1.0/spec/>

Logging, Tracing

Spring Boot configures logging per default to standard out. The default configuration isn't change in that regard, so all framework logging (especially Spring and Hibernate) go to standard out in standard format and can be grabbed or ignored via OS specific means.

All business components use the *Simple Logging Facade for Java (SLF4J)*. The actual configuration of logging is configured through the means of Spring Boot. No special implementation is included manually, instead *biking2* depends transitively on `spring-boot-starter-logging`.

The names of the logger corresponds with the package names of the classes which instantiate loggers, so the modules are immediately recognizable in the logs.

Configurability

Spring Boot offers a plethora of configuration options, those are just the main options to configure Spring Boot and available starters: [Common application properties](#)⁶⁸.

The default configuration is available in `src/main/resources/application.properties`. During development those properties are merged with `src/main/resources/application-dev.properties`. Additional properties can be added through system environment or through an `application-*.properties` in the current JVM directory.

During tests an additional `application-test.properties` can be used to add or overwrite additional properties or values.

Those are the *biking2* specific properties:

⁶⁸<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

Property	Default	Description
biking2.color-of-cumulative-graph	000000	Color of the cumulative line graph
biking2.dailyfratze-access-token	n/a	An OAuth access token for <i>Daily Fratze</i>
biking2.datastore-base-directory	\${user.dir}/var/dev	Directory for storing files (tracks and images)
biking2.fetch-biking-picture-cron	0 0 */8 * * *	A cron expression for configuring the <code>FetchBikingPicturesJob</code>
biking2.home.longitude	6.179489185520004	Longitude of the home coordinate
biking2.home.latitude	50.75144902272457	Latitude of the home coordinate
biking2.connector.proxyName	n/a	The name of a proxy if <i>biking2</i> runs behind one
biking2.connector.proxyPort	80	The port of a proxy if <i>biking2</i> runs behind one
biking2.gpsBabel	/opt/local/bin/gpsbabel	Fully qualified path to the <i>GPSBabel</i> binary
biking2.scheduled-thread-pool-size	10	Thread pool size for the job pool
biking2.tracker.host	localhost	The host on which the tracker (MQTT channel) should listen
biking2.tracker.stompPort	2307	STOMP port
biking2.tracker.mqttPort	4711	MQTT port

Property	Default	Description
biking2.tracker.username	\${security.user.name}	Username for the MQTT channel
biking2.tracker.password	\${security.user.password}	Password for the MQTT channel
biking2.tracker.device	iPhone	Name of the OwnTracks device

Internationalization

Only supported language is English. There is no hook for doing internationalization in the frontend and there are no plans for creating one.

Migration

biking2 replaced a Ruby application based on the *Sinatra* framework. Data was stored in a SQLite database which has been migrated by hand to the H2 database.

Testability

The project contains JUnit tests in the standard location of a Maven project. At the time of writing those tests covers >95% of the code written. Tests must be executed during build and should not be skipped.

Build-Management

The application can be build with Maven without external dependencies outside Maven. *gpsbabel* must be on the path to run all tests, though.

IV.9 Architecture Decisions

Using GPSBabel for converting TCX into GPX format

Problem

Popular JavaScript mapping frameworks provide easy ways to include geometries from GPX data on maps. Most Garmin devices however record track data in TCX format, so I needed a way to convert TCX to GPX. Both formats are relatively simple and in case of GPX good documented formats.

Constraints

- Conversion should handle TCX files with single tracks, laps and additional points without problem
- Focus for this project has been on developing a modern application backend for an AngularJS SPA, not parsing GPX data

Assumptions

- Using an external, non Java based tool makes it harder for people who just want to try out this application
- Although good documented, both file types can contain varieties for informations (routes, tracks, waypoints) which makes it hard to parse

Considered Alternatives

- Writing my own converter
- Using existing swiss army knife for GPS data: [GPSBabel⁶⁹](https://www.gpsbabel.org):

GPSBabel converts waypoints, tracks, and routes between popular GPS receivers such as Garmin or Magellan and mapping programs like Google Earth or Basecamp. Literally hundreds of GPS receivers and programs are supported. It also has powerful manipulation tools for such data. such as

⁶⁹<https://www.gpsbabel.org>

filtering duplicates points or simplifying tracks. It has been downloaded and used tens of millions of times since it was first created in 2001, so it's stable and trusted.

Decision

biking2 uses GPSBabel for the heavy lifting of GPS related data. The project contains a README stating that GPSBabel must be installed. GPSBabel can be installed on Windows with an installer and on most Linux systems through the official packet manager. Under OS X it is available via MacPorts or Homebrew.

Using local file storage for image and track data

Problem

biking2 needs to store “large” objects: Image data (biking and gallery pictures) as well as track data.

Considered Alternatives

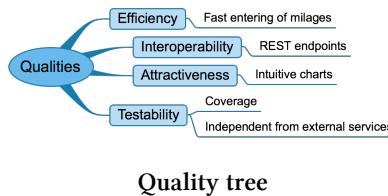
- Using some kind of Cloud storage like S3
- Using local file system

Decision

I opted for local file system because I didn't want to put much effort into evaluating cloud services. If *biking2* should runnable in cloud based setup, one has to create an abstraction over the local filesystem currently used.

IV.10 Quality Requirements

10.1 Quality Tree



10.2 Evaluation Scenarios

Testability / Coverage

By using *JaCoCo* during development and the build process⁷⁰ ensure a code coverage of at least 95%.

Testability / Independent from external services

The architecture should be designed in such a way that algorithms depending on external services can be tested without having the external service available. That is: All external dependencies should be mockable.

Example: `FetchBikingPicturesJob` needs a resource containing a RSS feed. Retrieving the resource and parsing it are at least two different tasks. Fetching the resource through a separate class `DailyFratzeProvider` makes testing the actual parsing independent from a HTTP connection and thus relatively simple.

⁷⁰<https://info.michael-simons.eu/2014/05/22/jacoco-maven-and-netbeans-8-integration/>

IV.11 Risks and Technical Debt

biking2 has been up and running for nearly 2 years now, the architecture contains no known risk for my usage scenario.

There is a possibility that the H2 database can be damaged due to an unexpected shutdown of the VM (that is OS or hardware failure). The risk is mitigated through regularly backups of the serialized database file.

IV.12 Glossary

Term	Description
AngularJS	AngularJS ⁷¹ is an open-source web application framework mainly maintained by Google and by a community of individual developers and corporations to address many of the challenges encountered in developing single-page applications.
Apache ActiveMQ	Apache ActiveMQ ⁷² is an open source message broker written in Java.
Apache License	The Apache License ⁷³ is a permissive Open Source license, especially designed for free software.
Bootstrap	Bootstrap ⁷⁴ is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.
Checkstyle	Checkstyle ⁷⁵ is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard.
Daily Fratze (DF)	An online community where users can upload a daily picture of themselves (a selfie, but the site did them before they were called selfies).
Fat Jar	A way of packaging Java applications into one single Jar file containing all dependencies, either repackaged or inside their original jars together with a special class loader.
Gallery picture	Pictures from tours provided manually by the hours in addition to the pictures collected automatically from <i>Daily Fratze</i> .

⁷¹<https://en.wikipedia.org/wiki/AngularJS>

⁷²https://en.wikipedia.org/wiki/Apache_ActiveMQ

⁷³<https://www.apache.org/licenses/LICENSE-2.0>

⁷⁴<https://getbootstrap.com>

⁷⁵<https://checkstyle.sourceforge.net>

Term	Description
Garmin	Garmin⁷⁶ develops consumer, aviation, outdoor, fitness, and marine technologies for the Global Positioning System.
GPSBabel	GPSBabel⁷⁷ is a command line utility that converts waypoints, tracks, and routes between popular GPS receivers such as Garmin or Magellan and mapping programs like Google Earth or Basecamp.
GPS Exchange Format	GPX , or GPS Exchange Format, is an XML schema designed as a common GPS data format for software applications. It can be used to describe waypoints, tracks, and routes.
JaCoCo	JaCoCo⁷⁸ is a code coverage library which can be used very easily from within NetBeans.
Java 8 or JDK 8	The eight installment of the Java programming language⁷⁹ and the first one to support functional paradigms in the form Lambda expressions.
JUnit	JUnit⁸⁰ is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.
MQTT	MQTT⁸¹ is a publish-subscribe based “light weight” messaging protocol for use on top of the TCP/IP protocol.
NetBeans	NetBeans⁸² is a free and open Source IDE that fits the pieces of modern development together.
OAuth	OAuth⁸³ is an open standard for authorization. OAuth provides client applications a ‘secure delegated access’ to server resources on behalf of a resource owner.

⁷⁶<https://en.wikipedia.org/wiki/Garmin>

⁷⁷<https://www.gpsbabel.org>

⁷⁸<https://eclemma.org/jacoco/>

⁷⁹[https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))

⁸⁰<https://junit.org/junit4/>

⁸¹<https://en.wikipedia.org/wiki/MQTT>

⁸²<https://netbeans.org>

⁸³<https://en.wikipedia.org/wiki/OAuth>

Term	Description
oEmbed	The oEmbed⁸⁴ protocol is a simple and lightweight format for allowing an embedded representation of an URL on third party sites.
RSS	Rich Site Summary or Really Simple Syndication. RSS⁸⁵ uses a family of standard web feed formats to publish frequently updated information: blog entries, news headlines, audio, video.
SLF4J	The Simple Logging Facade for Java (SLF4J)⁸⁶ serves as a simple facade or abstraction for various logging frameworks (e.g. <code>java.util.logging</code> , <code>logback</code> , <code>log4j</code>) allowing the end user to plug in the desired logging framework at deployment time.
SPA	Single Page Application
Spring Boot	Spring Boot⁸⁷ makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run”.
Spring Data JPA	Spring Data JPA⁸⁸ , part of the larger Spring Data family, makes it easy to easily implement JPA based repositories.
STOMP	Simple Text Oriented Messaging Protocol STOMP⁸⁹ provides an interoperable wire format so that STOMP clients can communicate with any STOMP message broker to provide easy and widespread messaging interoperability among many languages, platforms and brokers.
TCX	Training Center XML (TCX) is a data exchange format introduced in 2007 as part of Garmin’s Training Center product.

⁸⁴<https://oembed.com>

⁸⁵<https://en.wikipedia.org/wiki/RSS>

⁸⁶<https://www.slf4j.org>

⁸⁷<https://projects.spring.io/spring-boot/>

⁸⁸<https://projects.spring.io/spring-data-jpa/>

⁸⁹<https://stomp.github.io>

V - DokChess

By Stefan Zörner.

“Someday computers will make us all obsolete.”

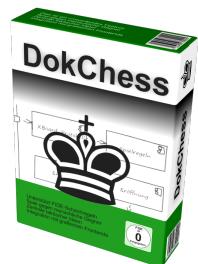
Robert (“Bobby”) Fisher, World Chess Champion 1972-1975, in 1975.

This chapter describes the architecture of the chess program *DokChess*. I originally created it as an example for presentations and training on software architecture and design. Later on, I implemented it in Java and refined it for a German book on documenting software architectures (see [www.swadok.de⁹⁰](https://www.swadok.de)). The source code is available on GitHub and more information, including this architectural overview in both English and German, can be found on [www.dokchess.de⁹¹](https://www.dokchess.de).

With the following architectural overview, you will be able to understand the important design decisions of DokChess. It shows the requirements relevant to the structure and design, basic solutions to problems, the structure of the software and the interaction of key elements. The outline of the content follows the arc42 template.

The target audience of this architectural overview is primarily software architects seeking advice and examples on how to document architecture appropriately.

Software developers who plan to create a chess program of their own receive valuable tips and learn about software architecture too.



⁹⁰<https://www.swadok.de>

⁹¹<https://www.dokchess.de>

V.1 Introduction and Goals

This section introduces the task and outlines the objectives pursued by DokChess.

1.1 Requirements Overview

What is DokChess?

- DokChess is a fully functional chess engine.
- It serves both as an easily accessible as well as attractive case study of software architecture design, evaluation and documentation.
- The understandable structure invites developers to experiment and to extend the engine.
- A high level of chess ability is not the goal. Nevertheless DokChess manages to play games which casual chess players will enjoy.

Essential Features

- Complete compliance with the FIDE Laws of Chess
- Supports games against human opponents and other chess programs
- Masters fundamental chess tactics, such as fork and skewer
- Works with modern graphical chess frontends

1.2 Quality Goals

The following table describes the key quality objectives of DokChess. The order of the goals gives you a rough idea of their importance.

Quality Goal	Motivation/description
Accessible example (Analysability)	Since DokChess first of all serves as a case study for software architects and developers, they quickly get the idea of design and implementation.
Platform appealing to experiments (Changeability)	Alternative algorithms and strategies, such as the evaluation of a chess position, can be implemented and integrated into the solution easily.
Using existing frontends (Interoperability)	DokChess can be integrated in existing graphical chess frontends with reasonable effort.
Acceptable playing strength (Attractiveness)	DokChess plays strong enough to beat weak opponents safely and at least challenges casual players.
Quick response to opponent's moves (Efficiency)	Since DokChess is used as live-demo in seminars and lectures, calculation of moves takes place quickly.

The quality scenarios in section [V.10](#) detail these goals and serve to evaluate their achievement.

1.3 Stakeholders

The following table illustrates the stakeholders of DokChess and their respective intentions.

Who?	Matters and concern
Software Architects	Software architects get an impression on how architecture documentation for a specific system may look like. They reproduce things (e.g. format, notation) in their daily work. They gain confidence for their own documentation tasks. Usually they have no deep knowledge about chess.
Developers	Developers accept responsibility for architectural tasks in the team. During the study of DokChess they acquire a taste for implementing a chess engine on their own. They are curious about concrete suggestions.
Stefan Zörner	Stefan needs attractive examples for his book. He uses DokChess as a case study in workshops and presentations on software design and architecture.
oose Innovative Informatik	Employers of Stefan Zörner at the time of DokChess conception. The company offers trainings, workshops and coaching on topics related to software development.

V.2 Constraints

At the beginning of the project various constraints had to be respected within the design of DokChess. They still affect the solution. This section represents these restrictions and explains – where necessary – their motivations.

2.1 Technical Constraints

Constraint	Background and / or motivation
Moderate hardware equipment	Operating DokChess on a standard notebook in order to show the software in the context of workshops and conferences on such a device.
Operating on Windows desktop operating systems	Standard equipment for notebooks of loose employees at the time of outlining the solution. High distribution of these operating systems at potentially interested parties (audience at conference talks, participants in trainings). Support for other operating systems (most notably Linux and Mac OS X) is desirable, but not mandatory.
Implementation in Java	Usage as an example in Java-centered trainings and Java conferences. Development with Java SE version 6 (DokChess 1.0), later Java SE 7. The engine should also run in newer Java versions, when available.
Third-party software freely available	If third-party software is involved (for example, a graphical front end), this should ideally be freely available and free of charge. The way the threshold to use it is kept low.

2.2 Organizational Constraints

Constraint	Background and / or motivation
Team	Stefan Zörner, supported by colleagues, friends and interested workshop or training participants
Schedule	Start of development in December 2010, first running prototype March 2011 (evening talk at oose in Hamburg), presentable version May 2011 (talk at JAX conference in Mayence, Germany). Completion of Version 1.0: February 2012 (Deadline book manuscript for 1st edition).
Process model	Risk driven development, iterative and incremental. To describe the architecture arc42 is used. An architecture documentation structured according to this template is a key project result.
Development Tools	Design with pen and paper, in addition Enterprise Architect. Work results for architecture documentation collected in Confluence Wiki. Java source code created in Eclipse or IntelliJ. However, the software can be built only with Gradle, i.e. without an IDE.
Configuration and version management	At the beginning (Version 1.0) Subversion at SourceForge, later Git at GitHub.
Test tools and test processes	JUnit 4 with annotation style both for correctness and integration testing and for compliance with efficiency targets.

Constraint	Background and / or motivation
Release as Open Source	The source code of the solution, or at least parts, made available as open source. License: GNU General Public License version 3.0 (GPLv3). Hosted at GitHub ⁹² .

2.3 Conventions

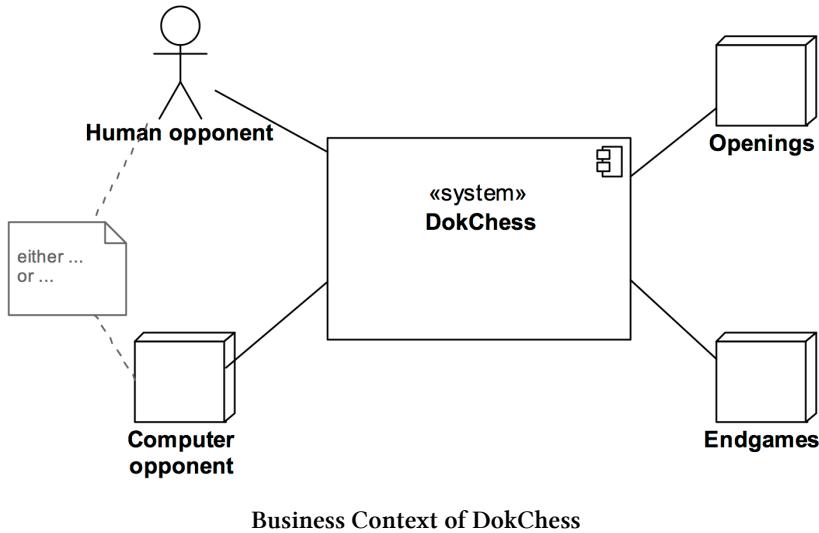
Convention	Background and / or motivation
Architecture documentation	Terminology and structure according to the arc42 template, version 6.0
Coding guidelines for Java	Java coding conventions of Sun / Oracle, checked using CheckStyle
Chess-specific file formats	Use of established standards for chess-specific notations and exchange formats within the solution. Topics: moves, positions, games, openings, ... Never develop own formats here. Principle: Favour open standards over proprietary formats (which commercial programs may use).

⁹²<https://github.com/DokChess/>

V.3 System Scope and Context

This section describes the environment of DokChess. Who are its users, and with which other systems does it interact with.

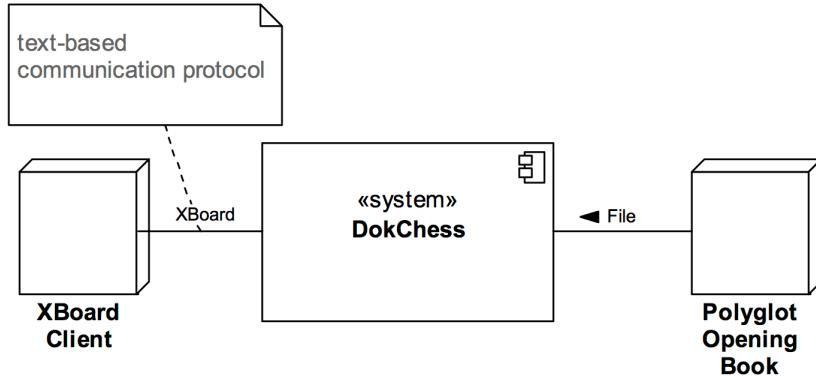
3.1 Business Context



Actor	Description
Human opponent (user)	Chess is played between two opponents, who move their pieces in turn. DokChess takes the role of one of the opponents, and competes against a human opponent. For this purpose, the two need to communicate, e.g. about their moves, or draw offers.
Computer opponent (external system)	As an alternative to a human opponent DokChess can also compete with a different engine. The requirements related to the exchange of information are the same.

Actor	Description
Openings (external system)	About the opening, which is the early stage of a game, extensive knowledge exists in chess literature. This knowledge is partly free and partly also commercially available in the form of libraries and databases. Within DokChess no such library is created. Optionally an external system is connected instead in order to permit a knowledge based play in the early stages, as expected by human players.
Endgames (external system)	If just a very few pieces are left on the board (e.g. only the two kings and a queen), endgame libraries can be used analogously to opening libraries. For any position with this piece constellation these libraries include the statement whether a position is won, drawn or lost, and if possible the necessary winning move. Within DokChess no such library is created. Optionally an external system is connected instead in order to bring clearly won games home safely, or to use the knowledge from the libraries for analysis and position evaluation.

3.2 Technical Context



Technical communication of DokChess with third parties

Actor	Description
XBoard client (external system)	A human player is connected to DokChess with a graphical front-end. The development of such is not part of DokChess. Each graphical frontend can be used instead, if it supports the so-called XBoard protocol. These include Xboard (or Winboard on Windows), Arena and Aquarium.
Polyglot Opening Book (external system)	Polyglot Opening Book is a binary file format for opening libraries. DokChess allows the optional connection of such books. Only read access is used.

On endgames

The implementation of a connection to endgame databases (such as [Nalimov Endgame tablebases⁹³](#)) has been dropped due to the effort of implementation (→ V.11.2 “Risk: Implementation effort too high”). The design, however, is open to appropriate extensions.

⁹³https://en.wikipedia.org/wiki/Endgame_tablebase

V.4 Solution Strategy

The following table contrasts the quality goals of DokChess (→ V.1.2) with matching architecture approaches and thus provides easy access to the solution.

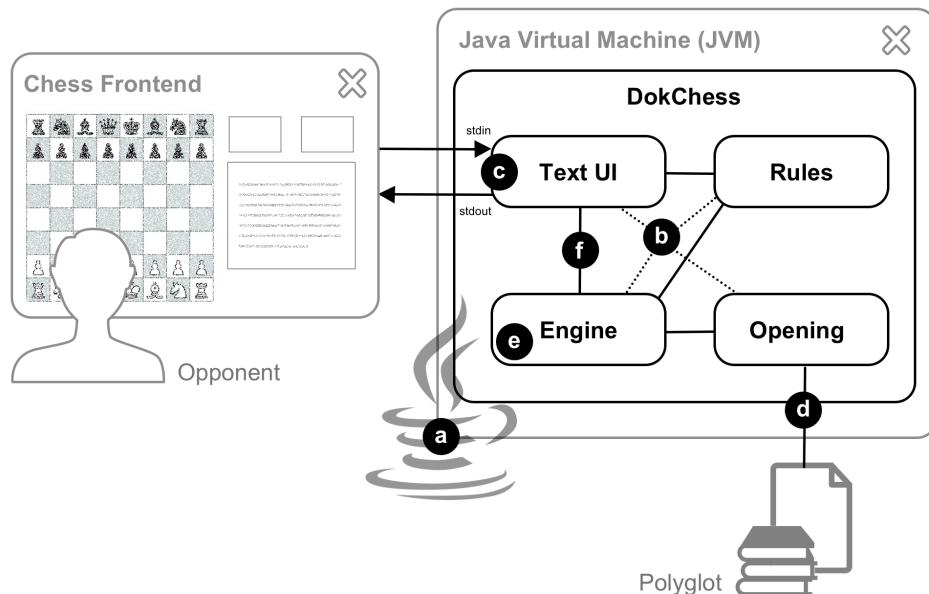
Quality Goal	Matching approaches in the solution
Accessible example (Analysability)	<ul style="list-style-type: none"> • architectural overview structured by arc42 • explicit, object-oriented domain model • detailed documentation of public interfaces with Javadoc
Platform appealing to experiments (Changeability)	<ul style="list-style-type: none"> • widely spread programming language Java → (a) • Interfaces for core abstractions (for instance: position evaluation, game rules) • immutable objects (position, move, ...) make implementation of many algorithms easier • “plugging” of elements with dependency injection leads to interchangeability → (b) • High test coverage as a safety net
Using existing frontends (Interoperability)	<ul style="list-style-type: none"> • Use of the common communication protocol XBoard, → (c) • Use of portable Java → (a)
Acceptable playing strength (Attractiveness)	<ul style="list-style-type: none"> • Integration of chess opening book libraries → (d) • implementation of minimax algorithm and a proper position evaluation → (e) • Integration tests with chess problems for tactics and mate positions
Quick response to opponent’s moves (Efficiency)	<ul style="list-style-type: none"> • Reactive extensions for concurrent calculation with newly found better moves as events → (f) • Optimization of minimax by alpha-beta pruning → (e) • Efficient domain model implementation

Quality Goal

Matching approaches in the solution

- Integration tests with time limits

Small letters in brackets, e.g. (x), link individual approaches from the right hand of the table to the following architectural overview diagram.



DokChess Architectural Overview

The remaining section V.4 introduces significant architectural aspects and refers to further information in chapter V.

4.1 Structure of DokChess

DokChess is implemented as a Java program with a main routine. It is roughly split into the following parts:

- An implementation of the rules of chess
- The engine itself, which selects the moves
- The connection to a graphical user interface via the XBoard protocol
- An adapter for a specific opening book format (Polyglot Opening Book)

This decomposition allows you to replace things such as the communication protocol or the opening book format if necessary. All parts are abstracted through interfaces. Their implementations are assembled via dependency injection (→ V.5 “Building Block View”, → Concept V.8.1 “Dependencies Between Modules”). The decomposition further allows the software, especially the chess algorithms, to be tested automatically. (→ Concept V.8.7 “Testability”).

The interaction between algorithms takes place using the exchange of data structures motivated by the domain implemented as Java classes (piece, move and so on → Concept V.8.2 “Chess Domain Model”). Here, better understandability is preferred at the cost of efficiency. Nevertheless, DokChess reached an acceptable playing strength, as a run through the corresponding scenarios shows (→ V.10 “Quality Scenarios”).

The key element of the data structure design is the game situation. This includes the placement of the chess pieces and other aspects that belong to the position (such as which side moves next). Again readability is preferred to efficiency in the implementation of the class motivated by the domain. An important aspect is that, like all other domain classes this class is immutable (→ decision V.9.2 “Are position objects changeable or not?”).

4.2 Game Strategy

For the integration of opening libraries, the “Polyglot Opening Book” file format was implemented (→ Building Block View V.5.1.4 “Subsystem Opening (Blackbox)”). This way, DokChess responds with profound chess knowledge in the beginning of a game.

The classic [minimax algorithm⁹⁴](#) with a fixed search depth in the game tree is responsible for the strategy as the game continues. Its basic implementation is single-threaded. The evaluation of a position at an end node in the game tree is based solely on the material (→ Building Block View level 2, V.5.2 “Engine (Whitebox)”). Nevertheless, these simple implementations already meet the quality scenarios under the given constraints.

An [alpha-beta pruning⁹⁵](#) illustrates the simple replacement of algorithms. Playing strength and efficiency considerably improve by searching the tree more deeply

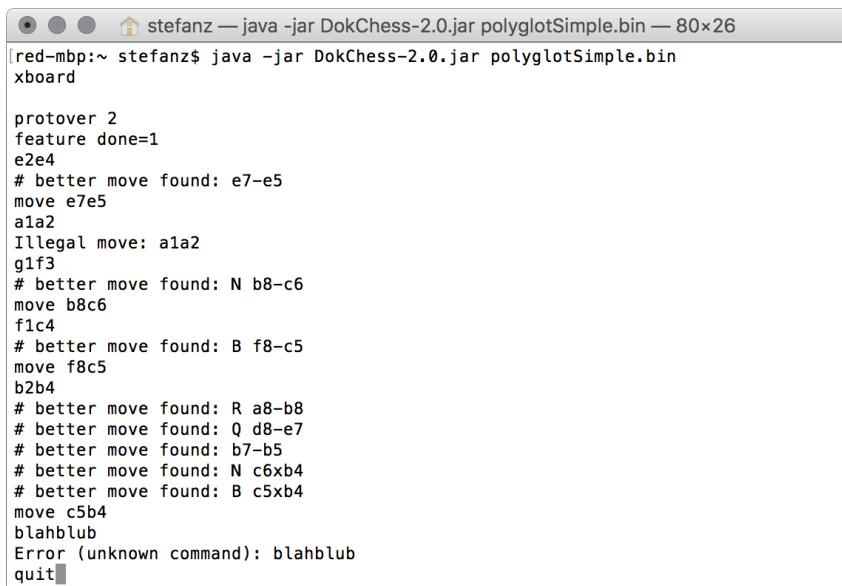
⁹⁴<https://en.wikipedia.org/wiki/Minimax>

⁹⁵https://en.wikipedia.org/wiki/Alpha–beta_pruning

within the same computation time. The immutable data structures of DokChess also facilitate implementing concurrent algorithms; a parallel minimax algorithm is included as an example.

4.3 The Connection of the Engine

DokChess has no graphical user interface. Instead communication takes place via standard input and output. The text-based XBoard acts as a communication protocol (→ decision V.9.1 “How does the engine communicate with the outside world?”). You can use DokChess interactively with the command line if you know the XBoard commands and are able to interpret the engine responses (→ Concept V.8.3 “User Interface”). See image below:



```
stefanz — java -jar DokChess-2.0.jar polyglotSimple.bin — 80x26
[red-mbp:~ stefanz$ java -jar DokChess-2.0.jar polyglotSimple.bin
xboard

protover 2
feature done=1
e2e4
# better move found: e7-e5
move e7e5
a1a2
Illegal move: a1a2
g1f3
# better move found: N b8-c6
move b8c6
f1c4
# better move found: B f8-c5
move f8c5
b2b4
# better move found: R a8-b8
# better move found: Q d8-e7
# better move found: b7-b5
# better move found: N c6xb4
# better move found: B c5xb4
move c5b4
blahblub
Error (unknown command): blahblub
quit]
```

DokChess on the command line

The actual DokChess engine is attached by a reactive approach (“Reactive Extensions”) (→ Runtime view, V.6.1 “Move calculation Walkthrough”). DokChess is accessible even during its analysis. This way, for instance a user can force the engine to move immediately.

On Windows, integrating DokChess in a UI is done with a batch file (*.bat). It starts

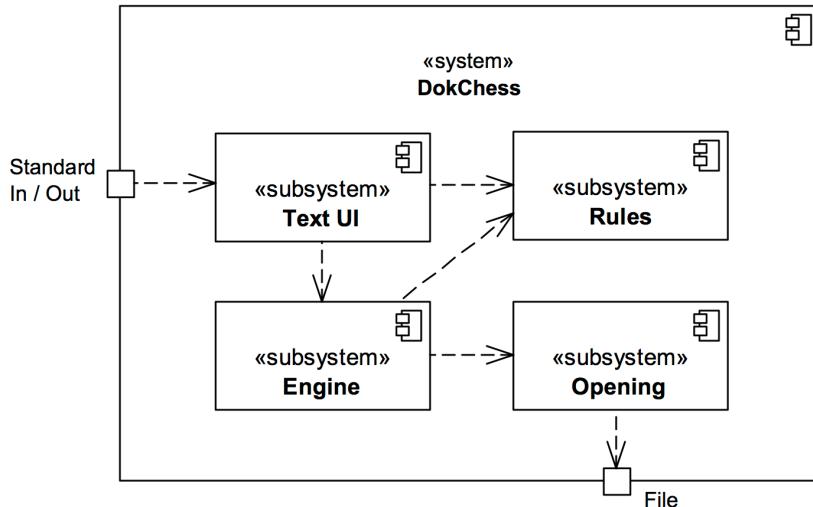
the Java Virtual Machine (JVM) with the class with the *main* method as a parameter
(→ [V.7](#) “Deployment View”).

V.5 Building Block View

This section describes the decomposition of DokChess into modules. These are also reflected in the package structure of the Java source code. In DokChess we call modules of the first decomposition level *subsystems*. The building block view level 1 presents them including their interfaces. For the subsystem Engine this overview also includes a more detailed breakdown into level 2 (→ V.5.2). Section V.6.1 (“Move Determination Walkthrough”) contains an example of the interaction between the subsystems at runtime.

5.1 Building Block View, Level 1

DokChess breaks down in four subsystems as presented below. The dashed arrows represent logical dependencies between the subsystems (“ $x \rightarrow y$ ” for “ x depends on y ”). The squared boxes on the membrane of the system are interaction points (“ports”) with the outside world (→ V.3.2 “Deployment Context”).



DokChess, building block view, level 1

Subsystem	Short description
Text UI	Realizes communication with a client using the XBoard protocol.
Rules	Provides the rules of chess and for instance can determine all valid moves for a position.
Engine	Contains the determination of a next move starting from a game situation.
Opening	Provides standard moves of the chess opening literature for a game situation.

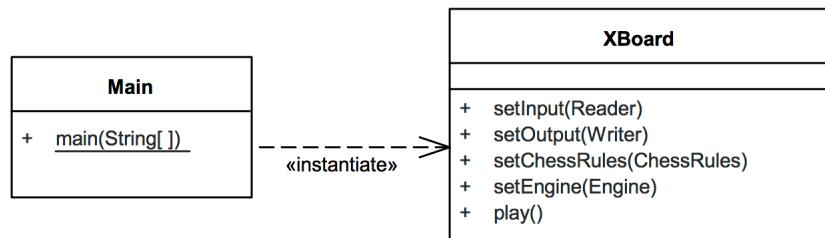
5.1.1 Subsystem Text UI (Blackbox)

Intent/Responsibility

This subsystem implements the communication with a client (for example, a graphical user interface) using the text-based XBoard protocol (→ Decision V.9.1). It reads commands from standard input, checks them against the rules of the game and converts them for the *Engine*. Responses from the Engine (especially the moves) will be accepted by the subsystem as events, formatted according to the protocol and returned via standard output. Thus the subsystem is driving the whole game.

Interfaces

The subsystem provides its functionality via the Java class `org.dokchess.textui.xboard.XBoard`.



Classes XBoard and Main

Method	Short description
setInput	Set the protocol input with a dependency injection (→ Concept V.8.1). Typically, the standard input (stdin), automated tests use a different source.
setOutput	Set the protocol output. Typically, the standard output (stdout), automated tests may use a different target.
setChessRules	Sets an implementation of the game rules, → V.5.1.2 “Subsystem Rules (Black Box)”
setEngine	Sets an implementation of the engine, → V.5.1.3 “Subsystem Engine (Black Box)”
play	Starts the actual communication (input / processing / output) in a infinite loop until the quit command.

Files

The implementation is located below the packages
`org.dokchess.textui...`

Open Issues

The implementation of the XBoard protocol is incomplete. Nevertheless it is sufficient for the requirements of DokChess. But in particular, the following features are not supported:

- Time control
- Permanent brain (thinking while the opponent thinks)
- Draw-offers and giving up of the opponent
- Chess variants (alternative rules, such as Chess960)

5.1.2 Subsystem Rules (Blackbox)

Intent/Responsibility

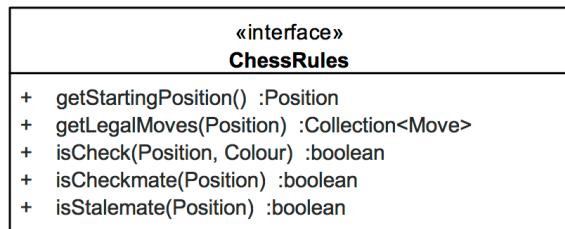
This subsystem accounts for the rules of chess according to the International Chess Federation (FIDE). It determines all valid moves for a position and decides whether

it is a check, a checkmate or a stalemate.

Interfaces

The subsystem provides its functionality via the Java interface `org.dokchess.rules.ChessRules`.

Default implementation of the interface is class `org.dokchess.rules.DefaultChessRules`.



Interface `ChessRules`

Method	Short description
<code>getStartingPosition</code>	Returns the starting position of the game. White begins.
<code>getLegalMoves</code>	Returns the set of all legal moves for a given position. The current player is determined from the position parameter. In case of a mate or stalemate an empty collection is the result. Thus the method never returns null.
<code>isCheck</code>	Checks whether the king of the given colour is attacked by the opponent.
<code>isCheckmate</code>	Checks whether the given position is a mate. I.e. the king of the current player is under attack, and no legal move changes this. The player to move has lost the game.
<code>isStalemate</code>	Checks whether the given position is a stalemate. I.e. the current player has no valid move, but the king is not under attack. The game is considered a draw.

Concept V.8.2 “Chess Domain Model” describes the types used in the interface

as call and return parameters (`Move`, `Position`, `Colour`). Refer to the source code documentation (javadoc) for more details.

Files

The implementation is located below the packages
`org.dokchess.rules...`

Open Issues

Apart from the stalemate, the subsystem can not recognize any draw. In particular, the following rules are not implemented (→ [V.11.2](#) “Risk: Implementation effort too high”):

- 50 moves rule
- Threefold repetition

5.1.3 Subsystem Engine (Blackbox)

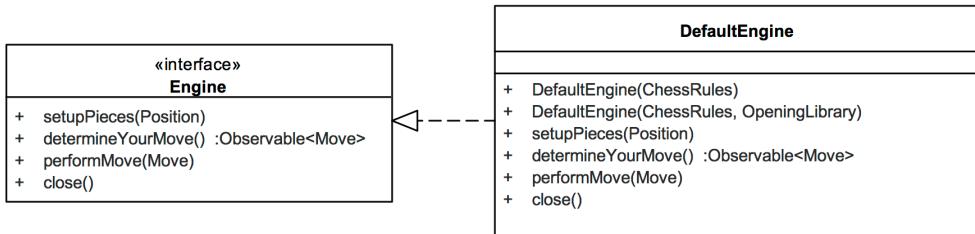
Intent/Responsibility

This subsystem contains the determination of a next move starting from a game position. The position is given from outside. The engine itself is stateful and always plays one game at the same time. The default implementation needs an implementation of the game rules to work. An opening library, however, is optional.

Interfaces

The *Engine* subsystem provides its functionality via the Java interface
`org.dokchess.engine.Engine`.

Default implementation is the class
`org.dokchess.engine.DefaultEngine`.



Interface Engine and class DefaultEngine

Methods of the Engine interface:

Method	Short description
setupPieces	Sets the state of the engine to the specified position. If currently a move calculation is running, this will be cancelled.
determineYourMove	Starts the determination of a move for the current game situation. Returns move candidates asynchronously via an Observable (→ Runtime View V.6.1 “Move Determination Walkthrough”). The engine does not perform the moves.
performMove	Performs the move given, which changes the state of the engine. If currently a move calculation is running, this will be canceled.
close	Closes the engine. The method makes it possible to free resources. No move calculations are allowed afterwards.

Methods of the **DefaultEngine** class (in addition to the **Engine** interface):

Method	Short description
DefaultEngine	Constructors, set an implementation of the chess rules, → V.5.1.2 Subsystem Rules (Black Box) and an (optional) opening book, whose moves will be preferred to own considerations → V.5.1.4 Subsystem Opening (Black Box).

Concept V.8.2 (“Chess domain model”) describes the types used in the interface as call and return parameters (`Move`, `Position`). Refer to the source code documentation (javadoc) for more information. You find details of the `Engine` subsystem implementation in the white box view in section V.5.2 of this overview.

Files

The implementation of the Engine subsystem and corresponding unit tests are located below the packages
`org.dokchess.engine...`

5.1.4 Subsystem Opening (Blackbox)

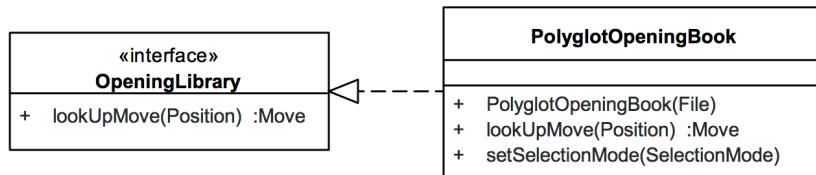
Intent/Responsibility

This subsystem provides opening libraries and implements the Polyglot opening book format. This format is currently the only one available, which is not proprietary. Corresponding book files and associated tools are freely available on the Internet.

Interfaces

The Opening subsystem provides its functionality via the Java interface
`org.dokchess.opening.OpeningLibrary`.

The class `org.dokchess.opening.polyglot.PolyglotOpeningBook` provides one possible implementation.



Interface `OpeningLibrary` and class `PolyglotOpeningBook`

Methods of the interface `OpeningLibrary`:

Method	Short description
<code>lookUpMove</code>	Returns a standard move for the specified position from the library, or null.

The `PolyglotOpeningBook` class is an adapter for the Polyglot opening book file format. Implementation of `OpeningLibrary` that reads a binary file in the appropriate format and returns a move to the specified position, if the library contains any.

Methods of the class `PolyglotOpeningBook` (in addition to interface `OpeningLibrary`):

Method	Short description
<code>PolyglotOpeningBook</code>	Constructor, expects the input file.
<code>setSelectionMode</code>	Sets the mode to select a move, if there is more than one candidate in the library for the given position.

Concept [V.8.2 “Chess Domain Model”](#) describes the types used in the interface as call and return parameters (`Move`, `Position`). Refer to the source code documentation (javadoc) for more information.

Files

The implementation, unit tests and test data for the Polyglot file format are located below the packages

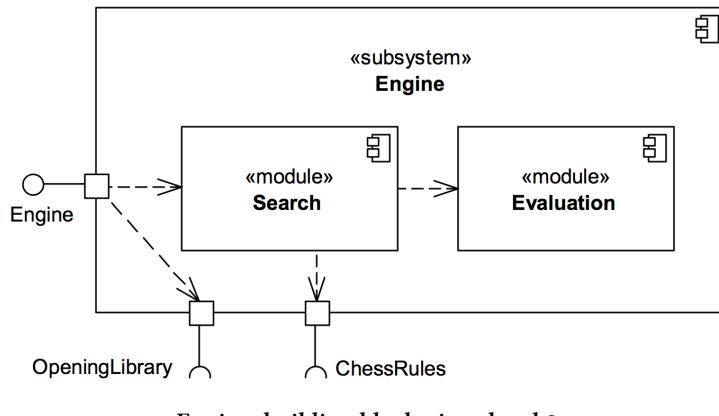
`org.dokchess.opening...`

Open Issues

- The implemented options for a move selection from the Polyglot opening book in case of several candidates are limited (the first, the most often played, by chance).
- The implementation can not handle multiple library files at the same time. It can therefore not mix them to combine the knowledge.

5.2 Level 2: Engine (Whitebox)

The engine breaks down in modules `Search` and (position) `Evaluation` as shown in the following diagram. If available, the determination of the move is initially delegated to an opening book. Only if the book does not provide a standard move `Search` is used.



Engine, building block view, level 2

Module	Short description
Search	Determines the optimal move for a position under certain conditions.
Evaluation	Evaluates a position from an opponent's perspective.

5.2.1 Search (Blackbox)

Intent/Responsibility

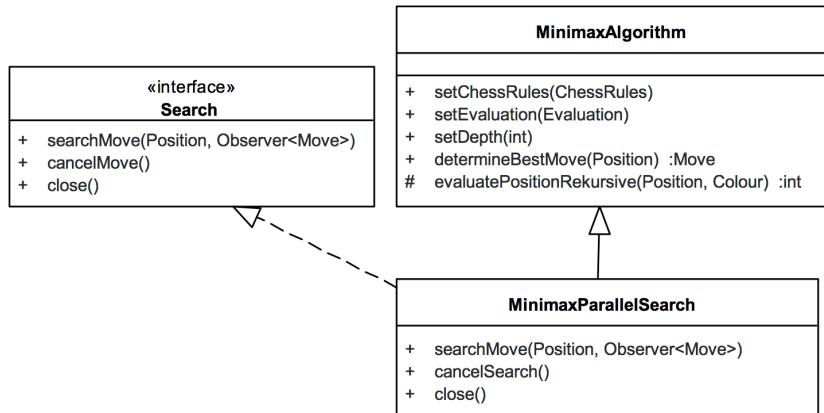
The module determines the optimal move for a position under certain conditions. In the game of chess an optimal move always exists, theoretically. The high number of possible moves and the resulting incredible mass of game situations to consider makes it impossible to determine it in practice. Common algorithms like the Minimax therefore explore the “game tree” only up to a certain depth.

Interfaces

The module provides its functionality via the Java interface `org.dokchess.engine.search.Search`.

Class `org.dokchess.engine.search.MinimaxAlgorithm` implements the interface with the Minimax algorithm. The class `MinimaxParallelSearch` uses the algorithm and implements the same interface `Search`. It examines several subtrees concurrently; if

it finds a better move the caller receives a message `onNext` via the observer pattern. The search indicates the completion of its work with the message `onComplete`.



Interface Search, classes MinimaxAlgorithm and MinimaxParallelSearch

Methods of the interface `Search`:

Method	Short description
<code>searchMove</code>	Starts a search for a move on the specified position. Returns gradually better moves as events on the passed observer. The end of the search (no better move found) is also signaled to the observer.
<code>cancelSearch</code>	Cancels the current search.
<code>close</code>	Closes the search completely. No moves may be determined after calling this method.

Methods of the class `MinimaxAlgorithm`:

Method	Short description
setEvaluation	Set the evaluation function on which the positions are rated when the maximum search depth is reached. → V.5.2.2 “Module Evaluation (Black Box)”
setChessRules	Sets an implementation of the chess rules via dependency injection, → V.5.1.2 “Subsystem Rules (Black Box)”
setDepth	Set the maximum search depth in half moves. That means at 4 each player moves twice.
determineBestMove	Determines the optimal move according to minimax for the position passed and given evaluation at fixed search depth. The method blocks and is deterministic.

Files

The implementation is located below the packages
`org.dokchess.engine.search...`

5.2.2 Evaluation (Blackbox)

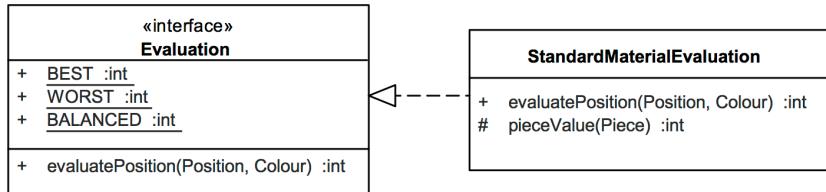
Intent/Responsibility

The module evaluates a position from an opponent's perspective. Result is a number where 0 is a balanced situation. A positive number describes an advantage for the player, a negative one a drawback. The higher the number, the greater the advantage or disadvantage. The module makes it therefore possible to compare positions with each other.

Interfaces

The Evaluation module provides its functionality via the Java interface
`org.dokchess.engine.eval.Evaluation`.

The class `org.dokchess.engine.eval.StandardMaterialEvaluation` is a very simple implementation. The interface contains constants for typical ratings.



Interface Evaluation and class StandardMaterialEvaluation

Methods of the interface Evaluation:

Method	Short description
evaluatePosition	Returns an evaluation value for the given position from the view of the specified player's colour. The higher the better.

The implementation class **StandardMaterialEvaluation** takes only the present pieces (material) into account. Each piece type has a value (pawn 1, knight 3, ..., queen 9). The pieces on the board are added accordingly. Own figures are positive, opponent's pieces negative. Accordingly, with balanced material the result is 0. If you lose a queen, the value drops by 9.

Files

The implementation is located below the packages
`org.dokchess.engine.eval...`

Open Issues

In the pure material evaluation it does not matter where the pieces stand. A pawn in starting position is worth as much as one short before promotion. And a knight on the edge corresponds to a knight in the center. There is plenty of room for improvement, which has not been exploited because DokChess should invite others to experiment.

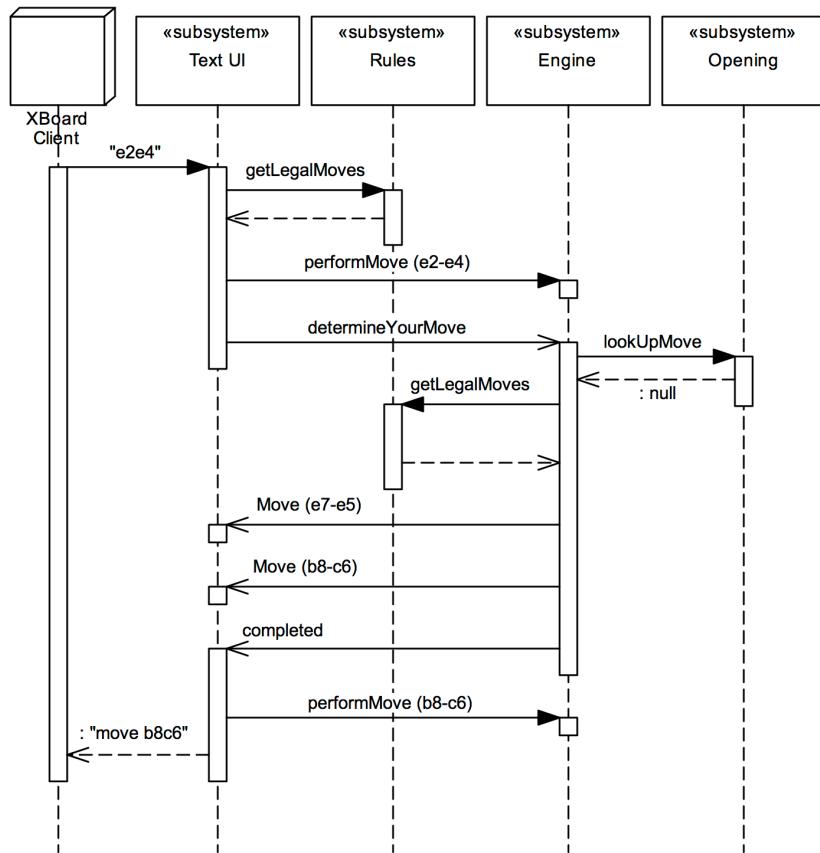
V.6 Runtime View

In contrast to the static building block view, this section visualizes dynamic aspects. How do the pieces play together?

6.1 Move Determination Walkthrough

After establishing the XBoard protocol, the client (white) starts a game by indicating a move. The following sequence diagram shows an example interaction at the subsystem level from the input “e2e4” (white pawn e2-e4) to DokChess’ response, which is the output “move b8c6” (black knight b8-c6, “[Nimzowitsch defense](#)”⁹⁶).

⁹⁶https://en.wikipedia.org/wiki/Nimzowitsch_Defence



Sample interaction for move determination

First, the *Text UI* subsystem validates the input with the aid of the *Rules* subsystem (→ Concept V.8.4 “Plausibility Checks and Validation”). The move in the example is recognized as legal and performed on the (stateful) *Engine* (the *performMove* message) afterward. Then, the *Text UI* subsystem asks the engine to determine its move. Since move computation can take a long time, but DokChess should still continue to react to inputs, this call is asynchronous. The engine comes back with possible moves.

The *Engine* examines at first whether the opening book has something to offer. In the example, this is not the case. The engine has to calculate the move on its own. It then accesses the *Rules* and determines all valid moves as candidates. Afterward, it investigates and rates them, and gradually reports better moves (better from the

perspective of the engine) back to the caller (the *Text UI* subsystem). Here, the observer pattern is used (implementation with [reactive extensions⁹⁷](#)).

The example diagram shows that two moves have been found (pawn e7-e5, knight b8-c6) and finally the message, that the search is complete, so the engine does not provide better moves. The *Text UI* subsystem takes the last move of the *Engine* and prints it as a string to standard output according to the XBoard protocol: “move b8c6”.

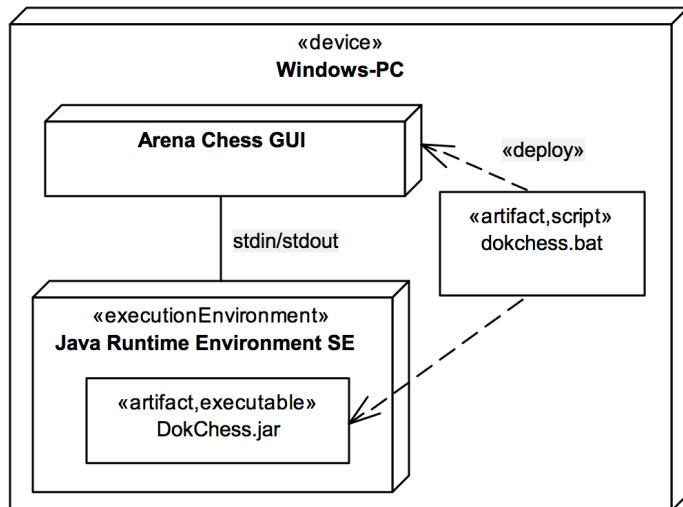
⁹⁷<https://reactivex.io/intro.html>

V.7 Deployment View

This view describes the operation of DokChess. As a Java program, it is relatively undemanding if you want to use it only on the command line. However, this is inconvenient and requires a physical chess board with coordinates, if the user cannot play blindfolded (→ image in V.4.3). Therefore, here is an explanation of how to configure DokChess in conjunction with a graphical frontend.

7.1 Windows infrastructure

The following deployment diagram shows the use of DokChess on Windows without an opening book. Arena is used as an example frontend (→ decision V.9.1 “How does the engine communicate with the outside world?”).



Deploying DokChess on a Windows PC

Software Requirements on a PC:

- Java Runtime Environment SE 7 (or higher)
- The JVM (javaw.exe) is in the *PATH*, otherwise adapt *dokchess.bat*

- [Arena⁹⁸](#)

DokChess.jar contains the compiled Java source code of all the modules and all the necessary dependencies (“Uber-jar”). The script file *dokchess.bat* starts the Java Virtual Machine with DokChess. Both are available on the computer in a common directory, because *dokchess.bat* relatively addresses the jar file.

Within Arena, the script file is declared in the following menu “Engine | Install a new Engine ...”. You will see a file selection, the file type can be limited to *.bat files. Then, set the engine type to “Winboard”. In other chess frontends, declaring an engine is very similar See corresponding documentation for details.

⁹⁸<http://www.playwitharena.de>

V.8 Cross-cutting Concepts

This section describes general structures and system-wide aspects. It also presents various technical solutions.

8.1 Dependencies Between Modules

DokChess invites developers to experiment and to extend the engine (→ V.1.2 “Quality Goals”). In order to do so, the modules are loosely coupled. DokChess modules are implementations of Java interfaces. Java classes that require parts signal this with appropriate methods *set«Module»(«Interface» ...)*. They don’t take care of resolving a dependency, for instance, by using a factory. Instead, the client resolves the dependencies by creating suitable implementations with *new* and putting them together with setter methods (also known as [Dependency Injection⁹⁹](#), short DI).

Interfaces and DI enable alternative implementations within DokChess. Adding functionality with the help of the decorator pattern (Gamma+94) is possible as well. Furthermore, aspect-oriented programming (AOP) solutions, which rely on Java dynamic proxies, are applicable interfaces. Plus, this handling of dependencies positively affects testability (→ Concept [V.8.7](#) “Testability”).

DokChess abstains from the use of a concrete DI framework. The modules are hard-wired in the code, however only in unit tests and glue code (for example, the *main* class). No annotation driven configuration is present in the code.

This gives adventurous developers free choice regarding a specific DI implementation. Since the Java modules are pure POJOs (Plain Old Java objects), nothing prevents configuration with the [Spring Framework¹⁰⁰](#) for example or CDI (Contexts and Dependency Injection for the Java EE Platform).

8.2 Chess Domain Model

“The game of chess is played between two opponents who move their pieces on a square board called a ‘chessboard’.”

from the FIDE Laws of Chess

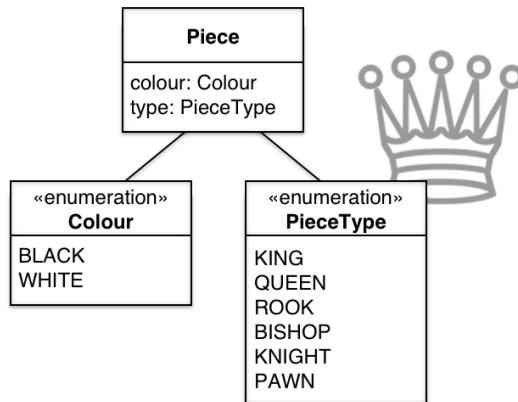
⁹⁹<https://martinfowler.com/articles/injection.html>

¹⁰⁰<https://projects.spring.io/spring-framework/>

The different modules of DokChess exchange chess-specific data. This includes the game situation on the chessboard (position) for instance, as well as opponent's and own moves. All interfaces use the same domain objects as call and return parameters.

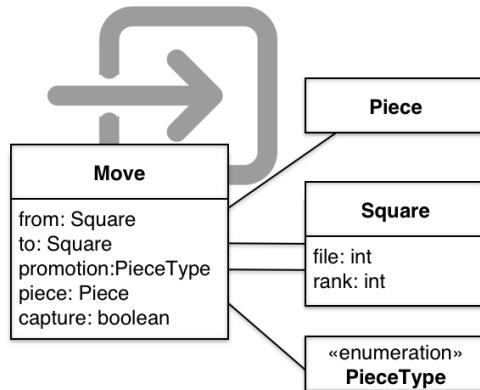
This section contains a brief overview of these data structures and their relationships. All the classes and enumeration types (enums) are located in the *org.dokchess.domain* package. See the source documentation (*javadoc*) for details.

A chess piece is characterized by colour (black or white) and type (king, queen, and so on). In the DokChess domain model, a piece does not know its location on the board. The *Piece* class is immutable, and so are all other domain classes.



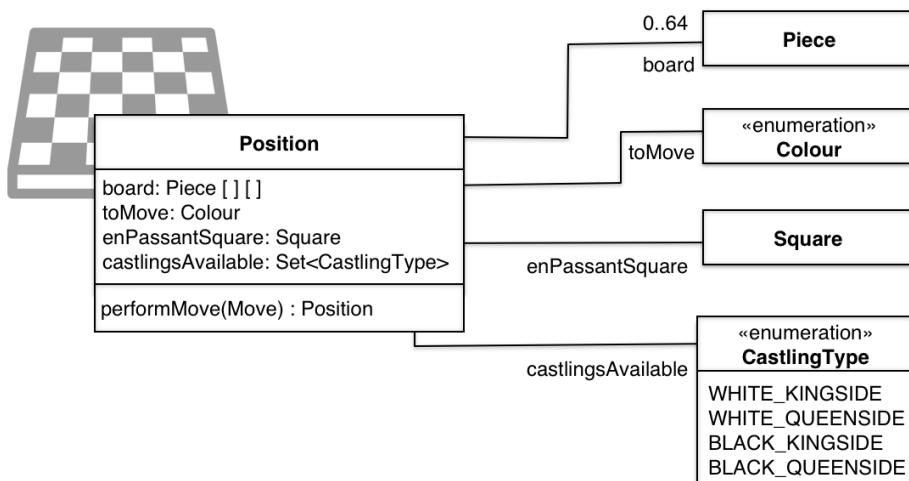
A piece has a colour and a type

A chessboard consists of 8 x 8 squares, which are arranged in 8 rows called *ranks* (1-8) and 8 columns called *files* (a-h). The *Square* class describes one of these. Since a square can be occupied by only one piece, source and target squares are sufficient to specify a move (the *Move* class). The only exception is the promotion of a pawn on the opponent's baseline. Here, the player decides which piece type they want to convert the pawn to (typically, but not necessarily, a queen). Castling moves are represented as king moves over two squares in the corresponding direction. The additional attributes for the moving piece and whether the move is a capture are useful for the analysis tasks of the engine.



A move from square to square

The *Position* class describes the current situation on the board. In particular, these are the piece locations on the board, which are internally represented as a two-dimensional array (8 x 8). If a square is not occupied, *null* is stored in the array. To complete the game situation, the *Position* class includes information about which side moves next, which castlings are still possible (if any), and whether capturing en passant is allowed.



A position describes a game state

The *Position* class is immutable as well. Therefore, the *performMove()* method returns a new position with the modified game situation (→ decision V.9.2 “Are position objects changeable or not?”).

8.3 User Interface

DokChess itself has no graphical user interface. It interacts via the XBoard protocol with the outside world (→ decision V.9.1 “How does the engine communicate with the outside world?”).

The XBoard protocol is text-based. If you have mastered the main commands (see Figure ...) starting DokChess in a command line (Unix shell, Windows command prompt, ...) enables you to interact with the engine. The table below shows a sample dialog, all commands are terminated with a new line). By default the engine plays black. You can change this with the “white” command of XBoard.

Client → DokChess	DokChess → Client	Comments
xboard		client wants to use the XBoard-Protocol (required since engines partly understand other, sometimes even multiple protocols)
	(new line)	
protover 2		protocol version 2
	feature done=1	line by line notification of additional features of the engine (here: none)
e2e4		white plays pawn e2-e42
	move b8c6	black (DokChess) plays knight b8-c6
quit		client ends the game (DokChess terminates)

Mann+2009¹⁰¹ describes the protocol itself in detail. In DokChess the Text UI subsystem is responsible for the protocol implementation (→ Building Block View, V.5.1.2 “Subsystem Text UI (Blackbox)”).

A more typical use of DokChess is a graphical chess frontend like Arena (see image

¹⁰¹<https://www.gnu.org/software/xboard/engine-intf.html>

below). It accepts the moves of the opponent in a comfortable interface, passes them to DokChess in the form of XBoard commands like in the table above (column “Client → DokChess”) and translates the answers (column “DokChess → Client”) graphically.



DokChess combined with the Arena chess frontend under Windows

8.4 Plausibility Checks and Validation

In simple terms, DokChess is an algorithm. It responds to moves of the opponent with its own moves. Two channels are relevant for input validation:

- the XBoard protocol for interactive user input from the opponent
- opening libraries in the form of files

Input that comes through the XBoard protocol is parsed from the corresponding subsystem. In case of unknown or unimplemented commands DokChess reports the XBoard command “Error” back to the client.

In case of move commands DokChess checks using the rules subsystem whether the move is allowed or not. In case of illegal moves DokChess reports the XBoard

command “Illegal move” back to the client. When using a graphical front end, this case should not occur, since these typically accept only valid moves. The case is likely relevant when interacting via command line (→ Concept V.8.3 “User Interface”).

Inputs that come through the XBoard protocol are parsed from the corresponding subsystem. For unknown or unimplemented commands DokChess reports the XBoard command “Error” back to the client. When setting up a position DokChess checks the compliance with the protocol, but not whether the position is permitted. In extreme cases (e.g. if no kings are present on the board), the engine subsystem may raise an error during the game.

For opening libraries DokChess only checks whether it can open and read the file. In case of a failure (e.g. file not found) it raises an exception (→ Concept V.8.5 “Exception and Error Handling”). While reading the file the opening subsystem responds with a runtime error to recognized problems (for example, invalid file format). However, the content of the library itself is not checked. For example, if invalid moves are stored for a position it is not recognized. The user is responsible for the quality of the library (see → V.3.1 “Business Context”). In extreme cases, the engine may respond with an illegal move.

8.5 Exception and Error Handling

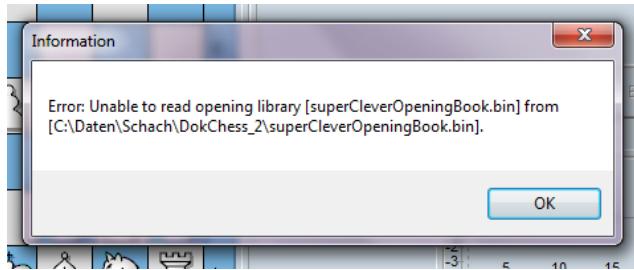
DokChess has no own user interface. Therefore it must indicate problems to the outside world.

All DokChess subsystem methods throw Runtime Exceptions. In addition in case of the *Engine* subsystem error messages (*onError*) may occur during an asynchronous search. Your own extensions (for example a custom search) must be implemented accordingly. Checked exceptions (e.g. *java.io.IOException*) need to be wrapped in Runtime Exceptions.

The javadoc of methods and constructors in question shows the rare cases where exceptions are expected in DokChess. Trouble reading an opening book file, or when trying move calculation within the engine at an invalid position (if identified). All other exceptions would be programming errors (please report them under <https://github.com/DokChess/>).

The XBoard subsystem catches all exceptions and communicates them via the XBoard protocol outwards (command “tellusererror”). A graphical front-end usually

visualizes them in an error dialog or alert box. The image below depicts that for the chess frontend Arena.



An error dialog in Arena

DokChess continues to work “normally”. The client decides whether proceeding in this specific situation makes sense. For example, it could continue to play without an opening book.

8.6 Logging and Tracing

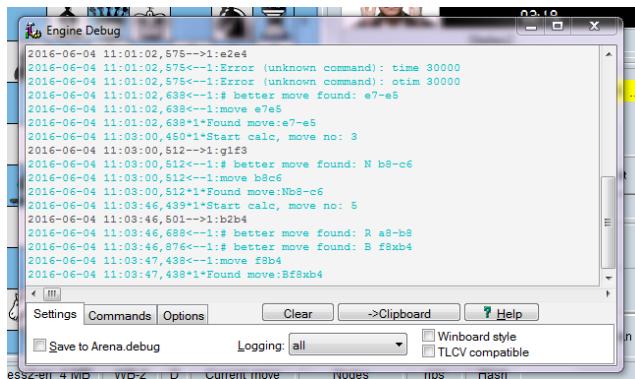
For improvements and extensions the analysis capabilities provided by DokChess are of note. Particularly in case of an error.

The DokChess functionality itself can be checked easily with unit tests. This is particularly true for the correct implementation of the game rules and for the game play of the engine (→ Concept V.8.7 “Testability”). It holds true for your own extensions in this area as well.

Therefore there is no fine-grained logging output within DokChess. Solutions like [SLF4J¹⁰²](#) are not present. In this way we avoid a dependency to an external library and the source code is not polluted by this aspect at all.

For the communication between the client and DokChess via the XBoard protocol – in addition to interactive operation via a shell (→ Concept V.8.3 “User Interface”) – the client may monitor the conversation. Common chess frontends permit this by writing log files and/or a simultaneous display of a log window during the game. The image below shows this functionality within Arena.

¹⁰²<https://www.slf4j.org>



Log and debug window for XBoard Protocol in Arena

If the engine blocks and it is unclear what went on on the XBoard protocol such tools are simply invaluable. Due to the availability of this feature a communication protocol tracing was not implemented within DokChess at all.

8.7 Testability

Nothing is more embarrassing for an engine than an illegal move.

The functionality of the individual modules of DokChess is ensured by extensive unit tests. You find a folder *src/test* next to *src/main*, where the Java source code of the modules is stored. It mirrors the package structure, and in the corresponding packages unit tests for the classes realized with [JUnit 4¹⁰³](#).

Standard unit testing, which examine the individual classes, are named as the class itself with suffix *Test*. In addition, there are tests that examine the interaction of modules, and in extreme cases the whole system. With the help of such tests, the correct playing of DokChess is checked. More complex, long-running integration tests are below *src/integTest*. This includes playing entire games.

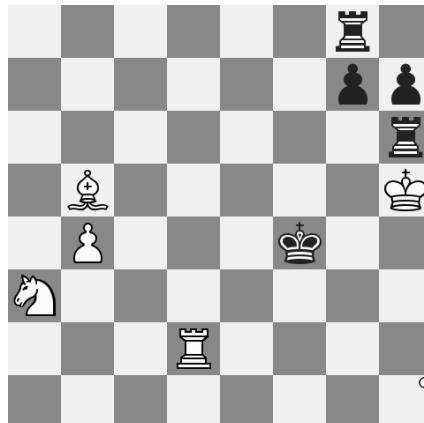
In many tests positions must be provided as call parameters. Here the Forsyth-Edwards Notation (FEN in short) is used. This notation allows the specification of a complete game situation as a compact string without a line break and is therefore perfect for use in automated tests.

The starting position in FEN for example is denoted:

¹⁰³<https://junit.org/junit4/>

1 "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"

Lowercase letters stand for black, uppercase for white pieces. For the piece types the English names (r for rook, p for pawn ...) are used.



Sample position

The game situation in the picture above with white before the 79th move, with 30 half-moves long no piece was captured and no pawn was moved, looks like this in FEN

1 "6r1/6pp/7r/1B5K/1P3k2/N7/3R4/8 w - - 30 79"

and reads “6 squares free, black rook, square free, new rank ...”.

Details about the notation can be found for example at [Wikipedia¹⁰⁴](https://en.wikipedia.org/wiki/Forsyth–Edwards_Notation). The *Position* class has a constructor that accepts a string in FEN. The *toString* method of this class also provides FEN.

¹⁰⁴https://en.wikipedia.org/wiki/Forsyth–Edwards_Notation

V.9 Architecture Decisions

This section enables you to understand two fundamental design decisions of DokChess in detail.

9.1 How Does the Engine Communicate with the Outside World?

Problem Background

As a central requirement DokChess must work together with existing chess frontends. How do we connect them?

A whole series of graphical user interfaces are available for playing against chess programs. Moreover, there are software solutions with a larger scope for chess enthusiasts. In addition to the game “Human vs. Machine”, they offer more functionality, such as analyzing games. Over time, new chess programs will be released – and others will possibly disappear from the market.

Depending on how the connection with such programs is realized, DokChess can or can not communicate with specific frontend clients. Thus, the issue affects DokChess’ interoperability with existing chess software and its adaptability to future chess software.

Influences on the Decision

Constraints

- Operation of the frontend at least on Windows desktop operating systems
- Support for freely available frontends
- Favoring established (chess) standards (→ V.2.3 “Conventions”)

Significantly Affected Quality Attributes

- Using existing frontends (Interoperability, → V.1.2 “Quality Goals”)
- Platform appealing to experiments (Changeability, → V.1.2 “Quality goals”)
- Adaptability (to future chess software)

Affected Risks

- Connecting to the Frontend fails (→ V.11.1 “Risks”)

Assumptions

- The investigation of just a few available frontends leads to all interesting integration options.

Considered Alternatives

In early 2011, the following chess frontends were investigated:

- Arena Chess GUI (available for free, runs on Windows)
- Fritz for Fun (commercially provided by ChessBase GmbH, runs on Windows)
- Winboard / XBoard (Open source, runs on Windows, Mac OS X, *nix)

As a result, two communication protocols have been identified as options:

- Option 1: UCI Protocol (universal chess interface), see [MeyerKahlen2004]
- Option 2: XBoard Protocol (also known as Winboard and as Chess Engine Communication Protocol), see [Mann+2009]

Neither of the two protocols are formally specified, but both are publicly documented.

Both protocols are text-based, and communication between the frontend and the engine is via stdin/stdout. In both cases the front end starts the engine in a separate process.

The following Table compares the three investigated frontends to their implemented protocols:

.	Arena 3	Fritz for fun	Winboard / XBoard
UCI	Yes	Yes	-
XBoard-Protocol	Yes	-	Yes

Decision

Under the given constraints, the quality goals can generally be achieved with both options. Depending on which protocol is implemented, different front ends are supported.

The decision in favor of the XBoard protocol was made in early 2011. The structure of DokChess allows us to add alternative communication protocols (UCI or other) without having to change the engine itself for this, see dependencies in the building block view (→ [V.5.1 “Building Block View, Level 1”](#)).

The preferred front end for Windows is Arena. It is freely available and tops the functionality of WinBoard. It has good debugging facilities. For example, it can present the communication between the frontend and the engine live in a window (→ screenshot in V.8). Arena supports both protocols.

By opting for the XBoard protocol, other operating systems (in addition to Windows, especially Mac OS X and Linux) are also supported with freely available frontends (see the preceding table). As such, a larger circle of interested developers may use the engine, which was finally essential.

9.2 Are Position Objects Changeable or Not?

Problem Background

For various DokChess modules, game situations on the chess board (so-called positions) must be provided and exchanged between them. Do we make the associated data structure changeable or unchangeable (immutable)?

During a game, the position changes when the opposing players move their pieces. In addition, the engine performs possible moves in its analysis, tries to counter the moves of the opponent, evaluates the outcoming positions, and then discards moves. The result is a tree that contains many thousands of different positions, depending on the search depth.

Depending on whether the position is immutable as a data structure or not, algorithms are easier or more difficult to implement, and its execution is efficient in a different way.

All modules depend on the position interface. A subsequent change would affect all of DokChess.

Influences on the Decision

Constraints

- Implementation in Java (→ [V.2.1 “Technical Constraints”](#))
- Moderate hardware equipment (→ [V.2.1 “Technical Constraints”](#))

Significantly Affected Quality Attributes

- Platform appealing to experiments (Changeability, → [V.1.2 “Quality Goals”](#))
- Acceptable playing strength (Attractiveness, → [V.1.2 “Quality Goals”](#))
- Quick response to opponent’s moves (Efficiency, → [V.1.2 “Quality Goals”](#))

Affected Risks

- Implementation effort too high (→ [V.11.2 “Risks”](#))
- Achieving the playing strength fails (→ [V.11.3 “Risks”](#))

Assumptions

- It is possible to implement a data structure with a fine-grained object model (that is, classes such as Piece and Move) that is efficient enough to provide the required playing strength within an adequate response time.
- In the future, concurrent algorithms should be implemented with the data structure as well.

Considered Alternatives

The starting point is domain-driven classes for square, piece and move (→ Concept [V.8.2 “Chess Domain Model”](#)). These classes are realized immutable as value objects (the e4 field always remains e4 after it’s construction).

For the position, two alternatives are considered:

- **Option (1): The position is changeable.** Individual methods of the interface, for example, performing moves or taking them back, change the object state.

Pseudo code for option 1:

```

1 Position p = new Position(); // starting position, white side to move
2 p.performMove(e2e4);           // king's pawn advances two squares, black\
3   to move
4 p.takeBackLastMove()          // starting position again

```

- **Option (2): The position is unchangeable (“immutable”).** That means a method for performing a move provides the new position (it copies the old one, then the move happens) as another immutable object as the result.

Pseudocode for option 2:

```

1 Position p = new Position();
2 newPosition = p.performMove(e2e4); // p keeps unchanged

```

The following table summarizes the strengths and weaknesses of the two options, they are explained below.

.	(1) changeable	(2) unchangeable
Implementation effort	(-) higher	(+) lower
Efficiency (memory consumption)	(+) more economical	(-) higher demand
Efficiency (time behaviour)	(o) neutral	(-) worse
Suitability for concurrent algorithms	(-) bad	(+) good

Option (1): The Position is Changeable

(+) Positive Arguments

The position with its extensive state is not copied in each move. This saves memory

and CPU time, and it treats the garbage collector with care. For analysis algorithms, however, it is necessary to implement functionality that takes back the effect of moves (“undo”). This Undo also takes time, hence the neutral rating (**0**) for the time behavior.

(-) Negative Arguments

The implementation of an Undo-functionality is complex. Not only does it have to set up captured pieces back on the board, the castling rule and en passant require special treatment as well. The Gamma+94 Command pattern suggests itself as an option. The use of that pattern within algorithms is also more complex because the latter must explicitly call methods to take back the moves.

Finally, changeable state has drawbacks related to concurrency.

Option (2): The Position is Unchangeable

(+) Positive Arguments

When you perform a move, the position is copied. It does not change the original. This eliminates the implementation of move reversal (Undo). Clients can simply store the old position as a value. This saves effort in the implementation compared to option (1).

Immutable objects offer significant advantages in concurrent algorithms.

(-) Negative Arguments

Copying the state for each new position takes time. When analyzing situations, it covers many positions, which can take a lot of time.

Beyond that, copying the state for each new position costs memory. The implementation of search algorithms with backtracking at least prevents complete game trees from ending up on the heap. Nevertheless, more memory is required, and the garbage collector has more work to do.

Both points have a negative effect on efficiency.

Decision

The decision for the unchangeable position (option 2) was made in early 2011 due to the advantages in terms of ease of implementation and the prospect of exploiting concurrency. All the disadvantages of option 2 are related to efficiency.

Due to the risk of failing to achieve the objectives with respect to the playing strength in an acceptable computing time (attractiveness, efficiency), prototypes of both versions have been implemented and compared in a mate search (Checkmate in three moves) with the minimax algorithm. With option 2, the search took 30% longer, assuming you implement copying efficiently. But this option met the constraints still well.

Further optimization options could reduce the efficiency disadvantage compared with option 1, if necessary. But they have not been implemented in the first run in order to keep the implementation simple.

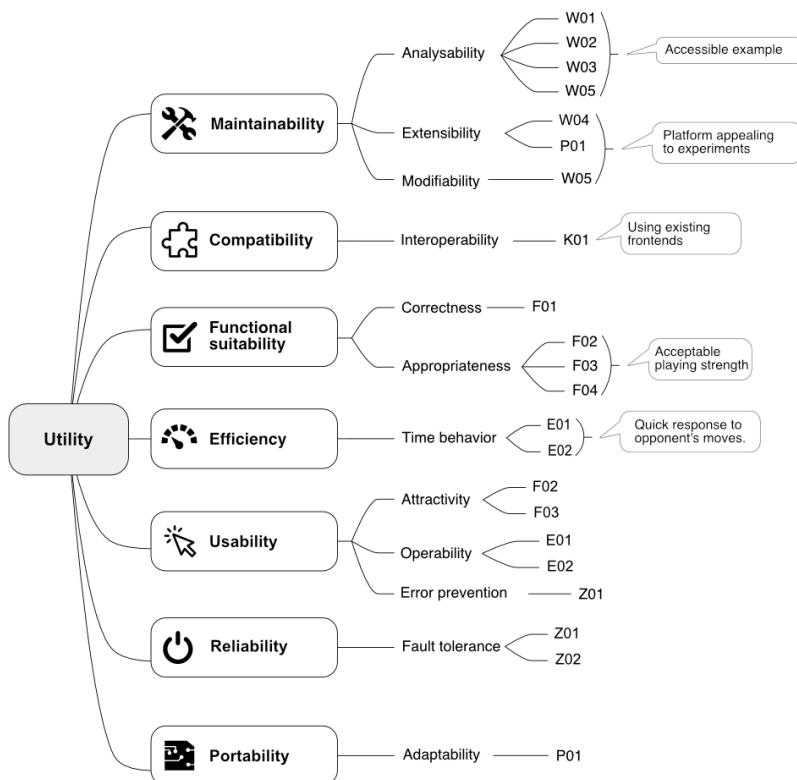
These options included the use of multiple processors/cores with concurrent algorithms. In the meantime (with DokChess 2.0), this was illustrated with a parallel minimax example.

V.10 Quality Requirements

The quality scenarios in this section depict the fundamental quality goals of DokChess (→ V.1.2) as well as other required quality properties. They allow the evaluation of decision alternatives.

10.1 Utility Tree

The following diagram gives an overview of the relevant quality attributes and their associated scenarios (→ table in V.10.2). The quality goals are also included in the figure, and each refers to the scenarios that illustrate them.



DokChess Utility Tree

10.2 Quality Scenarios

The initial letters of the identifiers (IDs) of the scenarios in the following table each stand for the parent quality attribute (in German), E for efficiency, for instance. These identifiers are also used in the utility tree (→ V.10.1). The scenarios cannot always be clearly assigned to one characteristic. Therefore, they sometimes appear several times in the utility tree.

No.	Scenario
W01	A person with basic knowledge of UML and chess looks for an introduction to the DokChess architecture. He or she gets the idea of the solution strategy and the essential design within 15 minutes.
W02	An architect who wishes to apply arc42 searches for a concrete example for an arbitrary section of the template. He or she finds the relevant content immediately in the documentation.
W03	An experienced Java developer searches for the implementation of a module described in the design. He or she finds it in the source code without detours or help from others.
W04	A developer implements a new position evaluation. He or she integrates it into the existing strategies without modification and without compilation of existing source code.
W05	A developer implements a piece-centric bitboard representation of the game situation. The effort (which includes replacing the existing, square-centric representation) is lower than one person week.
K01	A user plans to use DokChess with a frontend that supports a communication protocol that's already implemented by the solution. The integration does not require any programming effort, the configuration with the front end is carried out and tested within 10 minutes.
F01	In a game situation, the engine has one or more rule-compliant moves to choose from. It plays one of them.
F02	A weak player is in a game against the engine. The player moves a piece to an unguarded position which is being attacked by the engine. The engine consequently takes the dropped piece.

No.	Scenario
F03	Within a chess match, a knight fork to gain the queen or a rook arises for the engine. The engine gains the queen (or the rook) in exchange for the knight.
F04	Within a chess match, a mate in two unfolds to the engine. The engine moves safely to win the game.
E01	During the game, the engine responds to the move of the opponent with its own move within 5 seconds.
E02	An engine integrated in a graphical frontend plays as black, and the human player begins. The engine responds within 10 seconds with its first move, and the user gets a message that the engine “thinks” within 5 seconds.
Z01	During the game, the engine receives an invalid move. The engine rejects this move and allows the input of another move thereafter and plays on in an error-free way.
Z02	The engine receives an illegal position to start the game. The behavior of the engine is undefined, cancelling of the game is allowed, as well as invalid moves.
P01	A Java programmer plans to use a chess frontend that allows the integration of engines, but does not support any of the protocols implemented by DokChess. The new protocol can be implemented without changing the existing code, and the engine can then be integrated as usual.

V.11 Risks and Technical Debts

The following risks have been identified at the beginning of the project. They influenced the planning of the first three iterations significantly. Since the completion of the third iteration the risks are mastered. This architectural overview shows the risks, including the former eventuality planning, because of their large impact on the solution.

11.1 Risk: Connecting to the Frontend fails

There is no knowledge about connecting an engine to an existing chess frontend. Available open source engines are programmed in C and are delivered as executable programs (*.exe). Since DokChess is developed in Java, they are of limited use as examples. Nothing is known in the project about chess communication protocols.

If it is not possible to make a working connection, the solution can not be used with existing frontends. This not only lacks an important feature (→ V.1.1 “Requirements Overview”), but also makes the solution as a whole, especially as a case study, untrustworthy.

Contingency Planning

A simple textual user interface could be implemented in order to interact with the engine. The implementation of a DokChess specific graphical front end would be costly (→ Risk V.11.2 “Effort of Implementation”).

Risk Mitigation

Through an early proof of concept, certainty is achieved as soon as possible here.

11.2 Risk: Implementation effort too high

There is no experience with chess programming. Simultaneously, the rules that are to be implemented completely (see V.1.1 “Requirements Overview”) are extensive and complicated. The different pieces move differently, and there are special rules such

as stalemate and promotion. In the case of castling and en passant, the move history, and not only the current situation on the board, is relevant.

The programming of the algorithms is also non-trivial. For the connection of opening libraries and endgame databases, extensive research is required.

The implementation of DokChess runs as a hobby alongside, within the spare time. It is unclear whether this is sufficient to present ambitious results within schedule (→ constraints V.2.2).

Contingency Planning

If there is no runnable version for the conference sessions in March and September 2011, a live demonstration could be omitted. The free evening talk at oose in March could even be canceled completely (which could negatively affect the reputation).

Risk Mitigation

In order to reduce effort, the following rules are not implemented at first:

- 50 moves rule
- threefold repetition

Their absence has little consequence with respect to the playing strength, and consequence no consequence with respect to the correctness of the engine.

Connecting opening libraries and endgame databases has low priority and takes a back seat at first.

11.3 Risk: Achieving the playing strength fails

The quality goals demand both an acceptable playing strength and a simple, accessible solution. In addition, there are requirements with respect to efficiency. It is uncertain whether the planned Java solution with an object-oriented domain model and simple move selection can achieve these competing goals.

Contingency Planning

In conference talks, parts of the live demonstration could be omitted. If necessary, we could show games played before.

Risk Mitigation

Suitable scenarios (examples → [V.10](#) “Quality Scenarios”) concretize the quality goals. With the help of chess literature (chess problems to be precise) we develop test cases (unit testing), which specify what playing strength can be expected. Thus at least we detect early where the engine stands.

V.12 Glossary

“The game of chess is played between two opponents who move their pieces on a square board called a ‘chessboard’.”

From the FIDE Laws of Chess

The following glossary contains English chess terms and terms from the world of computer chess. Some of them go beyond the vocabulary of infrequent or casual chess players.

See [FIDE Laws of Chess¹⁰⁵](https://www.fide.com/component/handbook/?id=208&view=article) or the [Wikipedia glossary of chess¹⁰⁶](https://en.wikipedia.org/wiki/Glossary_of_chess) for more information.

Names of the Chess Pieces



Chess pieces (or chessmen)

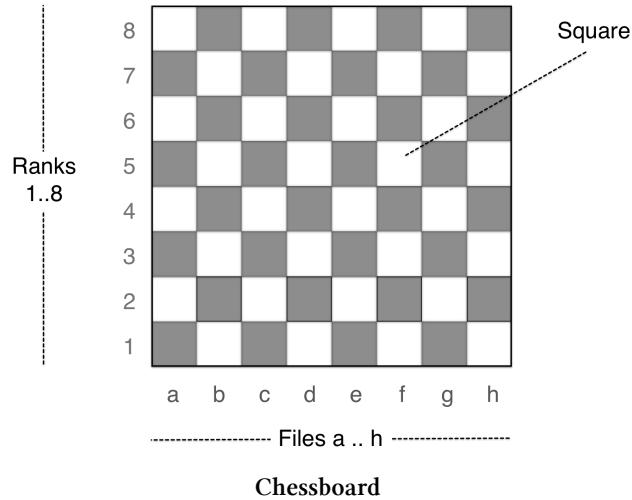
¹⁰⁵<https://www.fide.com/component/handbook/?id=208&view=article>

¹⁰⁶https://en.wikipedia.org/wiki/Glossary_of_chess

Chessboard Geometry

“The chessboard is composed of an 8 x 8 grid of 64 equal squares alternately light (the ‘white’ squares) and dark (the ‘black’ squares).”

From the FIDE Laws of Chess



Chess Terms

Term	Definition
50 move rule	A rule in chess, which states that a player can claim a <i>draw</i> after 50 moves whilst in the meantime no pawn has been moved and no piece has been taken.
Alpha-beta pruning	Significant improvement of the <i>Minimax algorithm</i> , in which parts of the search tree can be “cut off” without coming to a different resulting move.
Castling	A special move in chess where both the players’s king and one the rooks are moved. For castling, different conditions must be met.
Chess 960	A chess variant developed by Bobby Fischer. The initial position is drawn from 960 possibilities. Also known as Fischer Random Chess.
to drop (a piece)	A beginner’s mistake in chess. A piece is dropped, if it is moved to a field attacked by the opponent and can be taken by him or her without risk or disadvantage.
Draw	A chess game which ends with no winner. There are various ways to end it with a draw, one of them is <i>stalemate</i> .
Half-move	Single action (move) of an individual player, unlike the sequence of a white and a black move which is counted as a move e.g. when numbering.
En passant	A special pawn move in chess. If a pawn moves on two squares and an opposing pawn could capture him, if he would advance only one square, the latter pawn may capture it en passant.
Endgame	The final stage of a chess game. It is characterised when only a few types of pieces are left on the board.
Engine	Part of a chess program which calculates the moves. Typically, an engine has no graphical user interface. Also known as ‘chess engine’.

Term	Definition
FEN	Forsyth-Edwards Notation. Compact representation of a chess board position as a character string. Supported by many chess tools. Used in DokChess by unit and integration tests. Explanation see e.g. [Wikipedia] (https://en.wikipedia.org/wiki/Forsyth–Edwards_Notation)
FIDE	Fédération Internationale des Échecs, international chess organisation
Fork	A tactic in chess in which a piece attacks two (or more) of the opponent's pieces simultaneously.
Knight fork	Particularly common form of a <i>fork</i> with a knight as attacking piece.
Mate	End of a chess game in which the king of the player to move is attacked and the player has no valid move left (i.e. can not escape the attack). The player has lost. Also known as 'checkmate'.
Minimax algorithm	Algorithm to determine the best move under the consideration of all options of both players.
Opening	First stage of a chess game. Knowledge and best practices of the first moves in chess fill many books and large databases.
Polyglot Opening Book	Binary file format for <i>opening</i> libraries. Unlike many other formats for this purpose the documentation of it is freely accessible.
Promotion	A rule in chess which states that a pawn who reached the opponent's base line is immediately converted into a queen, rook, bishop or knight.
Skewer	A tactic in chess in which a straight line passing piece stands on a rank, file or diagonal with two opponents pieces and forces the fore of the two pieces to move away.

Term	Definition
Stalemate	End of a chess game in which the player to move does not have a valid move, but his or her king is not under attack. The game is considered a <i>draw</i> .
Threefold repetition	A rule in chess, which states that a player can claim a <i>draw</i> if the same position occurs at least three times.
WinBoard protocol	See <i>XBoard protocol</i> .
XBoard protocol	Text-based protocol for communication between chess frontends and <i>engines</i> . Also referred to as “WinBoard” or (more rarely) as “Chess Engine Communication Protocol”.

VI - docToolchain

By [Ralf D. Müller](#).



[docToolchain](#)¹⁰⁷ started out as the `arc42-generator`: a Gradle build which converts the `arc42-template` from AsciiDoc to various other formats like Word, HTML, ePub, and Confluence.

While working with AsciiDoc, I came up with several other little scripted tasks, which helped me in my workflow. These were added to the original `arc42-generator` and turned it into what I now call `docToolchain`.

The goal of `docToolchain` is to support you as a technical writer with some repeating conversion tasks and to enable you to implement the - sometimes quite complex - [Docs-as-Code](#)¹⁰⁸ approach more easily.

I derived the name *docToolchain* from the fact that it implements a documentation toolchain. Several tools like Asciidoctor, Pandoc, MS Visio (only to name a few) are glued together via Gradle to form a chain or process for all your documentation needs.

The following chapters describe the architecture of `docToolchain`. The whole project, together with its remaining documentation, is hosted on Github:

<https://doctoolchain.github.io/docToolchain/>

¹⁰⁷<https://doctoolchain.github.io/docToolchain/>

¹⁰⁸<https://docs-as-code>

VI.1 Introduction and Goals

The need to deliver an arc42 solution architecture in different formats was the primary driver which gave birth to docToolchain. As a technical person, I saw the value of text-based documentation. However, other stakeholders wanted to have the documentation

- “*where all other documentation is*” - in a Wiki
- “*how we always document*” - with MS Word
- “*in a revision safe format*” - as PDF

Soon after fulfilling these needs, I noticed that it is quite hard to update changing UML diagrams. “Couldn’t this be automated?” I thought to myself and created the `exportEA`-task as a simple Visual Basic script.

I didn’t plan docToolchain on the drawing board. It evolved in an agile way - through the requirements of the users:

Together with Gernot Starke, I gave a talk about this approach, and I got feedback from the audience like “... *but I don’t use Enterprise Architect - can I also use MS Visio?*”.

My answer was something like

“*no, not at the moment. But docToolchain is not only a tool. I want it to be an idea - a change in how you think. If EA can be easily automated, I am sure Visio can be automated too!*”

What I didn’t expect was, that a few days after this statement, we got a pull request with the `exportVisio`-Task!

This way, docToolchain evolved through the requirements and imagination of its users.

1.1 Problem Statement

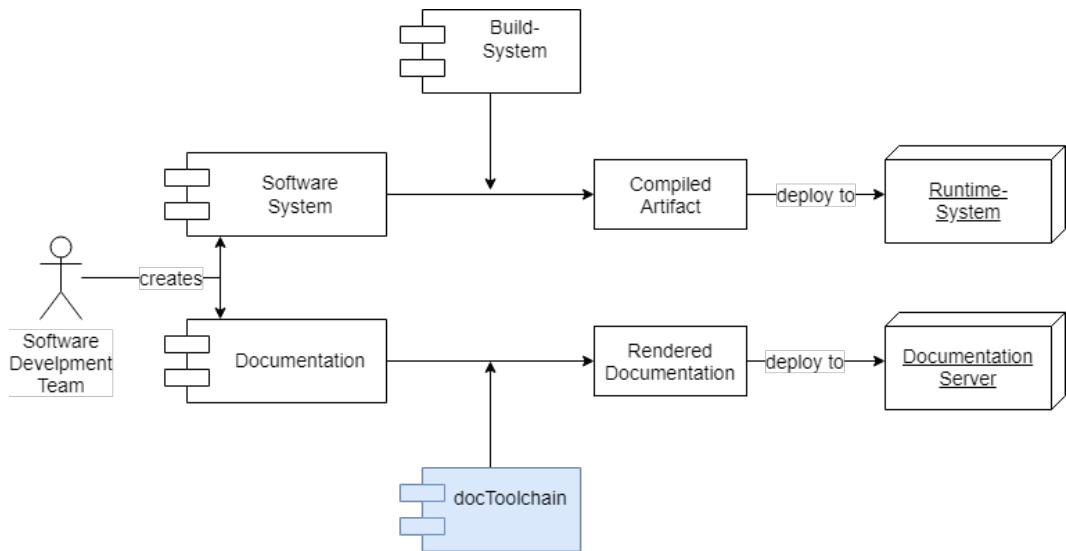


Figure 1: Problem Statement

When a software development team creates a software system, a build system is used to create a compiled artifact which is then deployed to the runtime system (see figure 1).

You can apply the same approach to your documentation. This is called the Docs-as-Code approach, e.g., you treat your documentation the same way as you treat your code. To do so, you need several helper-scripts to automate repeating tasks for building your documentation and it is a waste of time to set up these tasks for each and every project from scratch.

docToolchain - the system described in this documentation - solves this problem by providing a set of build scripts which you can easily install. They provide you with a set of tasks to solve your standard Docs-as-Code needs.

The functional requirements are mainly split up into two different kinds of functionality:

- Tasks to export content from proprietary file formats to be used in AsciiDoc
- Tasks to generate or convert different output formats

Translated into architectural requirements:

RQ1 - build system integration

The system should be easy to integrate into a standard software build to treat the docs the same way as you treat your code.

R2 - easy to modify and extend

docToolchain should be easy to modify and extend.

There are many different functional requirements to a documentation toolchain - every user should be able to fit it to his/her needs.

RQ3 - enterprise ready

The project should run in different enterprise environments. It has to run with different version control systems and without direct access to the internet (only via proxies, e.g., for dependency resolution)

Documentation is not only crucial for open source projects but also essential for enterprise projects. (Note: I also listed this as constraint C3)

RQ4 - OS independent

The toolchain should run on Windows, Linux, and macOS.

This enables Developers to use docToolchain on the operating system of their choice.

RQ5 - Maven and Gradle¹⁰⁹

The toolchain should be available for the two major build systems, Maven and Gradle.

This ensures that most JVM based projects can use it.

¹⁰⁹dropped for now (only Gradle is implemented) - see Design Decision 3. Workaround through a mixed build is available.

RQ6 - clone, build, test, run

The project should be set up in such a way that it runs right after you've cloned the repository.

This is to avoid the need for a special setup to be able to use it or change project code.

1.2 Quality Goals

The main quality goals are the following:

Confidentiality

This quality goal is not docToolchain specific but regarding security very important.

Because the system is not only targeted for open source environments but also for enterprise ones, it has to be ensured that no data is sent to 3rd-party systems without the user knowing it. For example, there are server-based text analysis tools and libraries which send the text to be analyzed to a 3rd party server. Tools and libraries like these must not be used.

Ease of Use

The docToolchain project is still a young open-source project and hence needs a stable and growing user base. The easiest way to achieve this is to make docToolchain extremely easy to use.

Maintainability

There is no big company behind docToolchain - only users who evolve it in their spare time. So, a developer should be able to make most changes needed to evolve docToolchain in a time-box of one to three hours.

1.3 Stakeholders

OSS Contributors:¹¹⁰ need to be aware of the architecture, to follow the decisions, and to evolve the system in the right direction.

¹¹⁰<https://github.com/docToolchain/docToolchain/graphs/contributors>

Contributors are mostly users of the system who need to fit a docToolchain task to their needs or need an additional task.

General docToolchain users: These are mainly software architects and technical writers with a development background. It is very likely that a docToolchain user also uses arc42 and is familiar with it.

The current focus is on users who are software architects in a corporate environment and work on the JVM with Git and Gradle.

These users need to include architectural diagrams in their documentation and work in a restricted environment (proxies, local maven repositories, restricted internet access, etc.).

They also need to be able to create different output formats for different stakeholders.

Users not working with the JVM: docToolchain also caught the attention of users who do not primarily work on the JVM. These users need to use docToolchain, just like any other command-line tool.

Readers: the main purpose of documentation is to be read by someone who needs specific information.

VI.2 Constraints

C1 - Run everywhere

docToolchain has to run on all modern operating systems which users need for development, e.g., Windows, Linux, macOS.

C2 - build for the JVM

docToolchain is built for the JVM. While some of the features provided might also work together with other environments, this can't be guaranteed for all tasks.

C3 - enterprise ready

docToolchain has to run in restricted enterprise environments. It has to run with proxies, minimal dependencies, restricted internet access, etc.

I also listed this as a requirement [RQ3](#)

C4 - headless

docToolchain has to run headless, e.g., without a display, to run on build servers.

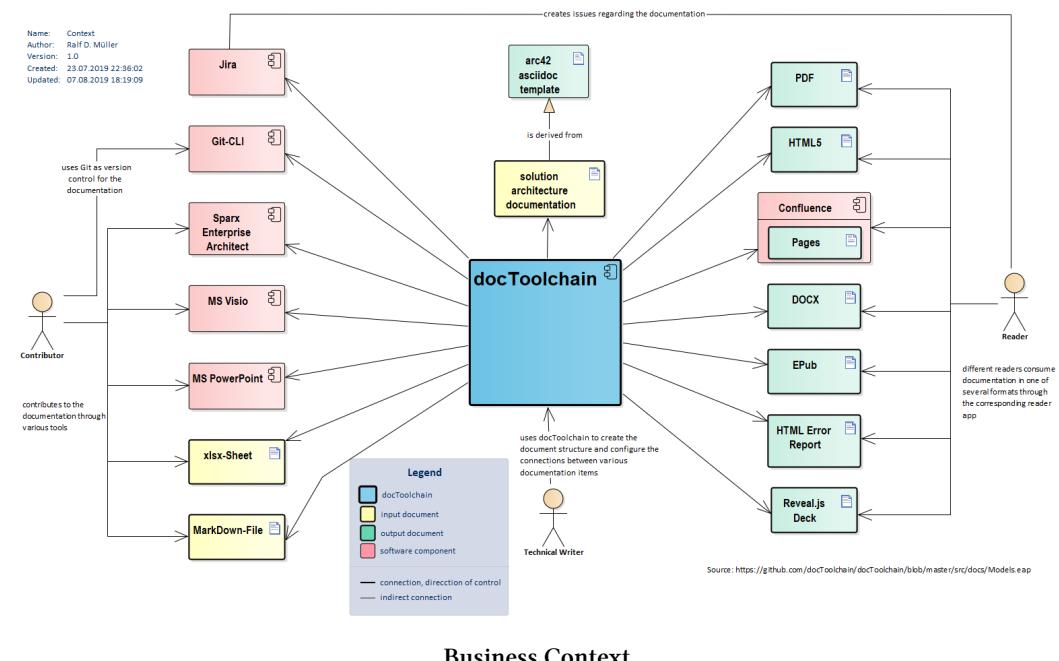
User interaction and windows features have to be avoided.

VI.3 System Scope and Context

This chapter describes the context for which docToolchain is designed and the functional scope of it.

Business Context

The following diagram shows the context in which docToolchain operates:



Business Context

The above diagram is correct regarding the connections from docToolchain, but abstracted regarding the relationships from the two actors “Contributor” and “Reader”:

Those actors represent users who contribute to the documentation or access it. The connections shown between those actors and the neighbor systems are abstract because these users do not directly access the files. And they might use different applications to read and modify the content.

docToolchain itself uses exactly the applications shown or reads and modifies the files directly.

It creates the output documents directly as file content. Only for Confluence, the Confluence API and thus Confluence as a System is used.

To read the input documents, docToolchain often needs an Application which can be remote-controlled to convert and read the documents. But it directly reads xlsx- and MarkDown-files.

The Actors represent roles. Each user can access the system through all of those roles. Each user will likely have a primary role.

The Reader and Contributor roles don't have a direct connection to docToolchain. They may read and modify content through different applications than docToolchain uses to access the content.

Actor / Neighbor	Description
Technical Writer (user)	These are the users who use docToolchain to manage their documentation. They create the document structure and configure the connection between different documentation items mainly via include¹¹¹ - and image¹¹² -directives in AsciiDoc.
Contributor (user)	These are the users who contribute to the documentation of the Technical Writer not necessarily knowing of docToolchain. They may use different applications to modify the content.
Reader (user)	These are the users who consume the documentation created by the Technical Writer and additional contributors. They use several reader applications for this task.
Jira (external system)	Issue tracking system from which docToolchain may export issues to be included in the documentation

¹¹¹<https://asciidoc.org/docs/user-manual/#include-directive>

¹¹²<https://asciidoc.org/docs/user-manual/#images>

Actor / Neighbor	Description
Git-Repository (external system)	Version control system from which docToolchain may export change log information to be included in the documentation
Sparx Enterprise Architect (external system)	UML modeling tool from which docToolchain may export diagrams (as images) and element notes to be included in the documentation
MS Visio (external system)	Diagramming tool from which docToolchain may export diagrams (as images) and notes to be included in the documentation
MS PowerPoint (external system)	Presentation tool from which docToolchain may export slides (as images) and speaker notes to be included in the documentation
xlsx-sheet (input document)	Spreadsheet document which may be transformed into AsciiDoc by docToolchain to be included in the documentation
MarkDown-File (input document)	Document which may be transformed into AsciiDoc by docToolchain to be included in the documentation
your solution architecture documentation (AsciiDoc-document)	The main document which references all other parts of your documentation through <code>include</code> -statements. For a solution architecture, this document is derived from the arc42-template
arc42 template (as AsciiDoc-document)	The template that helps you to structure the documentation of your solution architecture
docToolchain (system)	the system that is described here

Actor / Neighbor	Description
PDF (output file)	One of several output formats in which docToolchain can render documentation
HTML5 (output file)	One of several output formats in which docToolchain can render documentation
Confluence (external system)	The well known Wiki to which docToolchain can publish documentation
DOCX (output file)	One of several output formats in which docToolchain can render documentation
EPub (output file)	One of several output formats in which docToolchain can render documentation
HTML Error Report (output file)	To check the quality of the documentation, docToolchain can create an error report which contains the results of some syntax check on the generated HTML documents
Reveal.js Deck (output file)	One of several output formats in which docToolchain can render documentation

Technical Context

This section describes the technical interfaces to the components listed above.

System	Interface
Jira (external system)	REST-API
Git-Repository (external system)	Shell execution of commands
Sparx Enterprise Architect (external system)	COM-API through execution of a Visual Basic Script
MS Visio (external system)	COM-API through execution of a Visual Basic Script
MS PowerPoint (external system)	COM-API through execution of a Visual Basic Script

System	Interface
xlsx-sheet (input document)	Apache POI Java library
MarkDown-File (input document)	nl.jworks.markdown_to_asciidoc Java library
PDF (output file)	Asciidoctor PDF Plugin
HTML5 (output file)	Asciidoctor
Confluence (external system)	REST-API
DOCX (output file)	conversion of docBook file (generated by Asciidoctor) via pandoc
EPub (output file)	conversion of docBook file (generated by Asciidoctor) via pandoc
HTML Error Report (output file)	via htmlSanityCheck
Reveal.js Deck (output file)	additional Asciidoctor backend

Scope

This section explains the scope of docToolchain, e.g., what types of tasks docToolchain is responsible for and also what it is not responsible for.

docToolchain:

- is a collection of tasks to make creating and maintaining documentation easier
- tries to delegate tasks to other available tools wherever possible

docToolchain does **not**:

- re-implement tasks of other available tools
- modify manually created content
- serve the generated content and is not a build server for docs

VI.4 Solution Strategy

This chapter describes the strategy how we approached problems in the past and how we will approach them in the future.

General Solution Strategy

The first docToolchain tasks evolved from stand-alone scripts which were executed from the command-line. The next logical step was to include them in a build system.

In Gradle, it is straightforward to include a Groovy-based script as a new task:

```
1 task newTask (
2     description: 'just a demo',
3     group: 'docToolchain'
4 ) {
5     doLast {
6         // here you can put any Groovy script
7     }
8 }
```

The original `build.gradle` soon grew into a huge single file. To make it more modular, we split it up into so-called [script-plugins](#)¹¹³. You extract tasks which belong together in an extra `.gradle` file (for docToolchain you will find those in the `/scripts` folder) and reference them from the main `build.gradle`:

```
1 apply from: `script/exportEA.gradle`
```

To make docToolchain easier to use, it is currently in discussion to move those script plugins to binary plugins. These plugins reference a pre-compiled `.jar` file from a repository instead of downloaded scripts.

Individual Solution Strategy

The solution strategy for individual tasks can be broken up into two main classes of tasks.

¹¹³https://docs.gradle.org/current/userguide/plugins.html#sec:script_plugins

Export Tasks

These are tasks which export diagrams and plain text from other tools. All tools are different, so most of them need individual solution strategies.

These tools save their data often in a proprietary binary data format, which is not easy to put under version control.

The solution strategy for export tasks is first to try and read the file format directly to export the data. An excellent example of this is the `exportExcel` task, which uses the [Apache POI library¹¹⁴](#) to access the file directly. This solution makes the task independent of MS Excel itself.

However, there are tools where the file format can't be read directly, or the effort is too high. In these cases, the solution strategy is to automate the tool itself. An excellent example of this case is the `exportEA` task where the task starts Enterprise Architect in an invisible, headless mode and exports all diagrams and notes through the COM interface. The drawbacks are that this task only runs on Windows systems. It can also be slow, and while Enterprise Architect is invisible, it still claims the input focus so that you can't work with the machine during export. However, besides those drawbacks, the automation is still valuable.

A third kind of tools from which docToolchain exports data are web-based tools like Atlassian Jira. The obvious solution is to use the available REST-API to export it. There is also a Java-API which wraps the REST-API, but the direct REST-API is preferred.

Version Control of exported Artifacts

Most export tasks are quite slow. At least so slow that you don't want to run them every time you generate your documentation. In addition, some of the exports (like `exportEA`) don't run on a Linux-based build server. Because of this, the general strategy is to put the exported artifacts under version control. The tasks create Special folders into which they place - among the exported data - a `README.adoc` which warns the user that the files in the folder will be overwritten with the next export.

It first sounds strange to put exported or generated artifacts under version control, but you even get an additional benefit. The exported files - in contrast to their proprietary source files - can be easily compared and reviewed!

¹¹⁴<https://poi.apache.org/>

When a software system has a longer lifetime, it might also occur that some of the input systems get outdated. This may result in unreadable source files. In these cases, the version-controlled artifacts might also be quite useful since they are in easy to read text-only or image formats.

Generate Tasks

These are tasks which generate different output formats from the sources.

For these tasks, the general solution strategy should be to implement them as additional Asciidoctor backend (as reveal.js does for example) or at least as a plugin (as the PDF plugin does for example). However, the effort to create an Asciidoctor backend is, because of missing documentation, quite high.

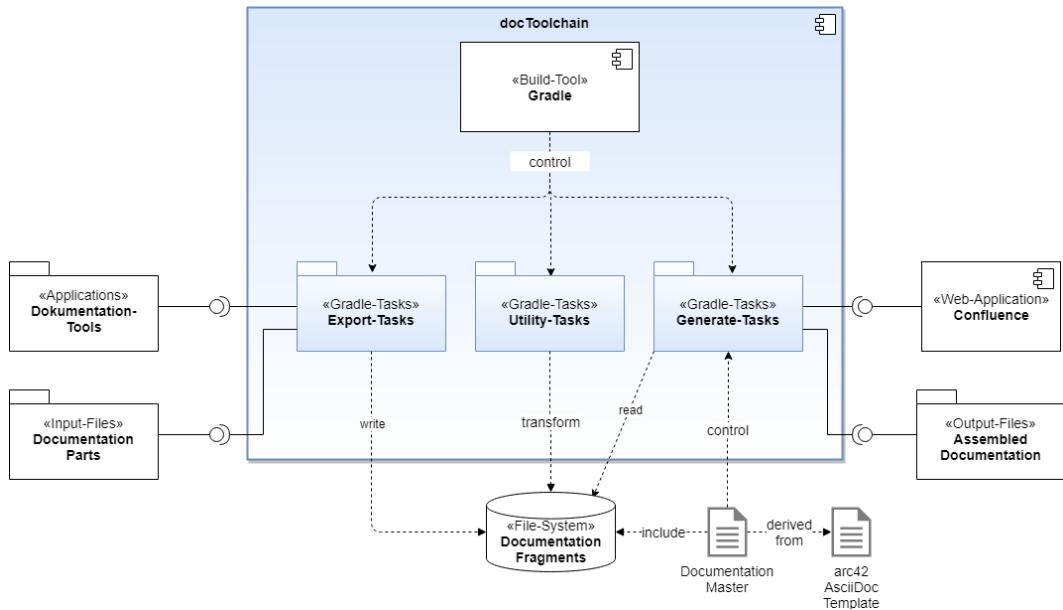
So, we used two other solution approaches:

- conversion from another already available format (HTML -> Confluence)
- conversion via Pandoc (Docx, ePub)

VI.5 Building Block View

5.1 Level 1

The following picture shows level 1 building blocks.



Building Block View - Level 1: Overview

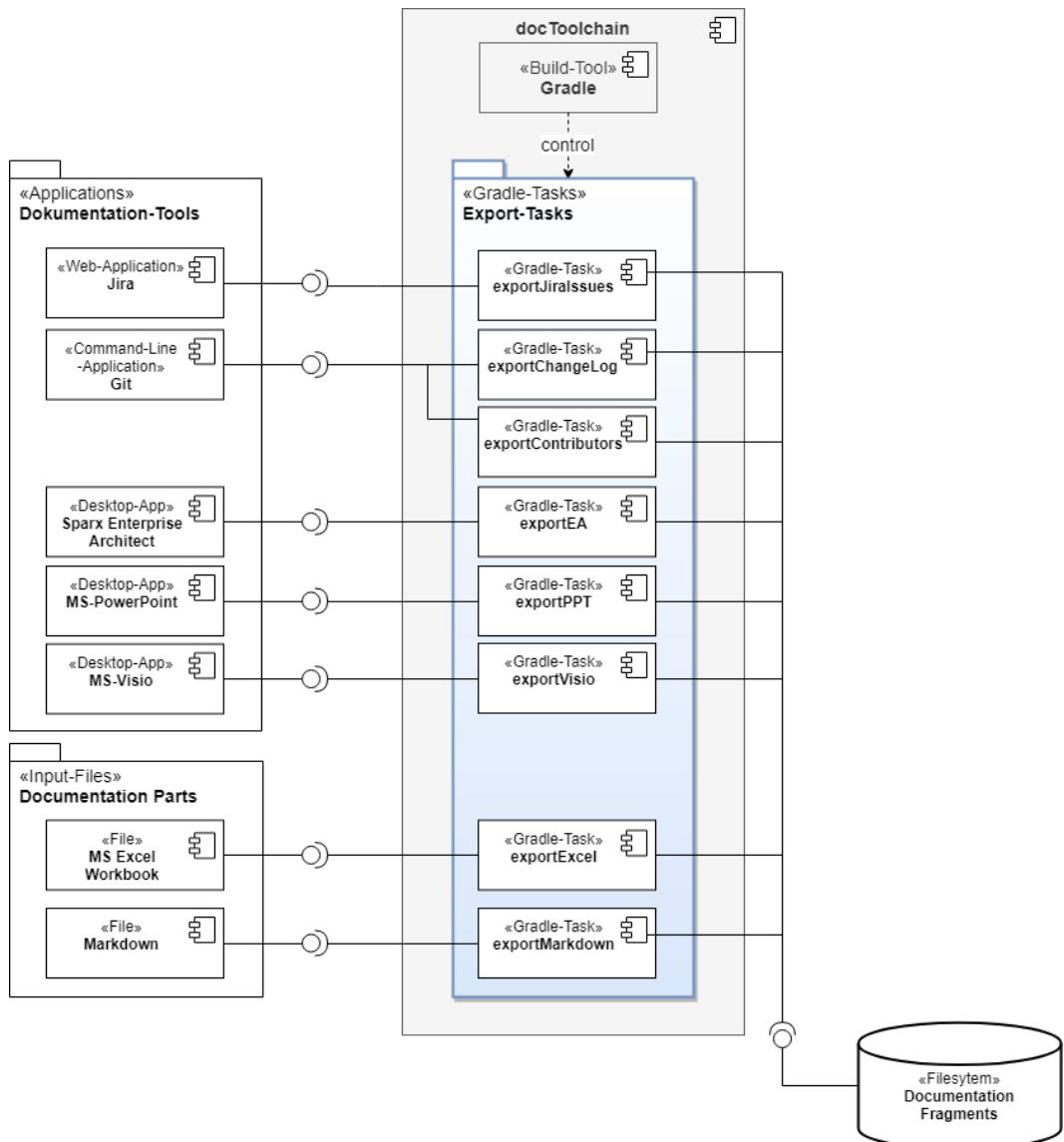
Component	Description
Gradle	This is the Gradle build system which is used for docToolchain to orchestrate the documentation tasks. Details can be found at https://gradle.org .
Documentation Tools	These are various tools which are used to create parts of your system documentation. Popular examples are MS Word and MS PowerPoint. docToolchain interfaces these tools directly instead reading the source files because there is no known direct file API for these tools. See <i>level 2: Documentation Tools</i> for details.

Component	Description
Documentation Source Files	Some parts of your documentation might be created in a format for which a direct file API exists. This package contains those input files.
Export Tasks	This package contains several tasks which export data from <i>Documentation Tools</i> by either interfacing the <i>Documentation Tools</i> or accessing the <i>Documentation Source Files</i> directly . See <i>level 2: Export Tasks</i> for details.
Documentation Fragments	All parts of your documentation which where exported from another source are stored in the file system for further processing.
Utility Tasks	This is a collection of Gradle tasks which do not belong to the <i>export-</i> or <i>generate-tasks</i> . See <i>level 2: Utility Tasks</i> for details.
Generate Tasks	All generate tasks have in common that they use the <i>documentation master</i> to generate the assembled documentation in a specific output format. This package contains all of these tasks. See <i>level 2: Generate Tasks</i> for details.
Confluence	Confluence is the well-known enterprise wiki created by Atlassian. Details can be found at https://www.atlassian.com/confluence .
Assembled Documentation	The goal of docToolchain is to assemble system architecture documentation. This package is the target for the various output files.
Documentation Master	This is the heart of your documentation. It uses AsciiDoc as markup language to implement your documentation. The <code>include</code> -directive of AsciiDoc is used to reference all <i>Documentation Fragments</i> at the correct location of your documentation. Through this mechanism, it controls how the documentation is assembled from the various input sources. There can be more than one documentation master, for instance one for each type of documentation and each stakeholder.

Component	Description
arc42 AsciiDoc Template	Since docToolchain is about system architecture documentation, the best way to get started is to derive the <i>Documentation Master</i> from the <i>arc42 AsciiDoc Template</i> . The template can be found at https://arc42.org/download .

5.2 Level 2

5.2.1 Export Tasks



Building Block View - Level 2: Export Tasks

Documentation Tools & Export Tasks

These are the tools from which docToolchain is able to export documentation fragments. Each tool is interfaced by at least one export task through a specific interface.

Component	Description
Jira & exportJiraIssues	Jira is the issue tracking system from Atlassian. The <i>exportJiraIssues</i> task exports a list of issues for a given JQL-Search. These issues are exported through the Jira Server REST API¹¹⁵ . The result is stored as an AsciiDoc table to be included from the <i>Documentation Master</i> . More documentation for this task can be found in the manual¹¹⁶ .
Git, exportChangeLog & exportContributors	Git is directly invoked as command-line tool through Groovy. The <i>exportChangeLog</i> task exports the list of change messages from git for the documentation folder as an AsciiDoc table to be included from the <i>Documentation Master</i> . More documentation for this task can be found in the manual¹¹⁷ . The <i>exportContributors</i> task exports the list of contributors and additional file attributes from git for all AsciiDoc files. This information is stored as AsciiDoc files to be included from the <i>Documentation Master</i> . More documentation for this task can be found in the manual¹¹⁸ .

¹¹⁵<https://docs.atlassian.com/software/jira/docs/api/REST/>

¹¹⁶https://doctoolchain.github.io/docToolchain/#_exportjiraissues

¹¹⁷https://doctoolchain.github.io/docToolchain/#_exportchangelog

¹¹⁸https://doctoolchain.github.io/docToolchain/#_exportcontributors

Component	Description
Sparx EA & exportEA	<p>Sparx Enterprise Architect (EA) is the UML modeling tool from Sparx Systems. The <i>exportEA</i> task exports all diagrams and element notes from a model through the COM API provided by EA. The API is interfaced through a Visual Basic Script started from the task. The COM API is more an application remoting interface than a data access interface. As a result, docToolchain starts EA headless and iterates through the model in order to export the data. The resulting images and text files can be included from the <i>Documentation Master</i>. More documentation for this task can be found in the manual¹¹⁹.</p>
MS-PowerPoint & exportPPT	<p>MS-PowerPoint is the presentation software from Microsoft. The <i>exportPPT</i> task exports all Slides of a slide-deck as .png image files together with the speaker notes as text files. As Interface, the COM API provided by PowerPoint is used through a Visual Basic Script which is started from the task. MS-PowerPoint is used in this case because it has a quite usable API. But the slide decks can also be created and modified with other presentation tools. The resulting images and text files can be included from the <i>Documentation Master</i>. More documentation for this task can be found in the manual¹²⁰.</p>

¹¹⁹https://doctoolchain.github.io/docToolchain/#_exportea

¹²⁰https://doctoolchain.github.io/docToolchain/#_exportppt

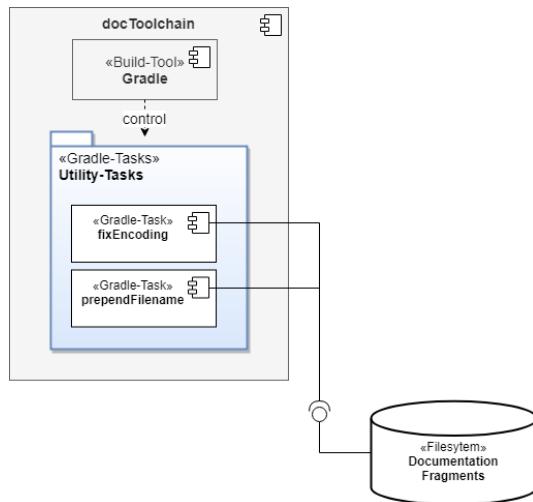
Component	Description
MS-Visio & exportVisio	MS-Visio is the diagramming software from Microsoft. The <i>exportVisio</i> task exports all diagrams as image files together with their corresponding diagram notes as text files. As Interface, the COM API provided by Visio is used through a Visual Basic Script which is started from the task. The resulting images and text files can be included from the <i>Documentation Master</i> . More documentation for this task can be found in the manual¹²¹ .
Documentation Parts & Export Tasks	These are files which can be directly interfaced by docToolchain without the need of an additional tool. Each file is interfaced by one export task through a certain library.
MS-Excel Workbook & exportExcel	MS-Excel is the spreadsheet software from Microsoft. The <i>exportExcel</i> task exports all spreadsheets contained in a .xlsx-workbook as CSV-files and AsciiDoc-Tables to be included from the <i>Documentation Master</i> . As Interface, the Apache POI library is used. More documentation for this task can be found in the manual¹²² .

¹²¹https://doctoolchain.github.io/docToolchain/#_exportvisio

¹²²https://doctoolchain.github.io/docToolchain/#_exportexcel

Component	Description
MarkDown & exportMarkdown	MarkDown is the well-known, and quite popular markup language. The <i>exportMarkdown</i> task exports all .md files found in the documentation folder to AsciiDoc to be included from the <i>Documentation Master</i> . The interface is a direct file access and as converter the nl.jworks.markdown_to_asciidoc library is used. More documentation for this task can be found in the manual ¹²³ .

Utility Tasks



Building Block View - Level 2: Utility Tasks

These are tasks which might be helpful in certain situations but do not directly belong to the scope of docToolchain

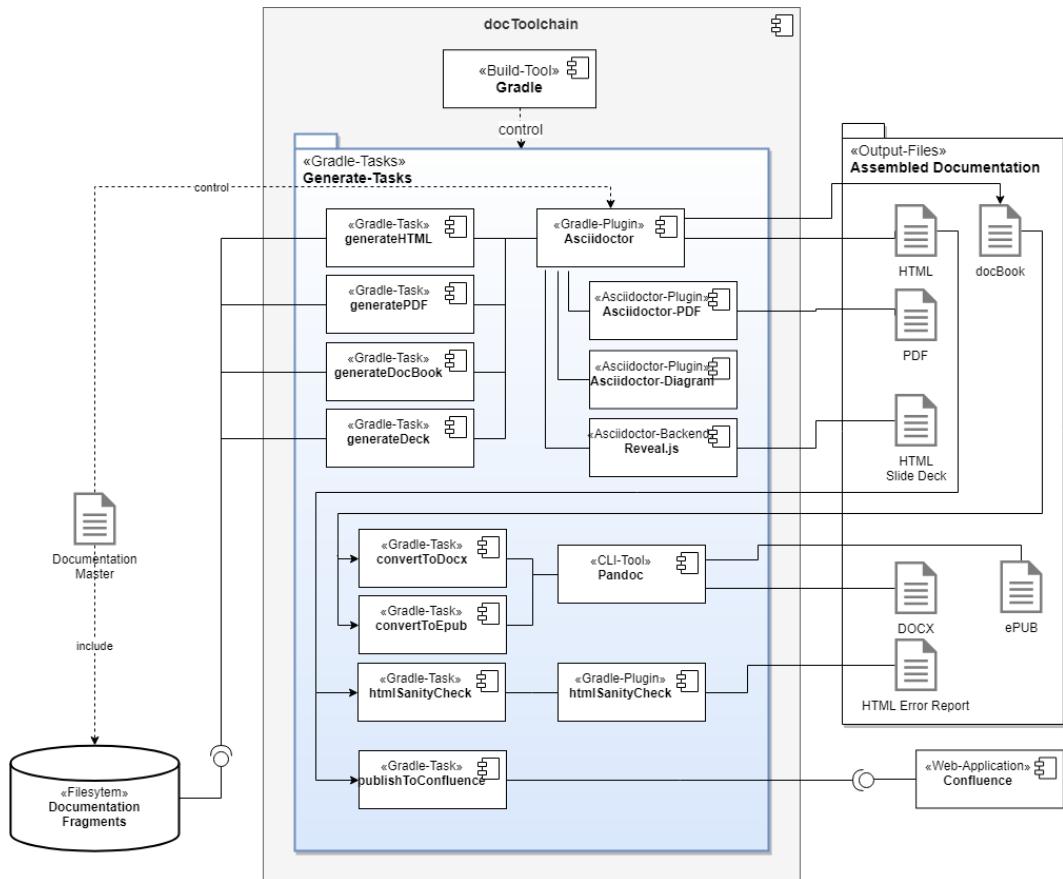
¹²³https://doctoolchain.github.io/docToolchain/#_exportmarkdown

Component	Description
fixEncoding	Asciidoctor is very strict regarding the encoding. It needs UTF-8 as file encoding. But sometimes a text editor or some operation on your documentation file might change the encoding. In this case, this task will help. It reads all documentation files with the given or guessed encoding and rewrites them in UTF-8. More documentation for this task can be found in the manual¹²⁴ .
prependFilename	This task is a workaround for the behavior of the Asciidoctor <code>include</code> -directive: in Asciidoctor, there is no way to determine the name of the currently rendered file, only of the master document. But for features like the list of contributors or a link back to the git sources, the name of the currently rendered file is needed. This task crawls through all AsciiDoc files and prepends their filename to solve this problem. More documentation for this task can be found in the manual¹²⁵ .

¹²⁴https://doctoolchain.github.io/docToolchain/#_fixencoding

¹²⁵https://doctoolchain.github.io/docToolchain/#_prependfilename

5.2.4 Generate Tasks



Building Block View - Level 2: Generate Tasks

This package contains all tasks which assemble your documentation to various output files in different formats.

Component	Description
Asciidoctor	<p>Asciidoctor as rendering engine is at the heart of this package. docToolchain uses Asciidoctor as Gradle plugin which wraps the Ruby implementation with jRuby. This is mostly transparent for the user, but it is important in the enterprise context: all Asciidoctor plugins also refer to their wrapped versions and not the Ruby gems. This way, we avoid to reference the Ruby gem repository as an external dependency for docToolchain. Instead, all wrapped plugins are referenced from the standard java repositories. More details can be found at https://asciidoctor.org/docs/asciidoctor-gradle-plugin/.</p>
Asciidoctor-PDF	<p>Asciidoctor needs this plugin to render your document as PDF. More details can be found at https://asciidoctor.org/docs/asciidoctor-pdf/.</p>
Asciidoctor-Diagram	<p>This plugin enables several text-based diagram formats like plantUML. They can be specified inline as textual representation. See https://asciidoctor.org/docs/asciidoctor-diagram/ and http://plantuml.com/ for more details.</p>
Reveal.js	<p>Asciidoctor parses the input file and renders each parsed element through defined markup fragments. A collection of those fragments is called a rendering backend. The Reveal.js backend is capable of rendering AsciiDoc as HTML which is turned into a slide presentation through some clever styles and JavaScript. See https://asciidoctor.org/docs/asciidoctor-revealjs/ for more details.</p>

Component	Description
generateHTML	This is the most straightforward task which generates the standard HTML output with the help of the Asciidoctor Gradle Plugin. Takes as input the <i>Master Documentation</i> which includes the <i>Documentation Fragments</i> . More documentation for this task can be found in the manual ¹²⁶ .
generatePDF	This task is equal to the <i>generateHTML</i> task, but it uses the <i>Asciidoctor-PDF</i> Asciidoctor Plugin to generate a PDF file as output. Takes as input the <i>Master Documentation</i> which includes the <i>Documentation Fragments</i> . More documentation for this task can be found in the manual ¹²⁷ .
generateDocBook	This task is equal to the <i>generateHTML</i> task but produces a docBook XML-file as output. It uses the Asciidoctor Gradle Plugin for this task. Takes as input the <i>Master Documentation</i> which includes the <i>Documentation Fragments</i> . More documentation for this task can be found in the manual ¹²⁸ .
generateDeck	This task uses the <i>Reveal.js</i> Asciidoctor backend to generate an HTML based slide deck. Takes as input the <i>Master Documentation</i> which includes the <i>Documentation Fragments</i> . More documentation for this task can be found in the manual ¹²⁹ .
convertToDocx & convertToEpub	These two tasks use <i>Pandoc</i> as command-line tool to convert the generated HTML documentation into a DOCx or ePUB file. More documentation for this task can be found in the manual ¹³⁰ .

¹²⁶https://doctoolchain.github.io/docToolchain/#_generatehtml

¹²⁷https://doctoolchain.github.io/docToolchain/#_generatepdf

¹²⁸https://doctoolchain.github.io/docToolchain/#_generatedocbook

¹²⁹https://doctoolchain.github.io/docToolchain/#_generatedeck

¹³⁰https://doctoolchain.github.io/docToolchain/#_generatedocx

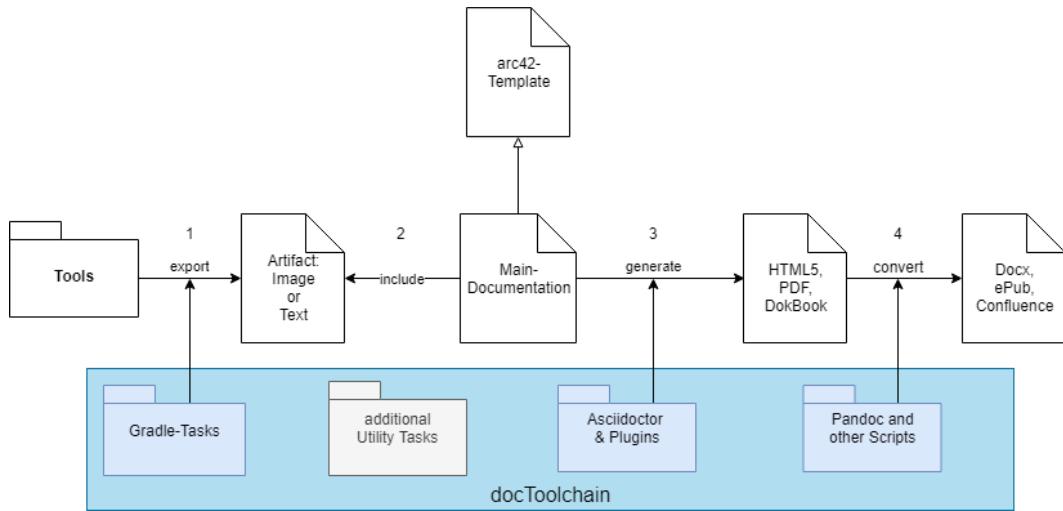
Component	Description
htmlSanityCheck	This task runs a check on the generated HTML documentation. It uses the Gradle Plugin of the same name to perform this task. It generates a Unit-Test like HTML Error Report. See https://github.com/aim42/htmlSanityCheck for more details.
publishToConfluence	This task takes the generated HTML output and splits it by the headline levels into separate pages. These pages and their images are then published to a <i>Confluence</i> instance. It uses the confluence REST API ¹³¹ to interface <i>Confluence</i> . More documentation for this task can be found in the manual ¹³² .
Confluence	Confluence is the well-known enterprise wiki created by Atlassian. Details can be found at https://www.atlassian.com/confluence .
Pandoc	Pandoc is “the swiss army knife for markup conversion”. It is used to generate DOCx and ePub files. See https://pandoc.org/ for more details.

¹³¹<https://developer.atlassian.com/server/confluence/confluence-server-rest-api/>

¹³²https://doctoolchain.github.io/docToolchain/#_publishtoconfluence

VI.6 Runtime View

There are no special runtime situations which need to be explained as sequence diagram. However, the following diagrams show the flow of information through the toolchain.



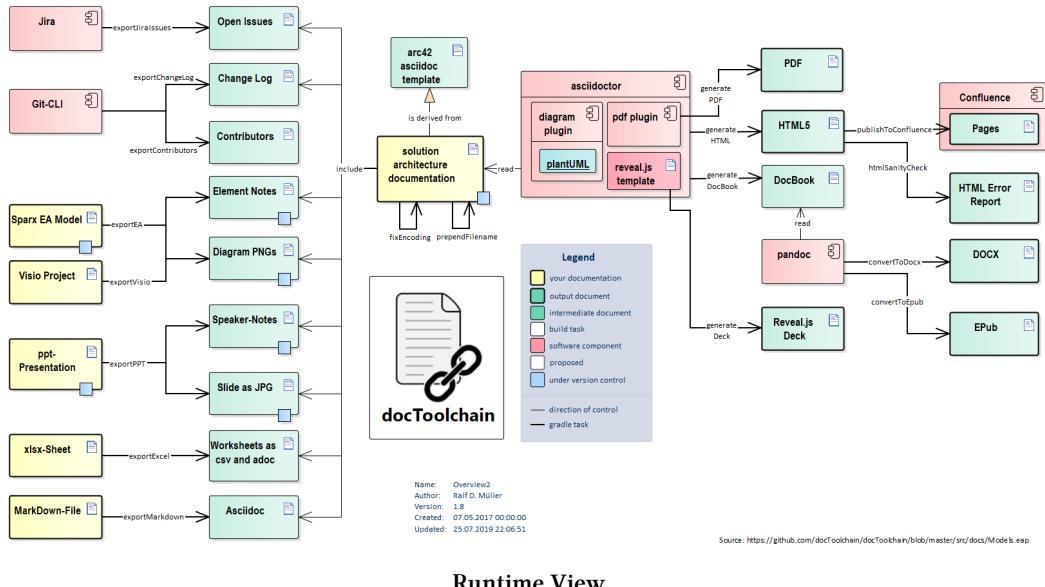
Runtime View: Abstract Flow

1. docToolchain exports data (diagram images and plain text) from documentation tools.
2. the main documentation (on the basis of the arc42-template) includes the exported artifacts
3. the final output is generated via Asciidoc and plugins...
4. ...and converted through additional scripts and tools

Besides, docToolchain contains some additional utility tasks which do not directly fit into the schema above, like the `htmlSanityCheck`-task.

The next diagram shows the details of all currently implemented tasks. It focuses on the various input artifacts which are transformed through the Gradle tasks which make up docToolchain. Each connection is either a Gradle task which transforms the artifact or an AsciiDoc include-directive which references the input artifacts

from the main documentation and thus instructs Asciidoc to assemble the whole documentation.



Runtime View

VI.7 Deployment View



docToolchain is only used to generate documentation and thus is not deployed to a production environment. However, depending on your requirements, you might want to deploy it not only to your dev machine but also to a build server.

The general deployment is done by cloning the docToolchain source repository. Depending on your needs, the details vary.

7.1 Option 1: Installed Command-Line Tool

For this option, you clone the docToolchain repository to a location outside of your project for which you want to write documentation. You then add the <docToolchain home>/bin-folder to your PATH variable. Now you can use docToolchain as command-line tool:

```
1 doctoolchain <docDir> <task>
```

This approach works quite well and independent of your project. However, it violates RQ6 - [clone, build, test, run](#) for your project. Someone who clones your project will also have to install docToolchain before he can work on the documentation. Also, breaking changes in docToolchain are also likely to break your build if you do not update docToolchain on all systems.

7.2 Option 2: Git Submodule

This approach is the currently preferred option. You add docToolchain to your project by adding it as a Git submodule.

```
1 git submodule add https://github.com/docToolchain/docToolchain.git
```

This line adds one additional folder to your project, containing the whole docToolchain repository:

```
1 yourProject/
2   └── docToolchain/      <-- docToolchain as submodule
3   └── src/
4     └── docs            <-- docs for yourProject
5     └── test             <-- tests for yourProject
6     └── java             <-- sources of yourProject
7   └── ...
8   └── build.gradle       <-- build of yourProject
9   └── ...
```

As you can see, this gives you a clean separation of your project and docToolchain. The git submodule is just a pointer to the docToolchain repository - not a copy. It points to a specific release. This approach ensures that whoever clones your project, also gets the correct version of docToolchain.

To use docToolchain with this approach, you configure it as a sub-project of your main Gradle project (see details in [this tutorial¹³³](#)).

If your project uses another build system like Maven, you can still use this approach. You simply use Maven to build your code and Gradle to build your documentation.

See also [TR3: Git Submodules](#)

7.3 Option 3: Docker Container

Last but not least, you can deploy docToolchain as command-line tool to a docker container. This then can be used in various ways (for instance in a Gitlab pipeline).

¹³³<https://docs-as-co.de/getstarted/tutorial2>

VI.8 Cross-cutting Concepts

Because of the nature of docToolchain (locally used development tool), most of the standard cross-cutting concepts (e.g., security, monitoring, etc.) do not apply.

This is why this section is short.

Automated Testing

In an ideal world, we should have covered all code with unit tests. However, currently, the smallest unit of docToolchain is a Gradle task. Because of the dependencies to Gradle, we can't easily test these tasks in isolation.

The current solution for automated tests is to use the [gradleTestKit¹³⁴](#) and [Spock¹³⁵](#).

Through the gradleTestKit, Gradle will start another Gradle instance with its own test configurations. The gradleTestKit ensures that all tasks can be integration-tested in an automated way.

Spock helps to define the tests in an easy to maintain, [behavior driven development¹³⁶](#) (BDD) way.

¹³⁴<https://docs.gradle.org/4.9/userguide/userguide.html>

¹³⁵<http://spockframework.org/>

¹³⁶https://en.wikipedia.org/wiki/Behavior-driven_development

VI.9 Architecture Decisions

This chapter lists the important design decisions which have been made while docToolchain evolved. They fulfill two tasks:

- explain users and contributors who are new to the project, why things are how they are
- help to revise decisions whose basis has changed

DD1: wrapped Plugins instead of Ruby Gems

Problem: Asciidoctor plugins can be referenced directly as ruby gems or as java dependencies to the JRuby wrapped jar files. While the ruby gems always provide the latest version, it might take some days for the JRuby versions to be available. Both types of plugins are downloaded from different repositories. In an enterprise environment, the ruby gem repository might not be accessible. Since the user of docToolchain is likely to be a JVM developer, and Gradle already depends on Java repositories, the ruby gem repository is an additional dependency and would violate [C3 - enterprise ready](#)

Decision: docToolchain will always use the JRuby version because the needed jar-files can be downloaded from the same repository as all the other java dependencies.

Decided by Ralf D. Müller, [April 2017](#)¹³⁷

DD2: Visual Basic Script for EA Automation

Problem: The UML-tool “Enterprise Architect” by Sparx Systems can be interfaced in several ways:

via internal scripts

This requires a change in the setup of the projects [EA](#) project.

¹³⁷<https://rdmueller.github.io/enterprise-edition2/>

via the Sparx Java API or via a COM-bridge like JaCoB

Both solutions require a DLL to be installed. This violates [C3 - enterprise ready](#).

In addition, experience showed that the Java API (at the time it was tested), wasn't free of flaws.

via visual basic

This requires some Visual Basic know-how instead of already available Java experience.

Decision: The first approach isn't easy to use, and the second even violates a constraint. So it was decided to use visual basic as an automation tool. Since this is the native interface for a COM-Object (which Enterprise Architect provides), the amount of bugs in the interface is expected to be minimal. For a first [POC](#), I used Visual Basic together with Visual Studio. This setup provides intelli-sense and code completion features and thus makes it easy to explore the API. However, the final task uses Visual Basic Script, which is a little bit different and doesn't provide you code completion features.

Decided by Ralf D. Müller, ca. August 2016

DD3: Support of Maven and Gradle

Problem: I initially planned to support the two major build systems known to me: Maven and Gradle. While Asciidoc supports both build systems through plugins, I soon noticed that even basic features like PlantUML and PDF generation are hard to maintain for both systems. In addition, some features are easy (for me) to implement through Gradle and harder to implement in Maven.

Decision: Support for Maven has been dropped.

Decided by Ralf D. Müller, August 2016

DD4: Binary Gradle Plugins

Problem: docToolchain consists of Gradle script plugins. Some developers consider these not as a clean approach, and they make it harder to use docToolchain (see [deployment view](#)).

However, script plugins also have advantages. They make the code more compact and easier to maintain because they consist of single files and are not distributed over several classes. The tasks are often so small that we included their source in the manual of docToolchain. When a user reads through the manual, he sees at once how the tasks are implemented. The chances are good that such a user turns into a contributor with this knowledge.

Decision: the pros of using script-plugins weigh currently more than the cons. docToolchain has to be easy to maintain and extend (see [RQ2 - easy to modify and extend](#)).

There is one exception to this rule: Some tasks - like the Pandoc tasks - are just wrappers to other executables. A binary plugin could even help in this situation by providing an automated download of the executable.

(in fact, a first version of a [binary Pandoc plugin is already available¹³⁸](#))

Decided by Ralf D. Müller, mid. 2018

DD5: AsciiDoc as basis

Problem: in order to follow the Docs-as-Code approach, a format for writing documentation is needed, which is solely based on plain text. Binary formats - mainly used by word processors - are not very well suited for the tools used by a developer and thus not suited for the Docs-as-Code approach.

There are several plain text markup formats available like MarkDown, Restructured-Text, Latex, and AsciiDoc.

Decision: docToolchain uses AsciiDoc as markup format to write documentation. AsciiDoc is easy to use but also powerful and has already proven that it is perfect for writing technical documentation.

DD6: Deployment Options

Problem: docToolchain is no application which can be deployed to a server. So there is no standard pattern for the deployment of docToolchain.

¹³⁸<https://github.com/docToolchain/pandocGradlePlugin>

Decision: the most common options to use (deploy) docToolchain are the three described in chapter 7 - Deployment View

Considered alternatives: the following deployment options have been considered but not implemented.

DD6.1 Embedded

This was the first approach used, and we do not recommend it anymore. In this approach, you use docToolchain itself as a base project, and add all code and documentation to that base project.

This approach makes it hard to update docToolchain itself since docToolchain, and your project are mangled together:

```
1  yourProject/
2  └─ scripts/           <-- docToolchain task definitions
3  |  └─ AsciiDocBasics.gradle
4  |  |
5  |  ...
6  |  └─ publishToConfluence.gradle
7  └─ docs/             <-- docToolchain rendered manual
8  └─ resources/        <-- docToolchain external submodules
9  └─ src/
10 |  └─ docs            <-- docs for docToolchain AND yourProject
11 |  └─ test             <-- tests for docToolchain AND yourProject
12 |  └─ java             <-- sources of yourProject
13 └─ ...
14 └─ build.gradle       <-- merged build file of docToolchain
                           AND yourProject
```

As you can see, this was only a good idea in the beginning, when docToolchain mainly consisted of an extended Asciidoctor build templates.

DD6.2 Gradle Script-Plugins

This approach is currently a theoretical option. It has never been used yet, but is mentioned here for the sake of completeness.

Gradle has a feature called Script-Plugins. The same way as docToolchain references the modular, local Gradle-files for different tasks,

```
1 apply from: 'scripts/exportExcel.gradle'
```

you can reference these scripts as remote files in your project's build.gradle-file:

```
1 apply from: 'https://github.com/docToolchain/docToolchain/blob/b415bfb/\\
2 scripts/exportExcel.gradle'
```

This line references an exact version of the file - so as long as you trust GitHub and your connection to GitHub, this approach is safe and works for all self-contained Tasks.

DD6.3 Gradle Binary-Plugins

See [DD4: Binary Gradle Plugins](#) for why this was not an option in the past.

For this option, the existing script plugins have to be converted binary plugins for Gradle.

As a result, they can be referenced directly from your build.gradle file. Html Sanity Check is an excellent example of this approach.

```
1 plugins {
2     id "org.aim42.htmlSanityCheck" version "1.0.0-RC-2"
3 }
```

We could turn docToolchain into a binary plugin which references all other plugins as its dependency. Alternatively, we could turn docToolchain into a collection of binary plugins, and a User of docToolchain would reference those plugins he needs.

VI.10 Quality Requirements

QS1: Confidentiality

In order to keep control over the content of your documentation, docToolchain shall never use a web-based, remote service for any task without the explicit knowledge of the user.

QS2: Security (besides Confidentiality)

When you think of documentation, security is not the first thing which comes to your mind, is it?

But when you automate some parts of your documentation, or even just include remote sources, security comes into play.

It can even be that you use a plugin (like the [asciidocorj-screenshot plugin¹³⁹](#)) which lets you execute arbitrary code in the generation phase.

In docToolchain, we assume that the documentation is under your control and hence *do not* apply any special security measures besides respecting [QS1: Confidentiality](#).

QS3: Repeatability

Every run of a docToolchain task creates the same predictable output.

This rule ensures that it doesn't matter if you move the source or the generated output from stage to stage.

QS4: Transformation Stability

No task of docToolchain will remove content. Tasks (like the export-tasks) may only add content. This rule is to ensure that all generated content contains the same information.

However, some formats might not be able to display the same layout and text style.

¹³⁹<https://github.com/asciidocorj/asciidocorj-screenshot>

QS5: Performance

Functionality first. docToolchain is about the automation of repetitive, manual tasks. So even when an automated task is slow, it will have a value.

However, we should strive for better performance of the tasks whenever we see a chance to optimize.

QS6: Ease of Use

docToolchain should be as easy to use as possible. The goal is that docToolchain is easy to install and that the use of all tasks does not need any knowledge about the inner working of the system.

Currently, a single binary Gradle plugin is seen as the right solution to achieve this quality goal, but it is not yet on the roadmap. (see also [DD4: Binary Gradle Plugins](#))

QS7: Maintainability

Many users with different needs use docToolchain. Those different needs are the result of different environments and different tools in use.

Therefore docToolchain tasks should be easy to modify to fit everyone's needs. At least those tasks which are not stable yet.

There should also be blueprints for new tasks.

The "easy to modify" quality goal is currently achieved through the scripting character of the Gradle script-plugins. The source is referenced from the documentation and easy to understand and thus easy to modify.

However, this is in contrast to the "Easy of Use" quality goal: script plugin vs. binary plugin.

VI.11 Risks and Technical Debt

This chapter describes the risks which might occur when you use docToolchain and how they are - from an architectural point of view - mitigated.

TR1: Outdated Technology

Description: A software system might live for several decades. During this time, some parts of the toolchain might get outdated. For instance, a UML modeler for instance for which you didn't buy a new license.

In such a case, you might not be able to re-generate parts of your documentation.

Mitigation: The highest risk of this kind is attached to the proprietary tools from which diagrams and texts are exported. docToolchain exports these artifacts not to the `/build` folder but to the `/src/docs` folder by purpose:

- if one of the tools gets outdated, you still have the exported data and hence can still work with it.
- in addition, the exported `.png`-images and plain text files are easier to compare than the often binary source formats. This enables a better review of the changes in your documentation.

TR2: Missing Acceptance of Docs-as-Code Approach

Description: When you start to use docToolchain and thus implement the Docs-as-Code approach, you risk that colleagues do not accept this “new” approach.

In our experience, this is a low risk. We've never seen people switch back to MS Word from AsciiDoc. However, it might give users a good feeling to be able to.

Mitigation: You can convert your docs to several output formats like PDF, MS Word and Confluence. This gives you not only a “fallback” when you reach a point where you think that the Docs-as-Code approach doesn't fit your requirements, it also allows you to work together with colleagues who haven't transitioned to Docs-as-Code yet.

TR3: Git Submodules

Description: Git Submodules are great in theory. As soon as you use them, you will notice that it needs a bit of practice.

- if added via ssh protocol, users without ssh configured for their git account can't clone them
- submodules easily get in a “detached head” state
- at least git on windows has authentication problems with submodules (use of pageant helps)

This could lower the acceptance of docToolchain in general.

Mitigation:

- always use https-protocol to add a submodule
- Someone should write a “Guide to Git Submodules” for docToolchain.

TR4: automated Tests (technical debt)

Description: the project does not have good test coverage. The Travis checks often break.

Mitigation: No appropriate mitigation yet, besides working on the test coverage.

VI.12 Glossary

Term	Definition
COM, Component Object Model	COM is an interface technology defined and implemented as a standard by Microsoft. See also COM on Wikipedia¹⁴⁰
COM-Bridge	A library which allows you to use one technology from within another. In this case, it allows you to access the COM-interface from within Java programs.
DLL, Dynamic-Link Library	Dynamic-link library (or DLL) is Microsoft's implementation of the shared library concept in the Microsoft Windows and OS/2 operating systems. See also DLL on Wikipedia¹⁴¹
EA, Enterprise Architect	A UML modeling tool by Sparx Systems. See EA on Wikipedia¹⁴²
JACOB	A Java-COM-Bridge. See https://sourceforge.net/projects/jacob-project/ for details
POC, Proof of Concept	Minimalistic code that proofs a certain idea
Sparx, Sparx Systems	The company behind Enterprise Architect
Task	Tasks are the smallest entity of the toolchain. A docToolchain task is implemented as Gradle task.

¹⁴⁰https://en.wikipedia.org/wiki/Component_Object_Model

¹⁴¹https://en.wikipedia.org/wiki/Dynamic-link_library

¹⁴²[https://en.wikipedia.org/wiki/Enterprise_Architect_\(software\)](https://en.wikipedia.org/wiki/Enterprise_Architect_(software))

VII - Foto Max

By [Hendrik Lösch](#).

FotoMaX is a purely theoretical product for ordering photos at a kiosk system, such as those found in drugstores or supermarkets all over Europe. The photos can additionally be ordered via a website that is also used by partnering businesses to provide additional services. This website is only mentioned where necessary but is not described in detail because otherwise the example would become too complex.

The documentation was created based on various real projects in context of cyber-physical systems but has no real implementation. It was created for and used during many workshops about architecture documentation. The goal is to show how to document systems that have been grown over many years, were maintained by multiple teams, and have a certain degree of interaction with special hardware components.

This is also the reason why the structure of this document differs slightly from the one described in arc42. Especially the first chapter might seem redundant in context of this book because it describes the overall structure. The idea behind this chapter is to give the readers all the information they need to understand which parts of the documentation are important for them. The last chapter is also one that is not part of arc42 because it contains typical information teams use to organize their daily work. It was added to show how to incorporate such things in your overall documentation and link between architecture documentation and team organization. Moreover, it contains actual links between different pages because the whole document was written with the use of a wiki system in mind.

VII.1 About this document

It is safe to assume that new readers of the document will start with the first page of the document. Thus we can use this page to provide the most important information and guide the reader through the document. This approach is very important especially when using wiki systems otherwise they can become very confusing.

This documentation describes all aspects that should and must be considered in the development and maintenance of the FotoMaX device software. It uses the arc42 template as a basic structure. According to this, the documentation is divided into the following subsections, which can be read from top to bottom, leading from an abstract and external problem view to a concrete internal solution view.

Introduction & Goals

This chapter contains basic information necessary to understand the product and the business context in which it was created.

Quality Requirements

This chapter deviates from the official Arc42 structure in order to emphasize the importance of the quality requirements. Normally, these would only appear much later in the document, which makes sense if they are described in great detail. In this case, here are only a comparatively small number of scenarios. All of these are very important and therefore brought into focus by appearing very early in the document.

This chapter describes the non-functional requirements of the software system and prioritizes its quality attributes. These are fundamental decisions that influence many different aspects of the software and software architecture.

Constraints

This chapter describes the constraints and requirements that limit software design, implementation or the development process.

Context & Scope

In this chapter the system is delimited from its environment. In this way, user roles, external systems and the associated interactions as well as needed interfaces are identified.

Solution Strategy

The actual implementation strategy is described in this chapter.

Building Block View

The structural design and its decomposition into building blocks of various granularity is described in this chapter.

Runtime View

All dynamic relationships that are to be highlighted are described in the runtime view chapter.

Deployment View

The deployment view clarifies in which runtime environments the individual components of the system are installed and executed.

Crosscutting Concepts

Fundamental decisions should not be made based on gut feeling, but as empirically as possible. In this chapter, all the information that is decisive for design decisions is collected.

Architectural Decisions

Fundamental decisions should not be made on the basis of a gut feeling, but as empirically as possible. In this chapter, all the information that is decisive for design decisions is collected.

Risks & Technical Debt

We see risks as all the things that can have a negative impact on the project and the software system if they are not handled appropriately. This chapter describes all risks and technical liabilities together with their possible effects and countermeasures, to make sure that they are not forgotten.

Glossary

The glossary summarizes all terms that may hinder the understanding of the documentation.

Organizational Topics

This chapter is not part of ARC42 but was added to this documentation to show how teams can organize their daily work next to the architecture documentation so that both information sources are kept close together. This allows to link between both documentations but also ensures that both are also always present. This in turn ensures that the architecture documentation is not perceived as something that exists somewhere somehow, but is an important part of one's own work.

This chapter contains information that does not relate directly to the software architecture, but to the day-to-day collaboration in implementing the software. This includes, the Definition of Ready and Definition of Done, but also guidelines, checklists and things like vacation planning.

VII.2 Introduction and Goals

This chapter describes the key requirements and driving forces that must be considered when implementing the software architecture and developing the system.

About FotoMaX

FotoMaX is a white label solution for ordering photo products. It enables retailers of various types to offer their own customers additional services, related to the processing and ordering of photographic content. This solution consists of two parts, the device hardware and software, which in their combination are set up at various partners as “Sales Units” or just “Units”. This document primarily describes the software and only discusses the hardware when it influences the software design significantly.

FotoMax does not take care of the actual printing of the orders. It provides the link between the companies that set up a sales unit in their stores and those that do the actual printing of the photo products. Based on the chosen version of FotoMax these could even be the same companies. The products are paid directly in the respective store or via mobile payment solutions.

In the simplest case, printing takes place directly in the store (“FotoMaX Standalone”) or can be forwarded to a print shop (“FotoMaX Connect”) to allow a larger variety of products like cloths, coffee cups or posters.

Typical points of sale:

- Supermarkets
- Drugstores
- Hotels
- ...

Stakeholders

The following people represent a particular view of the system and should therefore be consulted in fundamental decisions.

Please note that these persons and their contact details are purely fictional. Similarities to existing companies or persons are therefore purely coincidental.

Name	Contact	Role
Hannelore Meier	Hannelore.Meier@fotomax.net	Partner Manager Europe
Marcus Heinemann	Marcus.Heinemann@fotomax.net	End-User Representative
Mareike Edelman	Mareike.Edelman@fotomax.net	Operation Manager Europe
Peter Laudner	Peter.Laudner@fotmax.net	Enterprise Architect

VII.3 Quality Requirements

This chapter prioritizes the quality attributes of the software based on ISO 25010¹⁴³ and explains the associated decisions.

Prioritization

The prioritization of the quality attributes of the system was carried out with the involvement of the stakeholders (→VII.2.2). It was initially executed for the sales unit then modified later to incorporate the portal:

Priority	Attribute	Explanation
1	Functionality	The functional correctness and completeness of the software has top priority; as no logical errors may occur during operation and all workflows or processes must be implemented in a way that leads to a successful order.
2	Usability	The software must be easy to use by people with different levels of technical knowledge and be perceived as appealing. It must be possible to place an order as quickly and conveniently as possible to attract walk-in customers.
2	Security	The sales units are located in a public space and offers various possibilities for data transmission.
3	Maintainability	The software system is comparatively extensive and must be able to be redesigned for different partners, so it should be modular and easily testable for different configurations.
3	Availability	If the service is not available, it does not generate any revenue. Therefore, it must always be possible to place orders locally on sales units. The actual transfer of orders to the central services can take place with a certain delay.

¹⁴³<https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>

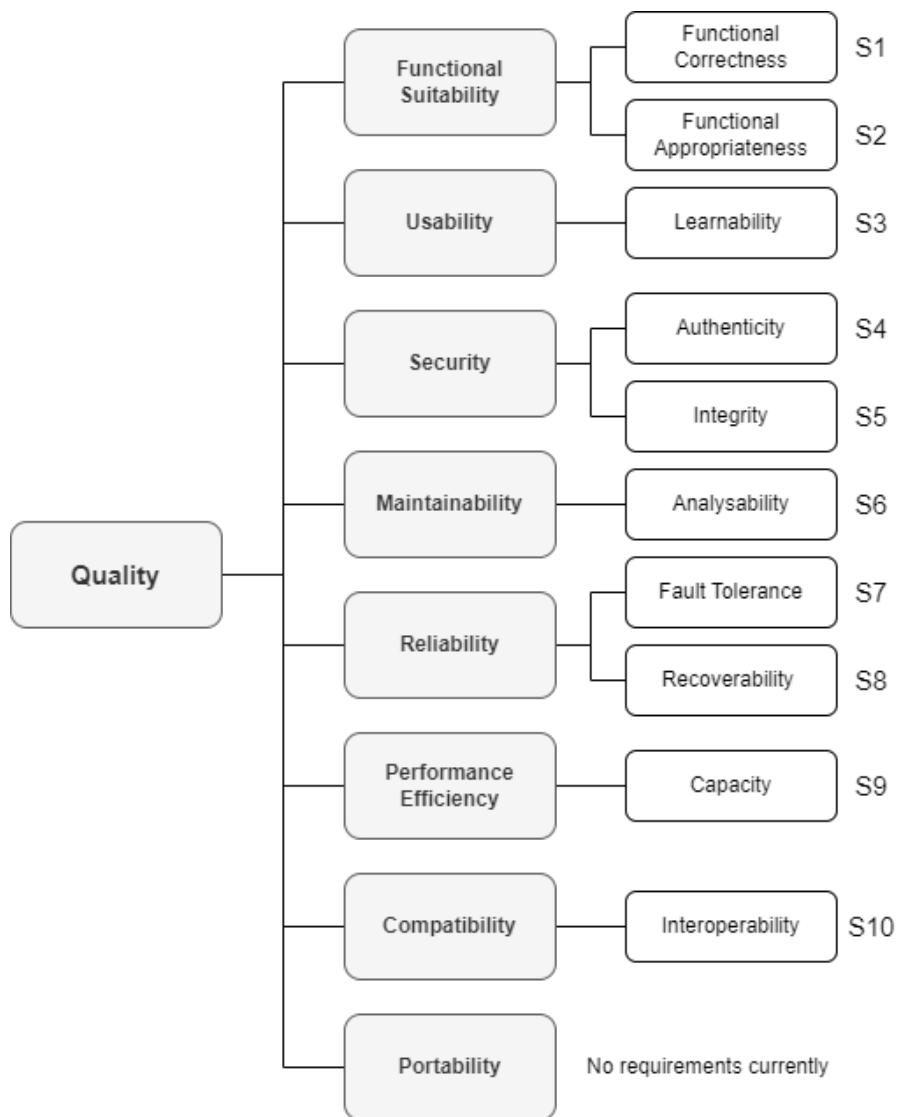
Priority	Attribute	Explanation
3	Efficiency	The expected load on the units is comparatively low, since they can only be used by one person at the same time. When implementing the system, consideration must be given to the manufacturing and hardware costs.
3	Compatibility	The software is primarily used for ordering photo products, so there should be compatibility with common image formats. It should also be possible to retrieve data from common capture devices without much effort.
4	Portability	The runtime environment, including both hardware and software, rarely changes and thus the system does not need portable at all.

Legend:

- | | | |
|---|----------------|---|
| 1 | Very important | Compromises are only possible in absolute exceptional cases. |
| 2 | Important | Compromises are only allowed if features with higher priority are strengthened. |
| 3 | Significant | Compromises are possible, as long as the core requirements are not disturbed. |
| 4 | Negligible | The characteristic is only to be considered to a minor extent. |

Quality Scenarios

The following scenarios specify how exactly the software system should behave in certain situations and what compromises may be possible.



#	Attribute	Description
S1	Functional Correctness	When a customer places an order, the system must round the final amount in a commercially correct manner.
S2	Functional Appropriateness	When a standard photo is ordered, it is subjected to compression. Compression artifacts must not be visible to the naked eye in the printed version of the photo.
S3	Learnability	A user who has not seen the software before must be able to order four different photo products within five minutes.
S4	Authenticity	When persons identify themselves as service technicians, all changes to the system must be automatically and immediately logged in an unalterable way.
S5	Integrity	If the software is started and an unknown file has been placed in the configuration directory, then the start must be aborted with an error message.
S6	Analyzability	New developers must be able to make value-adding changes to the software within one week.
S7	Error Tolerance	If the user enters invalid characters in an order, the error is highlighted to the user at the location of the error and the further order processing is blocked.
S8	Recoverability	If the device is restarted after an error, a sanity check runs and an error report is provided for further analysis, logging all errors that occurred.
S9	Capacity & Time Behavior	If a customer orders 1000 different images with a total size of one gigabyte, he must receive an order confirmation within five minutes.
S10	Interoperability	If a service technician connects a new printer, it must be operational within ten minutes without installing special drivers.

VII.4 Constraints

This chapter describes the constraints and requirements that limit software design, implementation or the development process.

Organizational Constraints

The company basically has two main products that are customized depending on the business partner in terms of features, workflows and graphical assets (→[VII.5.1](#)).

Start of development *Wizard: 07.2010 Portal: 03.2018*

Team Wizard	Five members, responsible for the implementation, adaptation and customizing of the software for the end-customer approach on the sales units.
Team Connect	Four members, responsible for processing, forwarding and accounting of orders.
Team Portal	Eight members, responsible for the portal solution where customers can view their orders and partners can view their commissions.
Team Data	Two members, responsible for the analysis and preparation of the accruing data for partners and for product maintenance.
Wizard Installations	Approx. 250 in different versions
Customers	Approx. 60 different pitch providers, approx. five different print providers, approx. 12 000 registered end users (<i>December 2022</i>)

Technical Constraints

- Technologies Used:
 - Wizard: .NET 4.8, C#, WPF
 - Portal: .NET 6, C#, Angular 11, NServiceBus for internal communication between services
- Runtime Environment:
 - Wizard: Windows 10, SQLite
 - Portal: Windows Server 2019, SQL Server 2019
- Release Cycles:
 - Wizard: About once every two to six months, depending on the partner, over the air.
 - Portal: Approx. every two weeks Technische Besonderheiten:

Note: 40 Wizard instances from before 2015 do not have a remote update capabilities, and must be updated by a service engineer until 2025. After that, the maintenance contracts expire, forcing the customer to upgrade to a remote-enabled version.

Conventions

- Code
 - The coding guidelines are located in [VII.14.2.1](#).
- Test
 - The testing guidelines are located in [VII.14.2.2](#).
- Review
 - The review guidelines are located in [VII.14.2.3](#).

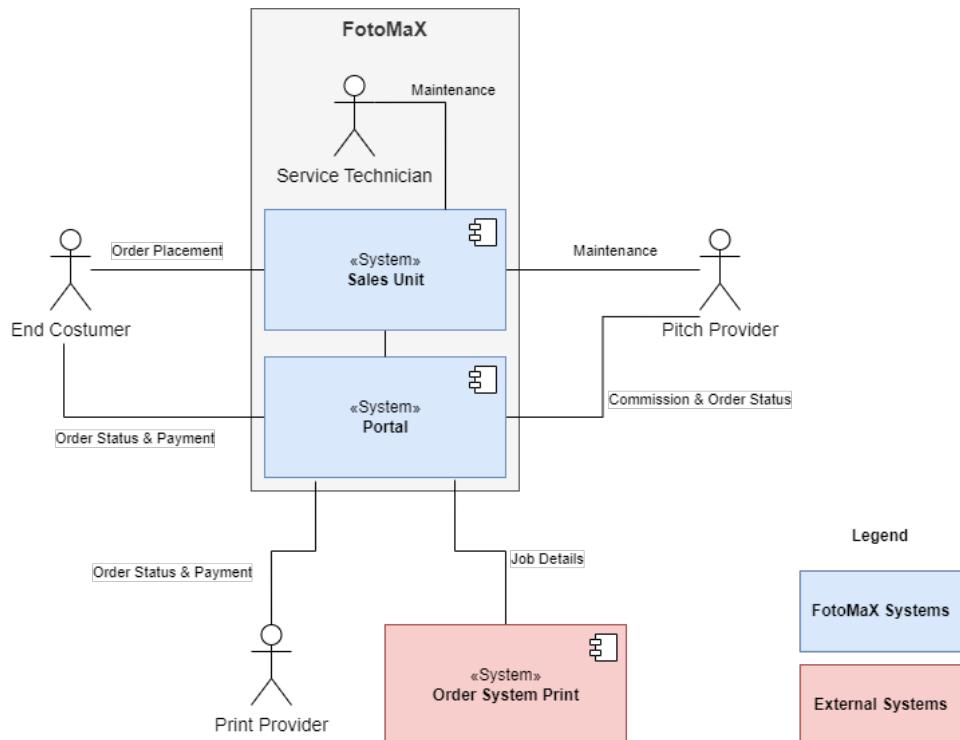
VII.5 System Scope and Context

In this chapter the system is delimited from its environment. This way user roles, external systems and the associated interactions, as well as the necessary interfaces are identified.

Business Context

The business context describes the external dependencies of the software system from a domain-oriented perspective.

FotoMaX maps two different workflows: In standalone mode, all jobs are processed on the client's premises; in connected mode, jobs are forwarded to external print service providers.



Designation	Description
End Customer	A person who uses photo printing services via a sales unit and can track them via the portal.
Service Technician	An employee of FotoMaX GmbH who performs initial submission and maintenance work on Sales Units.
Pitch Provider	An employee of the respective pitch provider who performs minor maintenance work on sales units. This includes replenishing printing paper and cartridges of the receipt printer or retrieving orders in standalone mode. In the Connected case, the parking space provider can view orders and check invoices.
Print Provider	An employee of the print provider who can view order data and modify orders.
Sales Unit	The hardware/software combination set up at a parking space provider to take orders from end customers.
Portal	Customer and partner portal via which orders can be tracked.
Order System Print	The external system of the print provider to which print orders are forwarded. Each print provider can be expected to have its own system.
Billing System / Printing Site	The billing system of the printing site provider, via which print jobs are billed.

Standalone Setup

In standalone mode the sales unit is only connected to a local printer and has no internet connection. All orders are processed and paid for within the premises of the pitch provider. The supplier rents both the sales unit and the printer from FotoMaX at a fixed monthly rate. In addition, consumables such as ink and paper are billed according to usage. In addition to printing the ordered items, the sales unit also prints a receipt, which has to be paid by the end customer at the checkout of the pitch provider. FotoMaX is not involved in the actual billing process.

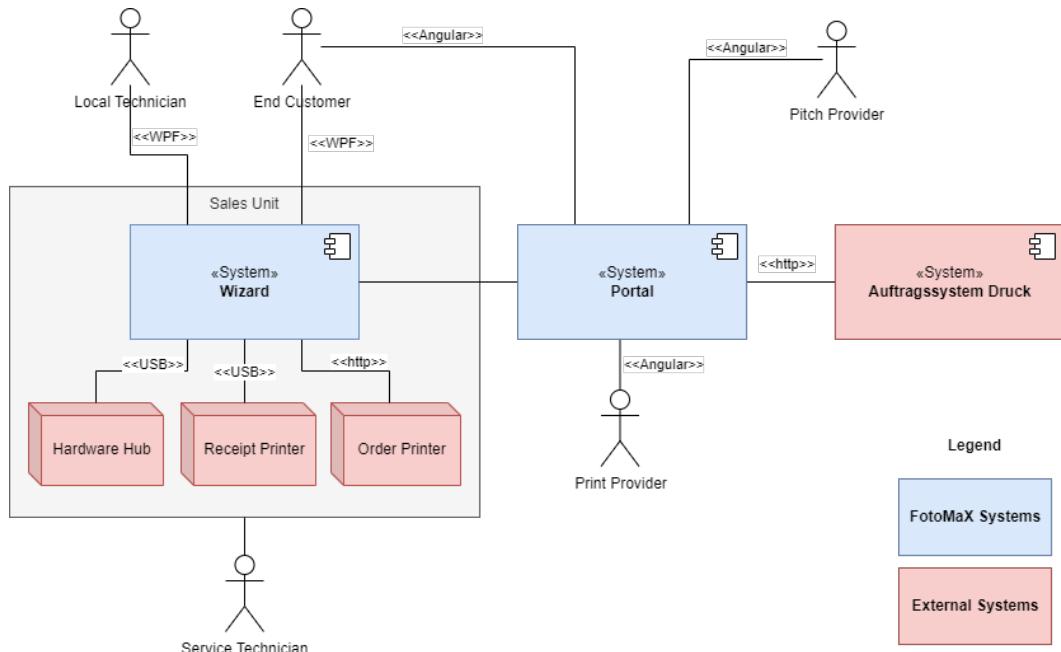
Connected Setup

The connected setup allows end customers to place more extensive orders and track them online. In addition, the orders are not executed exclusively on the premises of the provider's premises, but also at a corresponding print provider on additional products such as mugs, t-shirts or similar articles. In a partially connected setup, the billing is done by the pitch provider.

In a full connected setup, the orders are then either paid for by EC or credit card at the device or the customer receives an invoice via e-mail. In this case, the pitch provider is not involved in the billing of an order, but receives a fixed monthly stand fee, and commission on all sales. They can view their earnings at any time via the portal.

Technical Context

The technical context highlights the technical environment in which the software system exists and the respective interfaces to external systems.



Element	Description
Service Technician	An employee of FotoMaX GmbH, who performs maintenance work on sales units on site.
Local Technician	An employee of the respective sales pitch provider, who performs minor maintenance work on sales units. This includes replenishing printing paper and cartridges of the receipt printer or retrieving orders in standalone mode. The wizard's UI is used to confirm all maintenance tasks. Local technicians have no access to the sales unit's operating system.
End Customer	A person who uses photo printing services through a sales unit, or views order data on the customer management website.
Wizard	The software that takes orders and initiates prints either locally or remotely.
Hardware Hub	The central control unit via which various hardware components can be addressed. This is used to read images from cameras, USB sticks or memory cards. The connection is made via USB and is standardized for all sales units.
Receipt Printer	A thermal printer used to print out receipts for orders that then have to be paid at a cash register.
Order Printer	FotoMaX can forward print orders directly to printers of the location provider and thus trigger immediate printing. Printers can be connected via USB or as network printers.
Pitch provider	An employee of the pitch provider who can view orders and check invoices.
Print provider	An employee of the print provider who can view and, if necessary, change job status and orders.
Portal	The web portal that contains customer administration, order administration, and partner administration.
Job system	The external system of the respective print provider, to which jobs are forwarded. The connection varies greatly depending on the system and is therefore assumed in simplified form only as HTTPs.

The hardware is connected to the Wizard via the API of the operating system during a standardized configuration process after the manufacturing of the units. Thus, no special protocols have to be considered. The only exceptions are order printers: when integrated as network printers, those have to be configured separately.

VII.6 Solution Strategy

The Wizard was implemented in 2010 as a monolith, based on the .NET Framework and Windows Presentation Foundation. After a comprehensive architecture review, it was decided in 2019 to restructure it as a modulith.

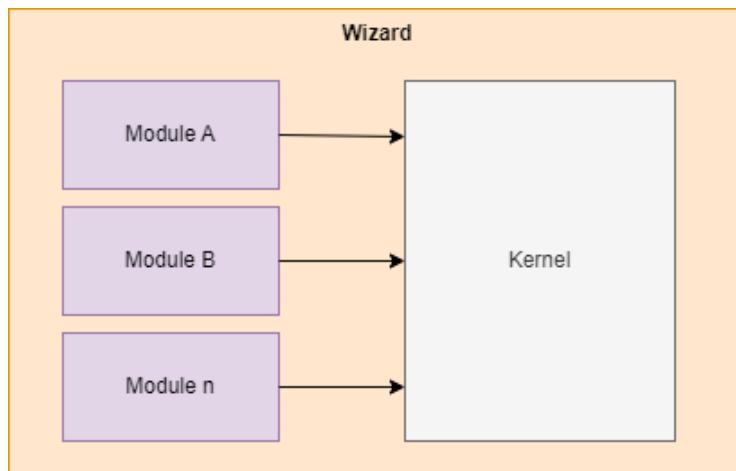
Since FotoMaX is a white label solution, this base application is heavily customized depending on the partner. Thus, while there are software modules that apply to all pitch providers, there are also some that are loaded only for specific providers.

This chapter explains both the associated base application and the migration strategy, which describes how the previous monolith is transformed into a modulith.

The Modulith

Basic Design

As a basis for the modulith, a framework application was created based on the Prism Framework (→[VII.11.4](#)), which provides general features for central services such as user administration, settings management, etc.

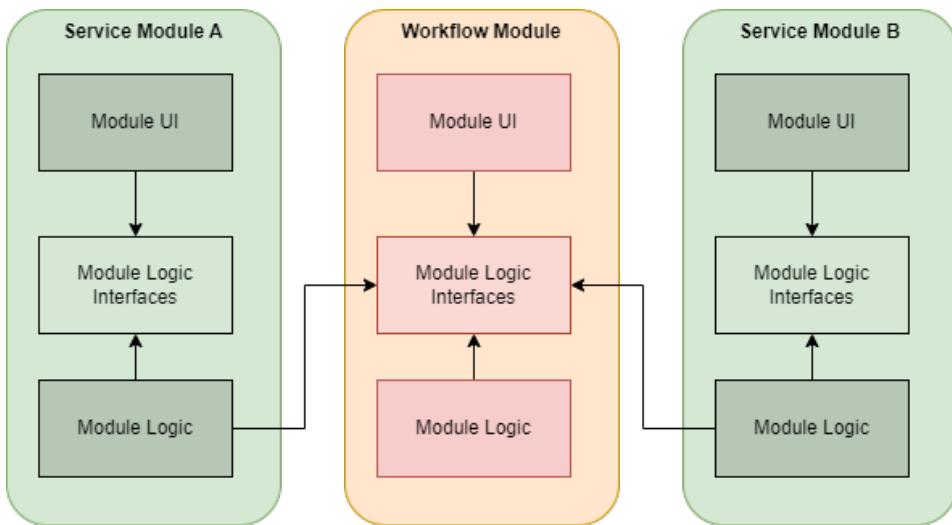


Module Design

A module, in the sense of a modulith, is a collection of functionalities with the same concerns. Within a module, the technical layers are separated from each other via interfaces. Modules can also use the functionality of other modules via interfaces. Dependencies between modules are monitored by static code analysis, to avoid harmful dependencies and complex dependency trees. The architecture distinguishes between two types of modules: Workflow modules and Service modules.

Workflow Modules: These are central modules that map domain specific processes and workflows. As a rule, they also have more complex user interfaces that are implemented as a workflow within the wizard.

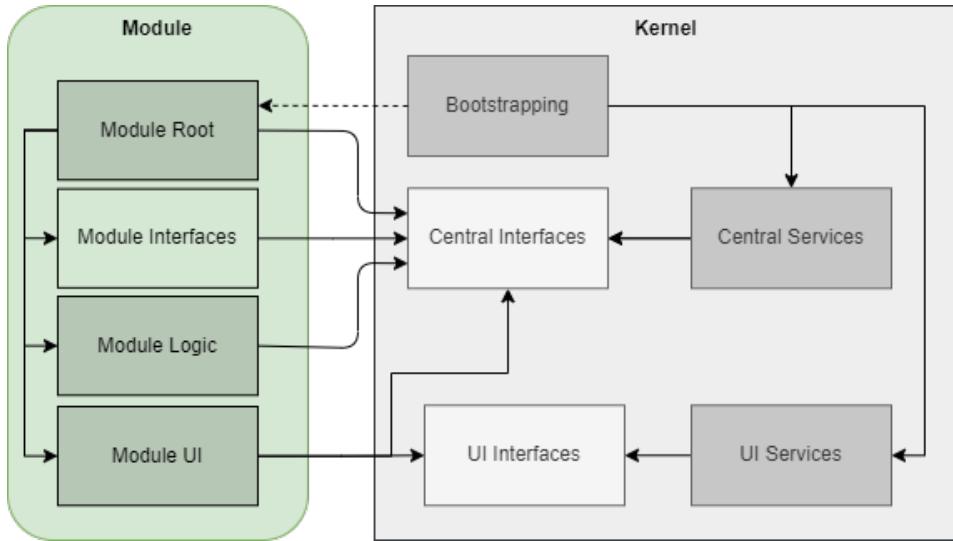
Service Modules: These are modules that add specific functionality to the workflow. These include, for example, assets of the respective customer such as logos, color designs, etc.; but also include functions that can change depending on the characteristics of the sales unit.



Module Integration

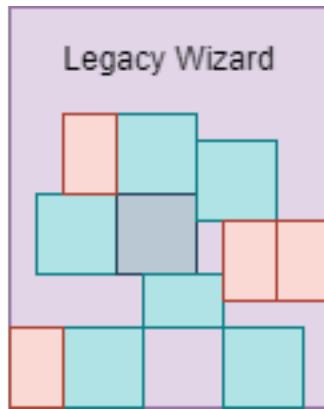
The kernel integrates the modules into the application during a bootstrapping process. The modules can access central interfaces, and they have a root element that is called by bootstrapper to control their initialization. Modules are strictly prohibited

from accessing the concrete implementation of either other modules or any central services. Rather, they have to request all services via dependency injection, using constructor injection, meaning that the service interfaces are injected as parameters into the module constructors.

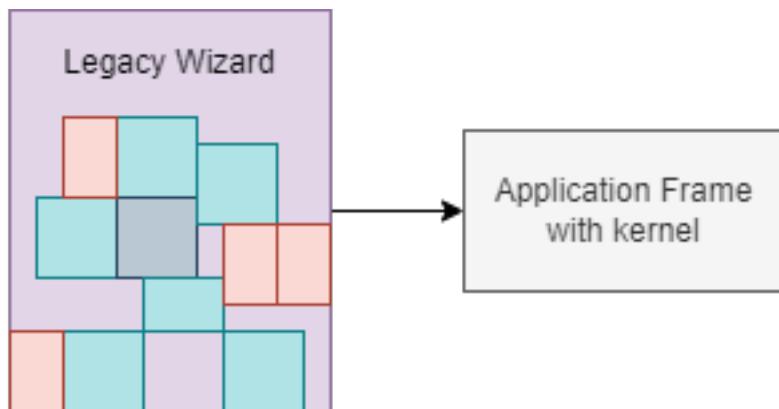


Migration Strategy

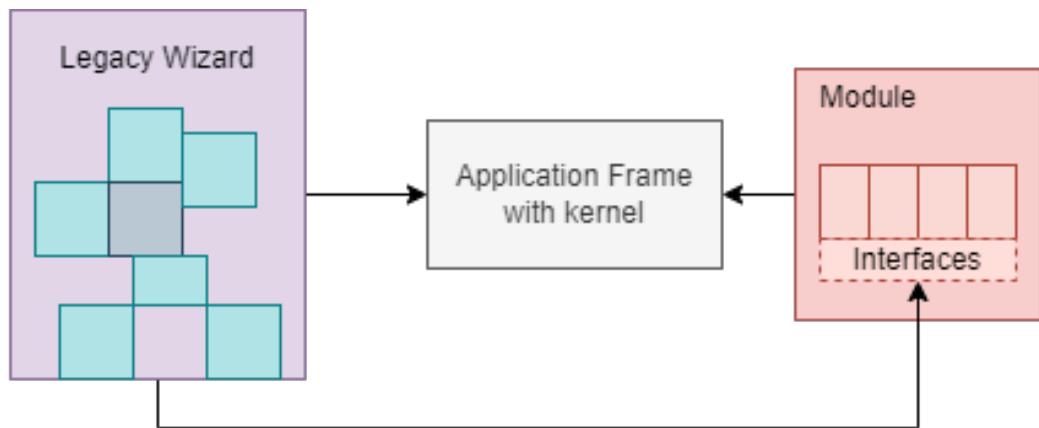
In 2019, the architecture of the Wizard had a highly coupled structure, with neither functional nor technical layering. The following diagram depicts this situation by using the same colors for components with the same responsibility, while the different sizes and distributions to represent the unstructured nature of the system.



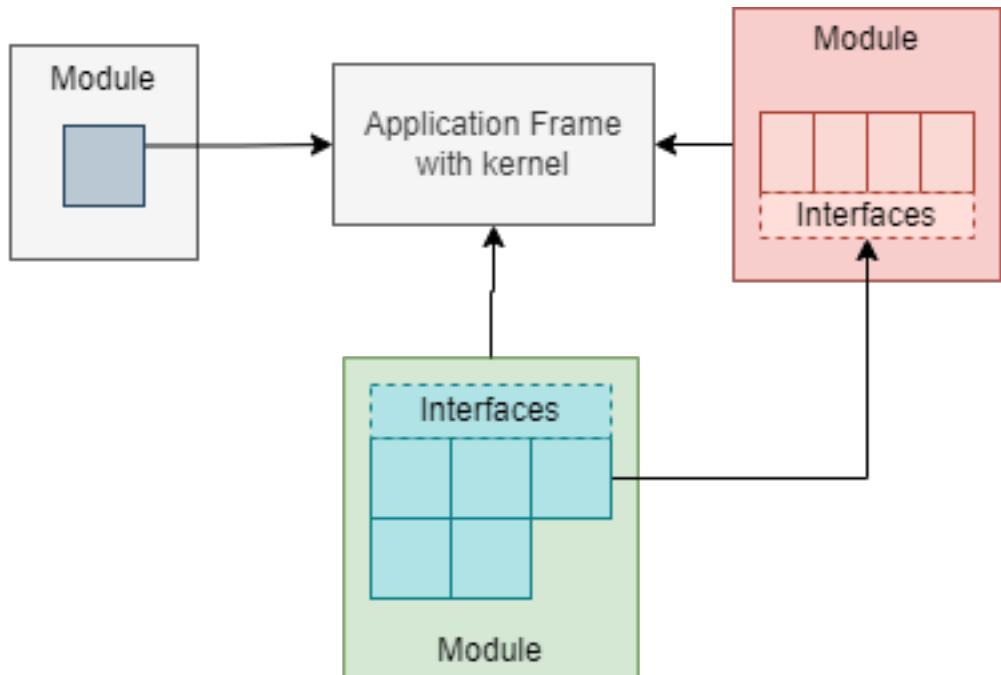
To decompose the monolith, first the various responsibilities within the codebase are assessed. Then a strategic decomposition is prepared, and a frame application is created, that can incorporate the modules.



Subsequently, the Legacy Wizard is included as a closed module in the frame application, so that the startup is no longer performed by the Legacy Wizard, but by the kernel of the frame application.



Based on the strategic considerations, the next step is to prioritize a business or technical context, and extract the associated components into a separate module, by extracting its public access points to an abstract interface layer. This interface layer is then referenced in the legacy code while its logic is restructured.



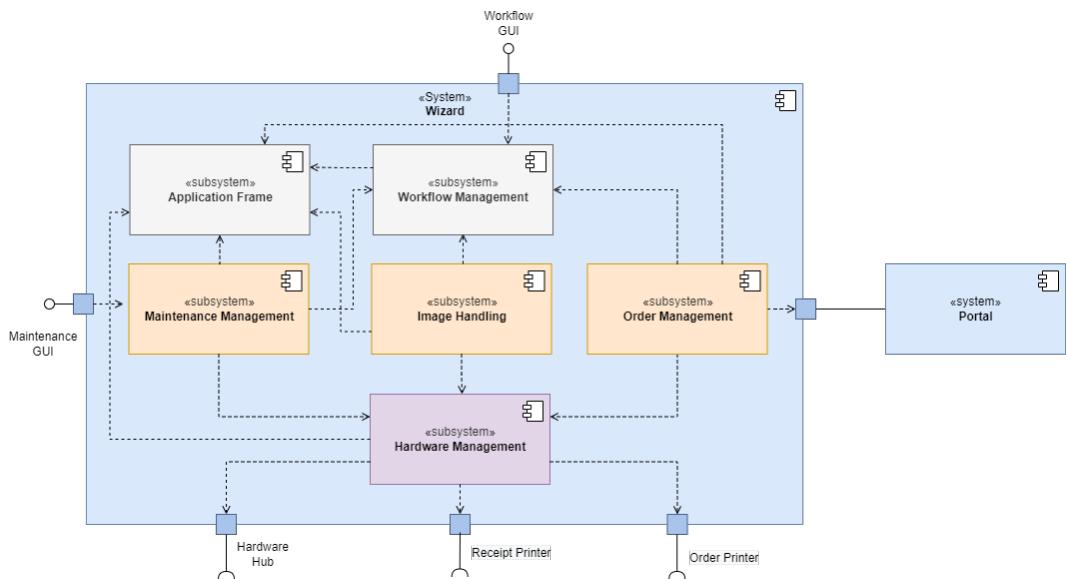
This procedure is continued until either the entire Wizard is disassembled, or only such components are left for which restructuring is not worthwhile.

VII.7 Building Block View

The structural design and its decomposition into building blocks of various granularity is described in this chapter. It uses different detail levels to zoom in from abstract structures into increasingly detailed and concrete elements.

Level 1: Subsystems of the Wizards

The following illustration shows the internal components of the Wizard and their interdependencies as well as their interaction with external interfaces. The latter partly interact with other software, hardware or even the different user groups on the basis of a graphical user interface.



There are three different types of subsystems in the wizard:

- infrastructure (gray),
- logic (orange),
- and hardware connection (purple).

All versions of the sales units used the same infrastructure components. The logic components are provided with their actual functionality at runtime through a configuration that is specific to the sales unit. The hardware connection in turn decouples all other components from the possible printers and connection options of the sales unit.

Application Frame

Purpose	Provides the functionality of a frame application that combines all other components at runtime. Also includes components for cross-cutting concerns such as settings management, logging, error handling, inversion of control, user interaction, etc.
Interfaces	No external interfaces but C# interfaces are provided for the various components.
Special Features	Much of the functionality is provided by the Prism framework and partially encapsulated by custom interfaces to reduce dependencies on the framework, so it is possible to replace the WPF and Prism in the future. Since the Application Frame is primarily a connection to the Prism Framework and a collection of loose components, there is no white-box representation of this in this documentation.

Workflow Management

Purpose	Contains the functionality necessary to cover ordered workflows with the software. This includes navigating forward and backward through work steps, branching in workflows and canceling operations.
Interfaces	Workflow Management offers few UI elements of its own as it provides the framework for the actual workflows. However, it is actually the main point of interaction for the end user via the workflow steps it aggregates.
Special features	Workflow Management cannot work without logic components and workflow configuration as it would only display an empty frame.

Maintenance Management

Purpose	Contains all features that are needed to read and change the machine state, getting information about consumables and help with calibration.
Interfaces	Maintenance Management interacts with service technicians via its own GUI, which is unlocked via a special hardware dongle. Additionally, it has access to various status information of the connected devices via the hardware management.
Special Features	Service technicians of FotoMaX can bypass the Maintenance Management to directly access the operating system.
Open points	Older versions of the Maintenance GUI could be accessed via the general UI. The functionality required for this can still be found in the source code for compatibility reasons, however, it is no longer linked in the UI.

Image Handling

Purpose	Image Handling contains all functions that can be used to read, process and modify images. In addition, it also contains the corresponding work steps, which are triggered via workflow management at runtime and displayed to the user.
Interfaces	No external interfaces, since all work steps are called via the workflow engine.
Special Features	A large part of the functionality is provided by the Prism framework and partially encapsulated by its own interfaces to reduce dependencies on the framework, since in the future a move away from WPF and thus from Prism is possible. Since Image Handling is a collection of loose components, there is no white-box representation of this in this documentation.
Risks	The code in Image Handling is very disorganized and needs a thorough restructuring. Especially the exchange of image data is solved via a global context and causes confusion. See also technical risks.

Order Management

Purpose	Order Management handles the actual ordering. This is done via the portal in the connected case or via the local printers in the standalone case.
Interfaces	In the connected case, Order Management establishes a connection to the portal and forwards the incoming order to it. In the standalone case, the image data is sent to a photo printer and the proof of purchase to a connected thermal printer. Hardware Management is then addressed for this purpose.
Special Features	Order Management must read from the configuration which work mode the sales unit is in and then decide independently how it processes orders.
Open points	There is only one dialog in which the entire order is summarized. This is used equally on all Sales Units. However, the pitch providers have already stated needs for adjustments here.

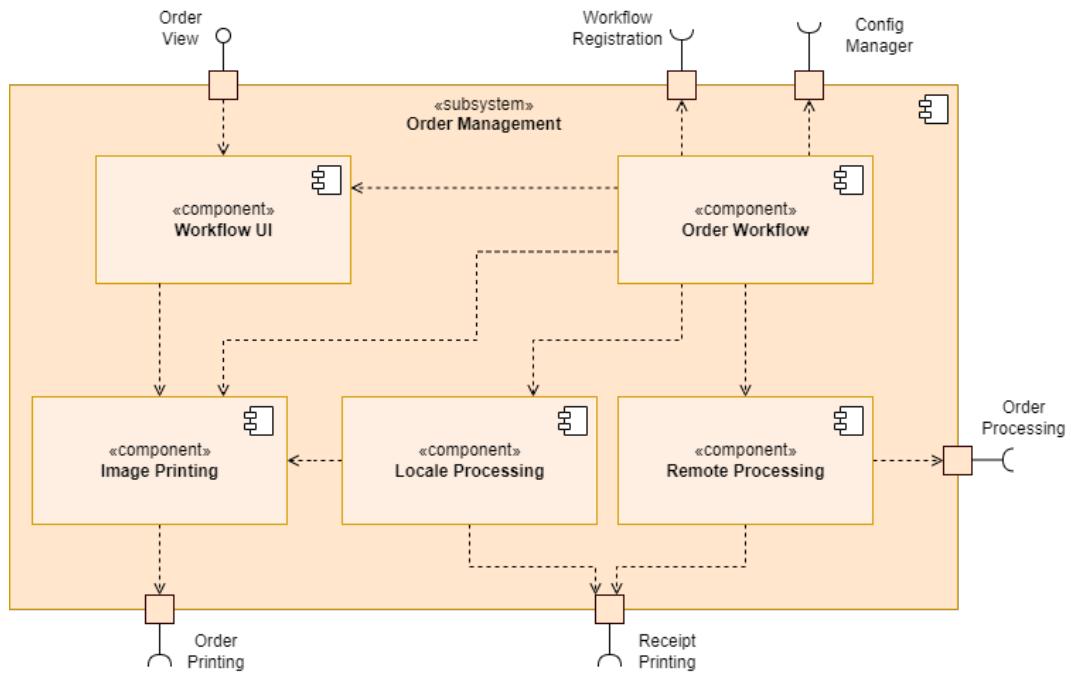
Hardware Management

Purpose	The hardware management abstracts the hardware accesses for all other subsystems.
Interfaces	Primarily the operating system's own interfaces are used to access the hardware.
Special Features	Hardware Management does not have its own UI. Since Hardware Management is primarily a collection of loose components, there is no white-box representation in this documentation.

Level 2 – White-Box View of the Subsystems

On these pages, individual subsystems of the Wizard are described in more detail.

Order Management White-Box



Order Management must decide whether the system is in connected or standalone mode based on a system configuration.

Name	Responsibility
Workflow UI	Contains all views that are dynamically integrated into the main UI.
Order Workflow	Aggregates all components of the order management and attaches them to the frame application and workflow based on a configuration.
Image Printing	In standalone configuration, this executes the printing of photo products at a local printer.
Local Processing	Takes over all processes that are necessary to execute a local print job. This also includes the issuing of a receipt.
Remote Processing	Handles all processes that are necessary to have a print job executed by a print partner. This also includes the issuing of a receipt.

Workflow Management White-Box

This part of the documentation has not been further elaborated as it does not add any value to the actual purpose of the documentation.

Maintenance Management White-Box

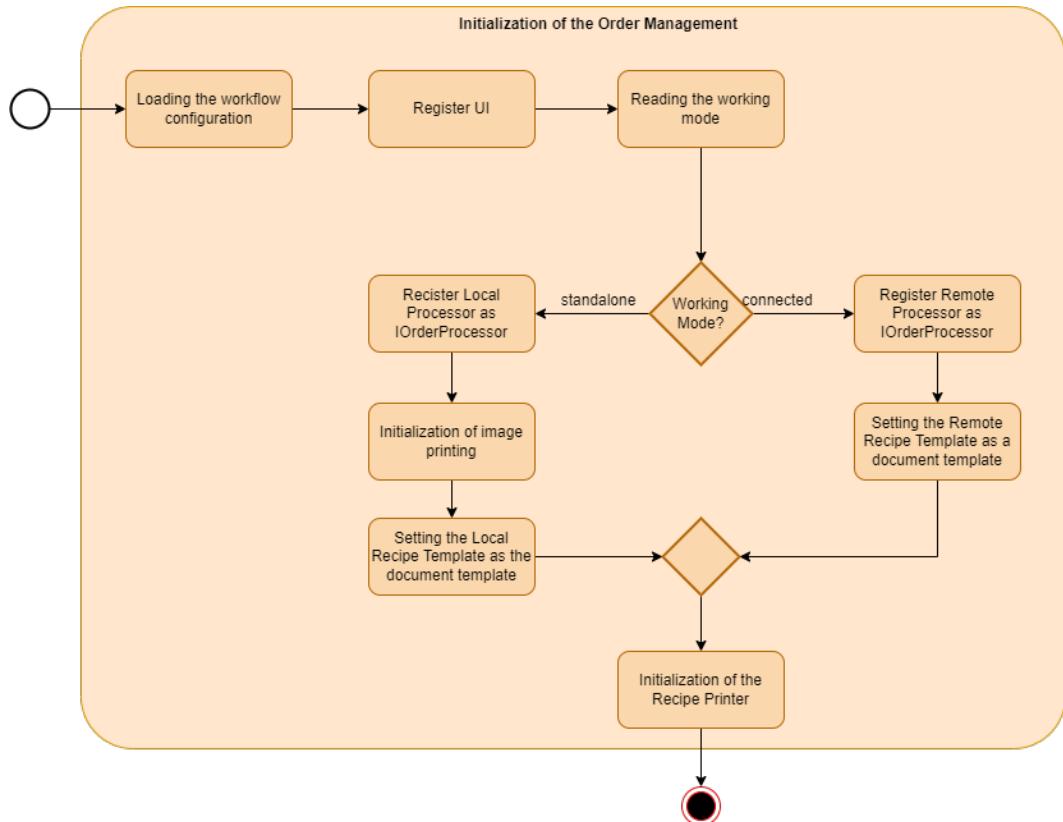
This part of the documentation has not been further elaborated as it does not add any value to the actual purpose of the documentation.

VII.8 Runtime View

All dynamic relationships that are to be highlighted are described in the runtime view chapter.

Application frame: Steps of the module initialization

Below is the sequence that is run through during program startup for each module to initialize itself and its managed components.

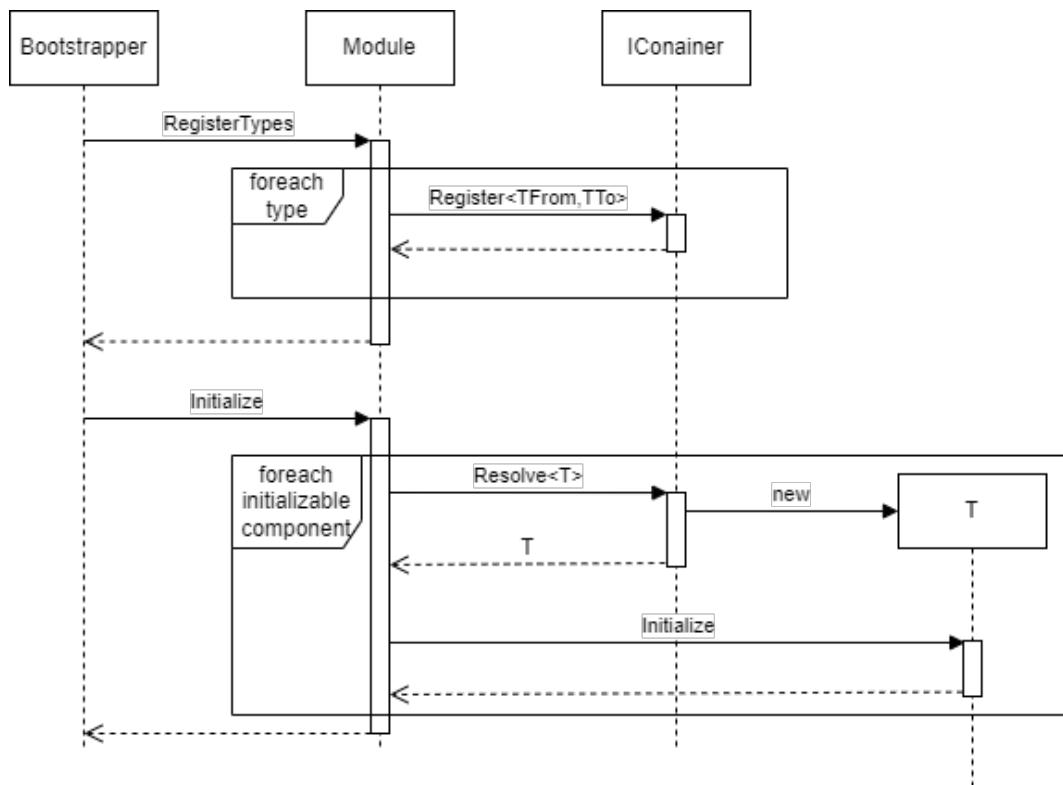


Not every type has to be initialized by the module itself. This is only necessary for module components which actually require a special initialization. The component

initialization should happen here if possible asynchronously around the start not to hinder.

Order Management: Initialization

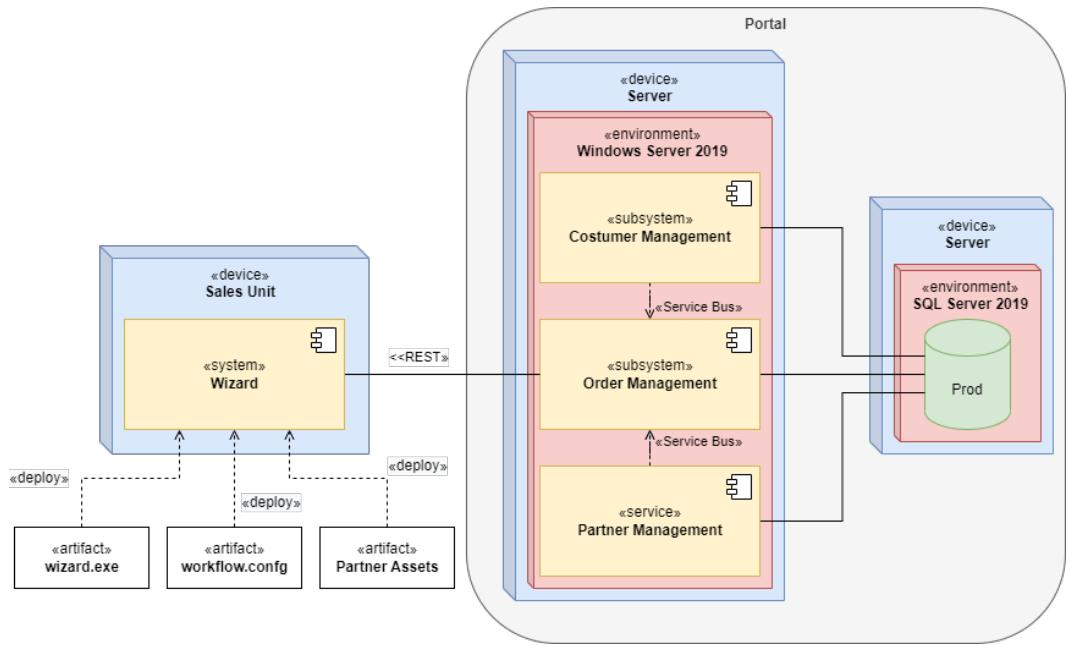
Like any other module, the Order Management is initialized during the program start. Depending on whether the Wizard is operated in standalone or connected mode, a different procedure for the order process has to be chosen. The following image shows the processes, that have to be executed during the initialization of the workflow module to load the correct components.



Note: The same document printer is always installed in the sales unit, therefore it can be decided via the selection of a template whether a remote order or a local order is to be issued.

VII.9 Deployment View

The deployment view clarifies the runtime environments in which the individual components of the system are installed and executed.



For the sake of clarity, the above illustration only shows the production environment in a simplified form. This does not contain virtualization environments, development environment, staging environments, and fallback servers. A detailed overview can be found in the Infrastructure documentation of the IT department, which takes care of the operation.

The production environment is mirrored in its structure as a test environment. This has two databases, Test and Dev.

Artifact	Description
wizard.exe	Executable file of the wizard, which is installed by a service technician on a sales unit before delivery. Updates are done by the service technician on site.
workflow.config	Specially adapted configuration of the workflows for the respective parking space providers.
Partner Assets	Logos, fonts, theme, etc. in the corporate identity of the parking space provider.
CostumerManagement.svc	Installation package for the Costumer Management subsystem installed on the runtime server.
OrderManagement.svc	Installation package for the Order Management subsystem installed on the runtime server.
PartnerManagement.svc	Installation package for the Partner Management subsystem that is installed on the runtime server.
Prod DB	Production database with all customer data. It is backed up incrementally once a day; a full backup is created every Sunday.
Test DB	Special database with anonymized test data. This is used for acceptance tests by the departments and for quality assurance.
Dev DB	Developer database, which is reset irregularly and is used by the developers to implement new features. It contains only anonymized test data.

VII.10 Cross-cutting Concepts

This chapter summarizes all concepts that cannot be assigned to a specific application layer, neither from a functional nor a technical point of view.

Exception & Error HandlingCrosscutting Concepts

This chapter describes what “errors” are, and how they should be handled.

Types of “errors”

Different types of errors can occur within software, therefore not everything that is commonly referred to as an error is also an error in the sense of software development.

We can distinguish between the following scenarios:

- Not visible to the user
 - Can be resolved by the software itself
 - Cannot be resolved by the software itself
- Visible to the user
 - Can be resolved by the user
 - Cannot be resolved by the user

This distinction shows that, depending on the situation, handling of unexpected behavior can be passed on to a higher escalation level, which may then contribute to a resolution of the situation. For example, if a file cannot be opened by the software, it makes no sense to crash the entire application with an exception. An appropriate error description should be presented to the user instead, letting them intervene. The user can then decide how to handle the situation and, if necessary, refrain from loading the file at all.

Exceptional Situations vs. “Exceptions”

Exceptions are the common means of choice in .NET to indicate errors. They have the advantage that they can be checked and logged using appropriate analysis tools. In addition, they automatically contain various environmental information such as a stack trace, the time of occurrence and others. The disadvantage is that they are not necessarily known to the caller. Thus, they may be overlooked, not handled and in the last instance close the entire application. In addition, the information that exceptions contain is not necessarily helpful for the user since it is very technical.

Note: Exceptions should be reserved for exceptional situations or pure programming errors. If an action fails due to a foreseeable event, then that event should be handled appropriately, and no exception should be thrown.

A good example of an unusual situation can be found in switch case statements, where the list of possible values might change over time. As a result, it is possible that case statements no longer cover all possible combinations. To detect such a problem at an early stage, a corresponding exception can be thrown in the default branch.

```
1 decimal calculatedPrice;
2
3 switch (customer.Status)
4 {
5     case CustomerStatus.Normal:
6         calculatedPrice = CalculatePrice(order);
7         break;
8     case CustomerStatus.VIP:
9         calculatedPrice = CalculateSpecialPrice(order);
10    break;
11    default:
12        throw new InvalidEnumArgumentException($"Unknown customer status!\\" +
13    ");
14 }
```

Not Exceptional

Exceptions are intended for exceptional situations, but how do you deal with non-exceptional situations? A result object is provided for this purpose. This contains

both technical error descriptions and error descriptions that a user can follow. The result object can therefore also be used to inform users about a problem situation and, if necessary, to prompt them to take countermeasures.

In the following example, this is illustrated by first checking whether the correct data has been transferred or not in the case of a division. If not, this is communicated to the user so that they can correct their input if necessary.

```
1 public Result<float> Divide(int dividend; int divisor)
2 {
3     if(divisor == 0)
4     {
5         return Result.Failed<float>("The divisor was zero thus division was not possible.", "Please change the input.");
6     }
7
8
9     return Result.Success<float>(dividend / divisor);
10 }
```

The example is deliberately simplistic and may lead to discussions. Whether a result object should be used or not is in fact very much a contextual decision. Within components it is quite possible to return concrete data types, as far as no errors can occur at all or an error automatically weighs so heavily that it must come to an exception.

Handling Errors

There are several ways to handle errors. In the following section, we will briefly discuss these to give you a better sense of what to think about in each case.

Ignore

If possible, error conditions should not occur in the first place. If an error is so unlikely that concrete handling seems too costly, then it should at least be logged. However, this should be discussed with the product owner and quality assurance.

Retry

Based on the context, it can be quite helpful to simply try an action again if it does not succeed instead of throwing an exception immediately. However, this should be discussed with the product owner and quality assurance.

Logging

Logging is a very extensive topic and has therefore been moved to a separate chapter (\rightarrow [VII.10.3](#)).

User Interactions

In case of an error, interaction with the user is possible in two ways.

- **Notifications** - information for the user without direct influence on the processing. Example: “System configuration faulty, Sales Unit is shutting down!”
- **Confirmation** - Information for the user, which requires an input from them, whereby they can influence the processing. Example: “File could not be read. Should it be tried again? Yes/No”

Special case: Final Exception Handler

It may happen that exceptions are not caught. In that case they end up in the final exception handler. This logs the error and shuts down the application in a structured way. Such errors should not occur in a production environment!

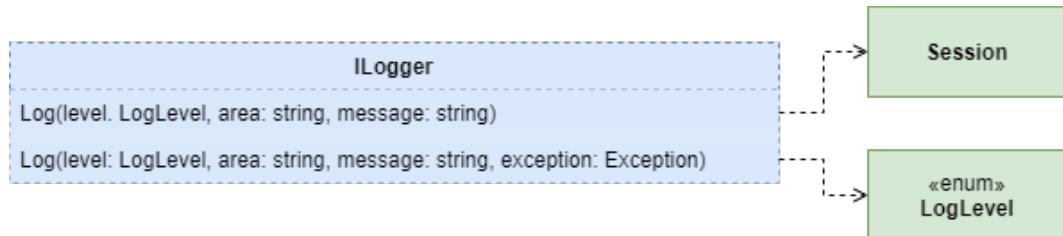
Logging

Why should you log?

When searching for the causes of errors, it is very helpful to know how the software behaves in its real environment. This is particularly important in the context of the wizard, since the sales units represent closed systems that are difficult to access from the outside. Remote debugging or similar are not possible at the current time. The production logs are therefore the only way to obtain information about the use and status of the sales unit.

How should you log?

The ILogger interface is used for logging in the portal and in the wizard. This is provided centrally and can be requested by any class via its constructor.



Log Level

Fatal	A Fatal log entry is written whenever the stability of the entire software is negatively affected. For example, a Fatal Log should always be written by the Final Exception Handler if exceptions occur that were not handled.
Error	Error log entries indicate that something has happened that should not have happened, without automatically affecting the stability of the software. The most common case for error logs are caught exceptions that could be handled.
Warning	Warnings are not necessarily errors, but indicate that something is wrong. They can occur, for example, if the connection to the printer is lost. In such a case, the code writes a warning and tries to reconnect after a few seconds. If it cannot establish a connection after several further attempts, it logs an error instead of a warning. If the error persists for a long time, it would result in a Fatal log entry.
Information	Information can basically be anything that is useful to understand the relationships in the software. This is not error information since that is already covered by other log levels.
Debug	Debug logs are used when debugging by developers. This log level is not available in the production environment and is only used in the test environments.

What should you log?

In addition to the log level and possible exceptions, the context in which the information occurred and when the log was written must also be logged. The more meaningful the logs, the easier it is to analyze them later.

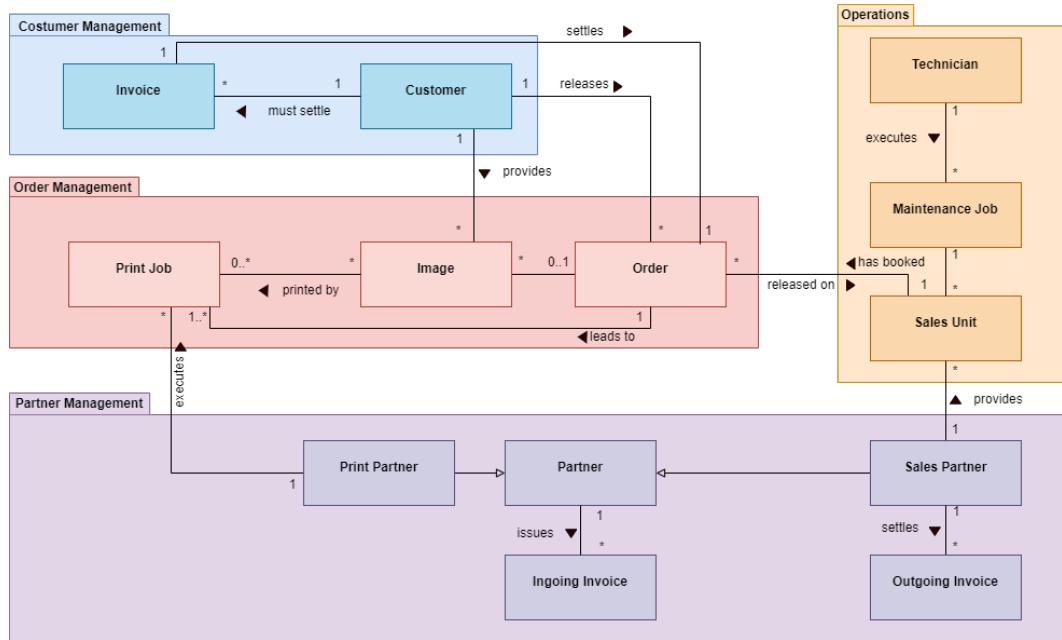
What should not be logged?

Logged data must not allow any conclusions to be drawn about individuals, including contact details. Furthermore, it is not permitted to write account details or similar sensitive information in the log. In order to allow better traceability through the different layers or services, a unique session ID is automatically assigned to each order transaction, and this is automatically part of each log message.

Domain Model

The domain model is a technical data model that helps both in the design of interfaces and in the communication between the various project participants. The following domain model describes the considerations in the context of the business processes of the FotoMaX company.

The following domain model is highly simplified and serves the purpose of a practical training. Therefore, it does not claim to be technically correct in its entirety. For example, large parts of *Customer Management* and *Partner Management* are missing. The domain *Operations* will only be found in this chapter and in no other, as it would have greatly complicated all other examples.



Entity	Description
Customer	Customer who initiates an order and provides images. Both activities can be done via a website, app or a sales unit.
Invoice	End customer invoice that settles the amount due resulting from an order. In the standalone case there is no connection to the customer.
Order	Order that is triggered by a Customer. If it was booked through a Sales Unit, a connection to the Sales Partner that enabled the order is established through that Sales Unit.
Image	Image provided by the customer. If provided via app or website, an order does not necessarily have to be triggered. Images can be printed multiple times and therefore belong to multiple Print Jobs. There are local print jobs (standalone) and remote print jobs (connected).
Print Job	Contains images that are forwarded together with various parameters to a print partner for the actual printing. An order on a sales unit automatically leads to the creation of a print job.

Entity	Description
Partner	Prints print jobs and invoices the related services as Incoming Invoice.
Print Partner	Prints print jobs and invoices the related services as Ingoing Invoice.
Sales Partner	Establishes sales units. In the connected case, receives a commission for each order placed, which is paid as an Incoming Invoice. In standalone mode, a monthly fee is issued as an outgoing invoice.
Incoming Invoice	An invoice issued by a partner and to be paid by FotoMaX.
Outgoing Invoice	An invoice issued by FotoMaX to a partner.
Sales Unit	A sales unit that is set up at a sales partner and triggers orders.
Maintenance Job	A maintenance order to a technician for one or more sales units.
Service Technician	Service technician who accepts and processes maintenance orders for sales units.

VII.11 Architecture Decisions

Fundamental decisions should not be made based on gut feeling, but as empirically as possible. This chapter collects all the relevant information for design decisions.

ADR 001 - Base Technology Selection

This documentation is primarily intended to show what an [Architecture Decision Report^a](#) can look like. This does not mean that such a simple description is sufficient for such a serious and long-term decision as in the present case. Especially in this case, a deep analysis is of course worthwhile, and the analysis results should then also be filed at this point!

^a<https://adr.github.io/>

Context

The basic technology of a software determines not only which APIs are available for development, but also which ecosystem will have an influence on the software in the long term. This ecosystem includes both free and proprietary libraries and frameworks. The selected technology also determines future viability, licensing costs and maintenance efforts.

In order to prevent technological fragmentation as much as possible, FotoMaX must have a base technology that can be used in the long term on the sales units, the portal, and possibly also on the end users' (mobile) devices. In addition, it must be stable and cost-effective to avoid unexpectedly high costs of maintenance, operation, and licensing.

In the course of setting up a new customer portal, a decision must be made about which basic technology should be used.

State

Accepted.

Decision

Date of adoption of the resolution: 15.06.2015

A decision was made in favor of .NET and the Microsoft ecosystem as the basis for software development at FotoMaX. .NET has been actively maintained by Microsoft for many years, and it allows the development of rich clients as needed for the sales units, websites, and mobile applications.

In addition, the decision makes it possible to continue maintaining the existing source code for the sales units without having to develop it from scratch. It is also possible to migrate the portal to the Azure Cloud with comparatively little effort.

Consequences

Higher licensing costs are generally to be expected when using .NET and Microsoft technologies as opposed to other technologies. These include the costs for development environments, database servers and operating systems. However, economies of scale can be exploited here by standardizing corporate IT.

ADR 002 - Base Architecture of the Wizard

Context

The architecture review of May 12, 2019, determined that the architecture of the Wizard is too difficult to maintain. Therefore, an alternative architecture is to be developed. The following architecture samples have been compared to the quality requirements to enable a selection. Please note: The evaluation was made based on an existing and complex software structure, which excludes a completely new development.

The following comparison is to be understood as an example of how architecture patterns can be compared with the aid of a decision matrix. The evaluation does not make a claim on completeness and correctness!

The following architecture patterns were compared:

Monolith	is an architecture pattern where the software system can only function as a whole during development and runtime. Separations between the (technical) concerns must be enforced organizationally, as they are separated to a limited extent (all code is developed in the same repository). This means less organizational effort for changes but increases organizational effort for maintenance.
Modulith	is a special type of monolith where essential parts are physically separated from each other by either being developed in multiple code repositories and/or the software is split into multiple parts at compile time but executed and deployed as a whole. This definition follows a hybrid of the terms “allocation monolith” and “run time monolith,” which are also commonly used. The modulith helps reduce maintenance costs compared to a monolith, but benefits from the runtime advantages of monoliths due to less complex runtime environments compared to service-oriented solutions. Moduliths are often an important step in migrating from monoliths to service-oriented architectures because they help to partition the monolith.
SOA	According to Wikipedia, “The term SOA has slightly different meanings. The first meaning is quite broad, describing it as a software design that divides functions into distinct units that developers make accessible over a network so that users can combine and reuse them when building applications. The other meaning describes it in more detail, following specific protocols and responsibilities.” This comparison is based on the broader definition.
Microservices	can be considered as a specific type of SOA. The distinctive feature of microservices is that they are decoupled top-down using specific protocols, represent specific business units, and can be deployed independently of each other. SOA, in contrast, tends to be interpreted as having only a collection of features as separate functionality.

Analyzability

Monolith	Modulith	SOA	Microservices
Poor	Good	Medium	Medium

For monoliths, the entire structure needs to be understood as it may not be decoupled. SOA and Microservices become complicated when the whole picture needs to be

understood. This can be done with special tools, but is more difficult than with moduliths, where the software exists as a whole.

Appropriateness

Monolith	Modulith	SOA	Microservices
Not applicable	Not applicable	Not applicable	Not applicable

This quality attribute must be addressed by specific implementation details that are not part of the architecture pattern.

Authenticity

Monolith	Modulith	SOA	Microservices
Not applicable	Not applicable	Not applicable	Not applicable

This quality attribute must be addressed by specific implementation details that are not part of the architecture pattern.

Completeness

Monolith	Modulith	SOA	Microservices
Not applicable	Not applicable	Not applicable	Not applicable

This quality attribute must be addressed by specific implementation details that are not part of the architecture pattern.

Confidentiality

Monolith	Modulith	SOA	Microservices
Good	Good	Medium	Medium

This quality attribute must be addressed by specific implementation details that are not part of the architecture pattern. However, services need to be secured more at runtime because they are deployed independently and separated from each other. This means a larger attack surface.

Fault Tolerance

Monolith	Modulith	SOA	Microservices
Poor	Medium	Good	Good

Because components in service-oriented architectures are physically separated from each other, faults can have less impact. Moduliths can compensate for faults somewhat better than monoliths, but are ultimately similarly vulnerable.

Installability

Monolith	Modulith	SOA	Microservices
Good	Good	Medium	Poor

Moduliths work as “deployment monoliths” and thus have the same advantage. The infrastructure for installing microservices is more difficult to set up than the infrastructure for just a few services.

Integrity

Monolith	Modulith	SOA	Microservices
Good	Good	Medium	Medium

This quality attribute must be addressed by specific implementation details that are not part of the architecture pattern. However, services simply need to be secured more at runtime because they are provided separately. This means a larger attack surface.

Interoperability

Monolith	Modulith	SOA	Microservices
Poor	Medium	Good	Good

Services use open interfaces that facilitate connecting external services. The Modulith is better prepared for such connections because it can be equipped with the appropriate adapters more quickly than is the case with the Monolith.

Maturity

Monolith	Modulith	SOA	Microservices
Good	Good	Good	Good

When used correctly, all architectural patterns can ensure the proper functioning of the software system.

Modifiability

Monolith	Modulith	SOA	Microservices
Poor	Medium	Good	Good

Monoliths are difficult to modify because their internal structure is less likely to be decoupled and therefore changes to one part can affect other parts. Microservices, on the other hand, operate as independently as possible.

Modularity

Monolith	Modulith	SOA	Microservices
Poor	Good	Medium	Good

Microservices are the most modular systems possible. SOA depends on the amount of and size of services. Moduliths are built with modularity in mind.

Recoverability

Monolith	Modulith	SOA	Microservices
Poor	Medium	Good	Good

Monoliths only work as a whole, so errors are more likely to affect the entire system. Moduliths have a greater chance of looking at individual components separately than monoliths do. Service-oriented architectures are associated with a whole range of tools that can be used to restore the status of individual services as quickly as possible.

Testability

Monolith	Modulith	SOA	Microservices
Poor	Good	Medium	Medium

Moduliths are great to test because they can run as a whole but also divided in their modules. Services based architectures are sometimes hard to test as a whole and monoliths cannot be decomposed.

Time behavior

Monolith	Modulith	SOA	Microservices
Good	Medium	Poor	Poor

Performance is lost with each layer of abstraction. Service-based systems have many more abstraction layers than software systems running in the same process.

User error protection

Monolith	Modulith	SOA	Microservices
Not applicable	Not applicable	Not applicable	Not applicable

This quality attribute must be addressed by specific implementation details that are not part of the architecture pattern.

Evaluation is based on ISO 25010.

Decision

Accepted - 20.05.2019

The future Wizard will be implemented as a Modulith. This allows the best backwards compatibility and a smooth migration of the existing system. The actual advantages of service-oriented architectures and microservices cannot be exploited in the case of the deployment scenario.

Consequences

An infrastructure must be created in order to be able to implement the Wizard as a modulith. A migration strategy is then required to break the Wizard down into modules and integrate them into the framework application in a loosely coupled manner.

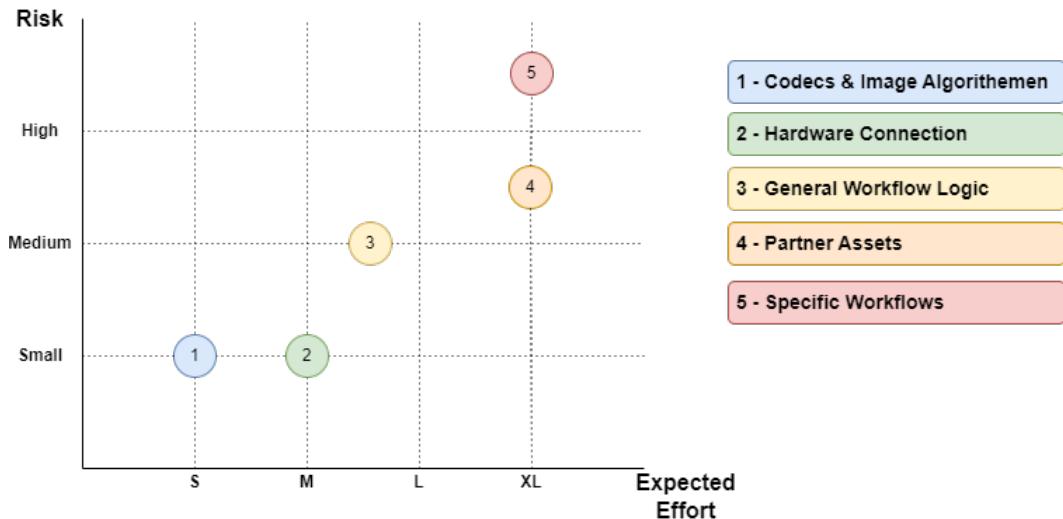
ADR 003 - Strategic decomposition of the wizard

This chapter is meant to show a more complex ADR as an example on how these can be used to justify design decisions and strategies.

Context

Part of the further development of the Wizard is its decomposition into workflow modules and service modules. This document describes which grouping and priority

should be taken into account in the decomposition. This is being coordinated with various stakeholders. It should also be noted that this decomposition is currently still very rough and is primarily intended as a preparation for creating a minimally modular application. In a further restructuring step, a functional decomposition can be performed on the basis of the target architecture.



Designation	Description	Efforts	Risks
Codecs & Image Algorithms	Codecs and image algorithms are used to read photos of various types and to prepare them for printing. Furthermore, they are used to apply filters or alienation effects to image contents.	The effort for extraction is comparatively low, since algorithms and codecs are used primarily, but do not have any external dependencies themselves.	The risk of extraction is comparatively low, since the use of the algorithms can be traced very well in the code, and the interfaces are very clear.

Designation	Description	Efforts	Risks
Hardware connection	This contains all driver information and dialogs that technicians come into contact with during commissioning and/or maintenance of sales units.	The hardware connection area is manageable. However, the dialogs are very rigid and not prepared for use in environments like the frame application.	Since dialogs are primarily seen by technicians, the risk is comparatively low. However, it can happen that technicians are hindered in their work in case of errors.
Workflow Engine	This group describes all the logic that is used for the general implementation of dynamic workflows and navigation. It is not (!) about the actual workflows, but only about their infrastructure.	The code has to be rewritten as much as possible because such a procedure was not foreseen when the existing application was designed.	The biggest risks come from the fact that the workflow engine has to be freely configurable and must not have any dependencies to the concrete workflows at individual partners. Since it is a central control element, errors here may have an impact all the way to the end customer.

Designation	Description	Efforts	Risks
Partner Assets	Logos, Themes, Workflow configurations etc. for different partners	Due to the large number of partners and their respective customizations, the effort required to extract the code is very difficult to estimate. For some partners, special adaptations were made to the Sales Unit, which must also be reflected in a restructured version.	Due to the large amount of code, the many possible combinations, and the pervasiveness of the application (some assets are hardcoded), the risk of overlooking things that then become apparent as errors directly to the customer or partner is very high.
Special Workflows	Dialogs, workflows, product descriptions, etc., the use of which can be configured via the partner assets.	The same applies here as with the partner assets, only in an increased form. Here, the same applies as for the Partner Assets, only in increased form.	Here, the same applies as for the Partner Assets, only in increased form. The same applies here as for partner assets, only in increased form.

Decision

Accepted 27.11.2019

Due to the low effort and low risk of the conversion measures, it was decided to first outsource the codecs and the image algorithms to a separate service module. Although this does not serve the goal of a functional modularization of the wizard, it does allow the migration to be run through once with a low investment before more complex and riskier conversion measures are undertaken.

After the migration of the codecs and the image algorithms, the implementation of the general workflow logic as an independent module will be pushed forward. Again,

this does not serve the long-term goal of functional modularization, but it does allow such a central and important component to be decoupled and easily testable from the outset.

Based on the resulting structures, the areas of partner assets and specific workflows can then be implemented.

Consequences

While the initial modules will most likely not yet conform to the long-term architecture, they will be designed to be easily adaptable. Once the workflow logic has been implemented, further analysis of the legacy code and more detailed planning of the migration will need to take place in consultation with the various stakeholders.

ADR 004 - Application framework

This chapter shows how decision matrices can be used to formalize the comparison of different solution alternatives.

Context

The basic architecture requires an application framework with which modular applications can be created. As part of this, the following evaluation was carried out in October 2020. This comparison was performed based on the best team's knowledge and belief by first establishing and ranking the evaluation criteria. Then the frameworks were analyzed by different people, presented to the team, and evaluated in terms of their suitability (Score). The procedure was deliberately designed in this way because it is a very important and long-term decision that should be made as objectively as possible.

Rank: * 3 – perfect * 2 – good * 1 – at least not bad * 0 – bad

The total score of an alternative is calculated by multiplying the score of each alternative by the prioritization of each criterion and then adding them up.

		PRISM		Catel	
		Score	WPF		
Documentation	Description	Rank	78	55	
	How much information is available?	3	3	excellently documented	partial (Stack overflow); many base classes; property bag (Undo/Redo available); One man show but fast reaction; huge community
	How well is it written? Is it up-to-date?				
Github Activity	How fast are issues handled?	1	2	1 week ago, Issues taken care of	1 day ago, Issues taken care of
	How many issues can be found?				
	How many people work on it?				
GitHub fame	How many people are interested in the framework?	1	2	4.2k stars, 1.2k forks	574 stars, 129 forks

		PRISM WPF		Catel	
View Handling	Is it possible to switch between views?	3	3	wire view/view-Model declaratively in XAML (DI is possible)	no special info found
Messaging	Does it have a message bus?	2	3	EventAggregator; messaging UI possible from background threads is possible	via static Message Mediator
Bootstrapping	Does it control how the application is started? Is the startup structured?	2	3	integral part	no, manual registration at IoC Container
Module Handling	Is it possible to create modules and load them dynamically?	2	3	yes, lots of different ways	it seems so (lack of documentation), it is not as good as Prism
UI Styling	Is it possible to create and manage themes?	1	0	not part of Prism	not part of Catel

	PRISM			Catel		
	WPF					
.NET 5 compatibility	3	3	yes	2	.NET Core 3.1	
Data Validation	Are there helpers to validate user input? Are there helpers to show validation errors?	2	1	not part of Prism but WPF makes it possible	3	yes
Costs (including learning costs)	Are there licence fees? How long might it take to learn?	2	2	moderate because excellent docu	2	huge learning cost because of lack in docu
Relevance	How often is it used? How many questions can be found on Stack Overflow? How easy is it to find information about?	2	2	950 downloads per day	0	not measurable

		PRISM WPF	Catel
Sustainability	How long does it exist? Are there signs that it will be closed down in the future?	3 3	yes 3 yes (2008)
Mockable for testing	Does it use IoC? Does central services have abstract interfaces?	3 3	yes 0 no (everything is static)
License	It must be an approved licence.	3 3	MIT 3 MIT

Decision

Accepted - 13.10.2020

The Prism framework was chosen as the application framework for the Wizard because it meets all the requirements. Static code analysis must be used to ensure that there are no dependencies on the framework outside of bootstrapping, module definition and the user interface. In these parts, dependencies can be tolerated, since there is a dependency on WPF here anyway. However, if a platform-independent technology is chosen in the future, then a new development of the entire wizard should not be necessary because of Prism!

Consequences

By choosing an application framework based on WPF, this technology will continue to be used for the wizard, which in turn means dependencies on Windows as the

operating system. At the same time, the migration of the existing application is easier because it has already been implemented with WPF.

VII.12 Risks & Technical Debt

ID	Risk	Description	Effects	Countermeasures
R1	Low test coverage in Wizard	<p>The Wizard was implemented over a longer period of time without unit tests, focusing on UI tests instead.</p> <p>As a result, the test coverage is below 10%.</p>	<p>There are occasional errors in the production environment. The effort of manual tests is very high.</p>	<p>Manual tests have been greatly increased and a fixed release process must be run before each release, in which the processes in the wizard are completely tested once. Developers are given time for test automation and refactoring. Test coverage is checked automatically and is not allowed to fall below a certain threshold for modified code.</p>

ID	Risk	Description	Effects	Countermeasures
R2	Porting the wizard to .NET 5 was not possible	June 2021 an attempt was made to port the wizard to .NET 5. This failed due to various reasons (see also Lessons Learned). For this reason, the wizard was initially only upgraded to .NET 4.8, which will not be compatible with newer .NET versions in the long run.	As time goes on, the Wizard will use obsolete technology that is no longer supported by Microsoft.	A new migration attempt has been planned for 2022. First the test coverage shall be increased, and unsupported components shall be replaced.
R3	WPF does not allow platform independence	WPF is used as the front-end technology of the Wizard. The entire architecture of the Wizard is geared towards the use of WPF and thus replacement is not easily possible.	WPF mandates the use of Windows. The licensing costs for Windows are high compared to Linux distributions. The WPF add-on libraries used in the software hinder the actual compatibility to newer .NET versions.	The incompatible libraries are replaced. Then the application is migrated to a newer .NET version. Then the need for platform independence is re-evaluated.

ID	Risk	Description	Effects	Countermeasures
R4	No remote access	Sales units cannot be accessed remotely even in the Connected version. Access is restricted to the extent that they can only transfer their data to the portal or receive minor software updates for the wizard. This decision was made deliberately to prevent attacks from the Internet.	Logging is limited to log files only and is therefore no longer up to date. Remote support is not possible and service technicians are always forced to make changes to the system on site. Updates of the systems are limited to the wizard but not to windows, which presents security risk.	An evaluation of alternative procedures has already begun. Based on this, the entire handling of the sales unit and the contractual arrangements with partners could change. It is therefore unlikely that the problem will be resolved in the short term.
R5	Global context for image exchange	All loaded images are kept in a static context class at runtime.	The logic to fill the context and empty it again after the end of a workflow is very complex. In the past, there were cases where customers saw other customers' images. This has been fixed but it must not happen again under any circumstances!	A restructuring of the image algorithms has top priority in the restructuring of the wizard.

VII.13 Glossary

Term	Description
Assets	Image files, color definitions, fonts, workflows, etc., through which the Sales Unit is adapted to the corporate identity of a pitch provider.
Pitch provider	A company that offers photo printing services via FotoMaX and provides a space for this purpose on their own premises.
Portal	The customer and partner portal of FotoMaX.
Print provider	Companies that carry out print jobs.
Sales Unit	Sales units consist of hardware components (screen, computer, etc.) and the Wizard software. If applicable, an external printer is also offered but does not count as part of the unit.
Wizard	Control software and front-end of a sales unit.
Workflow	The workflow or navigation sequence followed by the operation of the wizard. Typical workflow steps are: Reading images, selecting products, and placing an order.
...	...

VII.14 Organizational Topics

This chapter contains information that does not relate directly to the software architecture, but to the day-to-day collaboration in implementing the software.

Onboarding

Note that this chapter is a direct result of the quality requirements and scenarios regarding maintainability which states that new team members should be able to be productive in a certain amount of time.

This chapter describes resources to help new team members get the best possible start on the project.

Checklist for the start-up phase

The following checklist is intended to help new team members find their way around the team as quickly as possible and be able to work productively.

Check	Work Item	Contact Person	Reference Material
<input checked="" type="checkbox"/>	Basics of agile working	Scrum Master	Scrum Guide¹⁴⁴ , Agile Manifesto¹⁴⁵
<input checked="" type="checkbox"/>	Clarification of how the team has adapted the agile work	Scrum Master	Team Charter , Definition of Ready , Definition of Done
<input checked="" type="checkbox"/>	Invitation to regular meetings	Scrum Master	Daily, Review, Planning, Retro, Refinement
<input checked="" type="checkbox"/>	Invitation to agile tools	Scrum Master	Teamretro, Miro
<input checked="" type="checkbox"/>	Introduction to used agile tools	Scrum Master	scrum-poker.org¹⁴⁶ ,
<input checked="" type="checkbox"/>	Introduction to product	Product Owner	Teamretro, Miro
<input checked="" type="checkbox"/>	Introduction to architecture documentation	Architect	Marketing Documentation, User Manual, Test System Architecture Documentation

¹⁴⁴<https://www.scrum.org/resources/scrum-guide>

¹⁴⁵<https://agilemanifesto.org/>

¹⁴⁶scrum-poker.org

Check	Work Item	Contact Person	Reference Material
<input checked="" type="checkbox"/>	Read and understand architecture documentation	New Member	Architecture Documentation
<input checked="" type="checkbox"/>	Set up development environment	New Member	Installation Guide
<input checked="" type="checkbox"/>	Overview of the development process	Developer	Coding Guidelines, Review Guidelines
<input checked="" type="checkbox"/>	Overview of test procedures and tools used	Developer	Testing Guidelines, Testing Frameworks, Test isolation Frameworks
<input checked="" type="checkbox"/>	Read past sprint reviews	New Member	Sprint Reviews
<input checked="" type="checkbox"/>	Implement Tutorial	New Member	Tutorial
<input checked="" type="checkbox"/>	Checkout Repository	New Member	Git Repository
<input checked="" type="checkbox"/>	Write first test in production code	New Member	Source Code

Installation Guide

In a real documentation, this chapter would describe of how to set up a new development environment. Among other things, the following questions should be answered: * Which software is to be installed in which order? * Which environment variables or similar have to be set? * From where can the source code be obtained?

Tutorial

In a real documentation, a description would be found in this chapter, with which new team members can develop a first feature. This helps them to understand the basic architecture and the tools used, but also to test their development environment before they write their first real production code.

Guidelines

Coding Guidelines

This chapter would contain descriptions of how the code should be styled, if this was a real documentation. In general, the coding guidelines should mainly deal with special cases instead of explaining the code style in detail. The latter can be better checked by static code analysis tools and/or a linters. Things to consider are for example:

- * Mention of the code style used with a link to additional sources and incl. explanation of how it is checked.
- * Dealing with exceptions (if supported by the language)
- * Dealing with enumerations (if supported by the language)
- * Common (Anti-)Patterns to avoid

Testing Guidelines

This chapter should contain descriptions on how automated or manual tests are to be performed. It is worthwhile at this point to formulate guidelines individually for each type of test in order to be able to design them in a way that is appropriate for the target group.

- * Unit Test Guidelines
- * Integration Test Guidelines
- * Exploratory Test Guidelines
- * ...

Review Guidelines

Code reviews are a very important means of quality assurance. This document is intended to provide reviewers with a guideline to decide whether or not a pull request may be included in the main branch. This guideline only includes things that are not already checked by an automated check through build tools.

Precondition of a review

Check	Description
<input checked="" type="checkbox"/>	The review is performed by a person who was not involved in the programming himself.
<input checked="" type="checkbox"/>	The code was checked out locally by the reviewer.
<input checked="" type="checkbox"/>	The code was built locally.
<input checked="" type="checkbox"/>	Automated tests were executed locally.

Criteria

Check	Description
<input checked="" type="checkbox"/>	The software was built successfully.
<input checked="" type="checkbox"/>	The software can be started locally and simple operations in the area of the change are possible without complications.
<input checked="" type="checkbox"/>	No exceptions were detected during the execution of the code (see Exception Information of the debugger).
<input checked="" type="checkbox"/>	All automated tests can be executed successfully.
<input checked="" type="checkbox"/>	There are no ignored tests.
<input checked="" type="checkbox"/>	If tests were deleted, this was done justifiably or they were replaced by others.
<input checked="" type="checkbox"/>	There are no warnings in the local development environment.
<input checked="" type="checkbox"/>	There is no code that suppresses warnings in the code analysis tools.
<input checked="" type="checkbox"/>	The code satisfies the rules from the Coding Guidelines .
<input checked="" type="checkbox"/>	The tests satisfy the rules from the Testing Guidelines .

Daily Work

This part of the documentation is reserved for all topics regarding the organization of the daily work of the team.

This area would be separately handled for each and every team. Following only one example can be found to hold the overall documentation short.

Team Charter

The team charter is a set of rules created by each team itself to define how it wants to work together. It should only contain things that all members can agree on. Examples:

- We are punctual.
- Problems are resolved promptly and factually.
- Documentation and testing are as important as source code.
- We maintain a positive feedback culture.
- Cell phones may not be used during an appointment.

Definition of Ready

With the Definition of Ready, the team determines which prerequisites must be met before the actual development can start. The DoR was created by the team and each team member agreed on it.

- The requirements are described as user stories and placed in the backlog.
- The user story has been assigned a business value.
- The user story is understood by all team members.
- Testable acceptance criteria are available.
- External dependencies have been named.
- The user story has been estimated in story points.

Definition of Done

With the Definition of Done, the team determines when it is finally finished with the work on a feature. The DoD was created by the team and each team member agreed on it.

- A code review was performed by a developer who was not involved in the implementation.
- All acceptance criteria have been implemented.
- Test coverage is at least 75%.
- All tests are successful.
- The code has been transferred to the Main Branch.
- There was a formal review by the product owner.

Sprint Reviews

Results or resources from and for the sprint reviews can be stored in this area. In this way, the information is not lost and can contribute to continuous improvement.

Vacation Planning

If there is no better solution, this area of the documentation can be used to track the availability of team members and plan vacations. This allows a more effective sprint planning.

VIII - Mac-OS Menubar Application

By Gernot Starke.

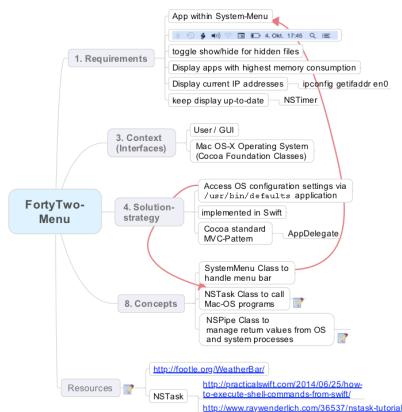
This surely is one of the shortest and most compact architecture documentations.

Its goal is to show that arc42 can be *extremely* tailored and reduced to keep only a minimal amount of documentation.

I implemented this example as a brief excursion into the Swift programming language. I did not want to invest in documentation, though I wanted to persist the most important facts about this implementation.

The result: A few keywords on requirements, names of some important OS classes, a few links to resources I found useful during implementation.

I used a mindmap instead of text, tables and diagrams.



Extremely Short Documentation

The Authors

Gernot Starke

Dr. Gernot Starke ([innoQ¹⁴⁷](https://innoq.com) Fellow) is co-founder and longstanding user of the (open source) [arc42¹⁴⁸](https://arc42.org) documentation template. For more than 20 years he works as software architect, coach and consultant, conquering the challenges of creating effective software architectures for clients from various industries.



In 2008 Gernot co-founded the International Software Architecture Qualification Board ([iSAQB e.V.¹⁴⁹](https://isaqb.org)) and since then supports it as an active member.

2014 he founded the (open source) Architecture Improvement Method [aim42¹⁵⁰](https://aim42.org).

Gernot has authored several (German) books on software architecture and related topics.



Gernot studied computer science at the Institute of Technology in Aachen (RWTH Aachen) and finished with a Diploma. He then worked as a developer and consultant for smaller software companies, before coming back to university for international research on methodical software engineering. 1995 he received his PhD from Johannes Kepler University of Linz, Austria (Prof. Gerhard Chroust for his thesis on “Software Process Modeling”).

He then joined Schumann AG in Cologne and did consulting and development work for several years. He became technical director of the “Object Reality Center”, a joint-venture of Sun Microsystems and Schumann Consulting

¹⁴⁷<https://innoq.com>

¹⁴⁸<https://arc42.org>

¹⁴⁹<https://isaqb.org>

¹⁵⁰<https://aim42.org>

AG and lead the first European Java Project (the Janatol project for Hypobank in Munich).

Since then he has consulted and coached numerous clients from various domains, mainly finance, insurance, telecommunication, logistics, automotive and industry on topics around software engineering, software development and development process organization.

Gernot was an early adopter of the agile movement and has successfully worked as Scrum master in agile projects.

He lives in Cologne with his wife (*Cheffe Uli*) and his two (nearly grown-up) kids, two cats and a few Macs.

Email Gernot Starke¹⁵¹ or contact him via Twitter @gernotstarke¹⁵².

¹⁵¹<mailto:gernot.starke@innoq.com?subject=arc42%20By%20Example>

¹⁵²<https://twitter.com/gernotstarke>

Hendrik Lösch

Hendrik wrote his first real program in QBasic in 1998 when he was 14 years old. From there on he started to work with Html, created several websites and during his studies he entered a world where he wouldn't have much to do with websites for a long time. We are talking about cyber-physical systems, where hardware *and* software play an important role.

He draws his greatest motivation from the complex interactions of the software and hardware world as well as the challenge of transferring common software development practices to an environment where we developers are not only kept from our work by the end users and management, but also by "hardware guys". As a reward for his persistence, he occasionally gets to play with such great things as lasers, electron raster microscopes or industrial 3D printers at his employer [ZEISS¹⁵³](#). In this way, he has helped in the past to ensure that people can see better again, industrial plants produce less waste and microchips do not cost even more than they already do.



In addition to these activities as a software architect and developer, he conducts architecture reviews several times a year for a wide variety of clients both inside and outside the ZEISS Group. In this way, he has been able to gain profound insights into very different software systems and industries.

He prefers to share the knowledge he has gained, at conferences or in the form of his video trainings at LinkedIn Learning. Further information can be found on his website [hendrik-loesch.de¹⁵⁴](#).

¹⁵³<http://www.zeiss.com>

¹⁵⁴<http://www.hendrik-loesch.de>

Michael Simons

Michael Simons is a father of two, husband, geek, programmer and passionate cyclist.



Michael took his apprenticeship at the [FZ Jülich¹⁵⁵](#) and studies at FH Aachen: Campus Jülich. He is a PRINCE2 ® Registered Practitioner and sometimes torn between the roles of an architect and project manager. In 2018, he was announced a Java Champion.

Nowadays, Michael works as a Senior Software engineer at [Neo4j¹⁵⁶](#) in the Spring Data team. In a previous life he worked at ENERKO INFORMATIK, an Aachen based company dealing with GIS system and has a background focussed on geographic information systems for utilities and price calculation at the energy market. In his brief time at INNOQ he helped customers modernize their application systems. Michael is known for having a certain passion for SQL and Spring. You can buy Michaels book about modern software development with Spring Boot here: [here¹⁵⁷](#)



Michael is a dedicated blogger and engaged in various open source projects. You'll find his stuff starting at [michael-simons.eu¹⁵⁸](#).

You can reach Michael via [email¹⁵⁹](#) or on Twitter as [@rot-nroll666¹⁶⁰](#).

¹⁵⁵https://www.fz-juelich.de/portal/DE/Home/home_node.html

¹⁵⁶<https://neo4j.com>

¹⁵⁷<http://springbootbuch.de>

¹⁵⁸<http://michael-simons.eu>

¹⁵⁹<mailto:michael@simons.ac>

¹⁶⁰<https://twitter.com/rotnroll666>

Stefan Zörner

From the Bayer AG via IBM and oose to embark. Stefan Zörner¹⁶¹ has twenty years of experience in IT and still looks forward to the future with excitement. He supports clients in solving architecture and implementation problems. In interesting workshops he demonstrates how to use practical design tools as well as spreading enthusiasm for real life architectural work.



About embark



Hamburg-based embark¹⁶² is your first choice for architectural projects. We are an independent consulting company, working creatively to find the best solution for you. Our customers trust us because we actively shape their systems, technologies and platforms as needed. This meets the long-term goal to gain optimum value from your IT investments.

Our team of consultants are generally recognized experts, with hands-on experience from various international projects. Technical and professional expertise, combined with efficient approaches are the leading characteristics of our architectural and engineering work.

Document your software architecture decisions and principles effectively and thoroughly!

Do you want to build a strong foundation for architecture documentation across your organization? Do you want to use a standardized template like e.g. arc42?

Architecture documentation isn't what you might think of first: baroque diagrams, large concepts, etc. Start your architectural overview with only a few basic 'ingredients'. Our cheat sheet¹⁶³ gives you a quick and helpful introduction.

¹⁶¹<mailto:stefan.zoerner@embarc.de>

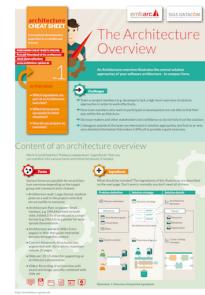
¹⁶²<https://www.embarc.de>

¹⁶³<https://www.embarc.de/architektur-spicker/>

We also support you with specific coaching, workshops or initial presentations for documentation topics. Key benefits for your team:

- A common understanding of architecture decisions / concepts / solutions.
- Appropriate communication to different target audiences.
- A proper basis for discussions / reflection / architecture reviews.
- Get new team members efficiently up to speed.
- Learn from our best practices from many other projects.

Please find more information on our [website¹⁶⁴](#).



¹⁶⁴<https://www.embarc.de/themen/dokumentation/>

Ralf D. Müller

More than twenty years of experience in web development have shaped Ralf D. Müller¹⁶⁵'s experience and thinking. And as trained Six Sigma Black Belt, the productivity of the whole is most important for him. That's why he dedicated much of his spare time to make the process to document a software project even easier. No question that the arc42 template is at the heart of this process. In addition, docToolchain¹⁶⁶ helps development and architecture teams to implement the Docs-as-Code¹⁶⁷ approach.



¹⁶⁵<mailto:ralf.d.mueller@docs-as-co.de>

¹⁶⁶<https://github.com/docToolchain/docToolchain>

¹⁶⁷<https://docs-as-co.de>

Contacting the Authors

In case you have questions or suggestions concerning the examples, arc42 or software architecture stuff in general, please let us know.

Gernot Starke

Just email Gernot Starke¹⁶⁸ or contact him via Twitter as [@gernotstarke](https://twitter.com/gernotstarke)¹⁶⁹.

Hendrik Lösch

Just email Hendrik Lösch¹⁷⁰ or contact him via his website hendrik-loesch.de¹⁷¹.

Michael Simons

Just email Michael Simons¹⁷² or contact him via Twitter as [@rotnroll666](https://twitter.com/rotnroll666)¹⁷³.

Ralf D. Müller

Just email Ralf D. Müller¹⁷⁴ or contact him via Twitter: [@RalfDMueller](https://twitter.com/RalfDMueller)¹⁷⁵.

Stefan Zörner

Just email Stefan Zörner¹⁷⁶ or contact him via Twitter: [@StefanZoerner](https://twitter.com/StefanZoerner)¹⁷⁷.

¹⁶⁸<mailto:gernot.starke@innoq.com?subject=arc42%20By%20Example>

¹⁶⁹<https://twitter.com/gernotstarke>

¹⁷⁰<mailto:mail@hendrik-loesch.de?subject=arc42%20By%20Example>

¹⁷¹<https://hendrik-loesch.de>

¹⁷²<mailto:misi@planet-punk.de?subject=arc42%20By%20Example>

¹⁷³<https://twitter.com/rotnroll666>

¹⁷⁴<mailto:ralf.d.mueller@docs-as-co.de>

¹⁷⁵<https://twitter.com/RalfDMueller>

¹⁷⁶<mailto:stefan.zoerner@embarc.de>

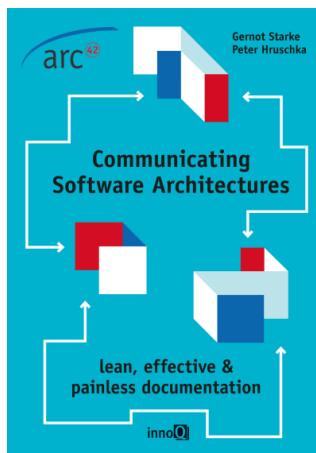
¹⁷⁷<https://twitter.com/StefanZoerner>

Further Reading

If you liked the way we explained, communicated and documented software architectures, you might want to know more about arc42.

Communicating Software Architectures

This practical guide shows how you can effectively apply the practical and well-proven arc42 template to design, develop and document your software architecture.



It contains more than 200 practical tips how to improve your architecture communication and documentation:

- immediately actionable tips
- arc42 for practical software development
- effective communication and documentation of software architectures
- arc42 to construct, design and implement new systems
- arc42 to document existing system
- arc42 for { large | medium | small } systems

- tools for arc42: wikis, asciidoc, modeling tools an others
- frequently asked questions around arc42

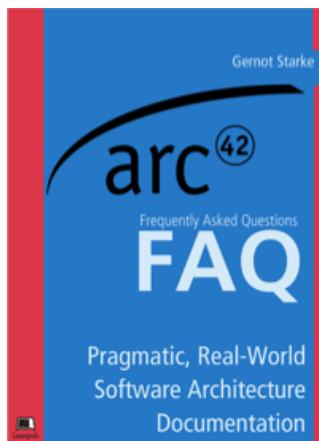
You find it on <https://leanpub.com/arc42inpractice>¹⁷⁸.

¹⁷⁸<https://leanpub.com/arc42inpractice>

arc42 FAQ (Frequently Asked Questions)

More than 150 questions around arc42 topics - available for free as eBook and online¹⁷⁹.

NEW: FAQ has now a searchable public (free!) website at <https://faq.arc42.org>¹⁸⁰



Contains questions in the following categories:

- General: Cost, license, contributing
- Methodology: minimal amount of documentation, where-does-what-info-be-long, notations, UML
- arc42 sections: quality requirements, context, building blocks, runtime scenarios, deployment, concepts etc.
- Modelling: UML, diagrams, interfaces, ports, understandability, consistency, clarity
- arc42 and Agility: Scrum, Kanban, definition-of-done, minimal, lean, economical documentation
- Tools for arc42
- Versioning and variants: versioning documents, variants of systems
- Traceability: tracing requirements to solution decisions and vice-versa

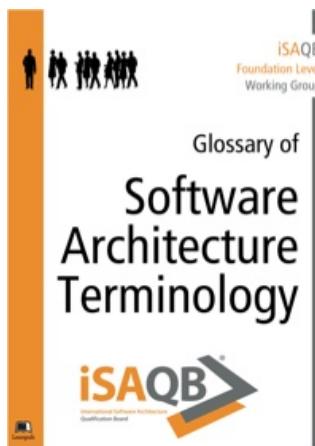
¹⁷⁹<https://leanpub.com/arc42-faq/read>

¹⁸⁰<https://faq.arc42.org>

- Management: very large systems, standardization, governance, checklists, access-rights
- Customizing arc42: tailoring and customizing, known adaptions of arc42

Glossary of Software Architecture Terminology

An extensive (and free!) collection of terms used in software architecture. Created and maintained by the iSAQB e.V., the association for software architecture education.



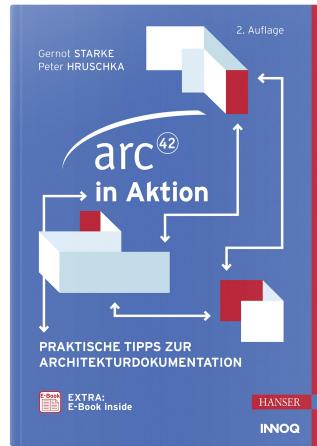
In addition to definitions this eBook also contains translation tables, currently for German and English.

You find it on <https://leanpub.com/isaqbglglossary>¹⁸¹.

¹⁸¹<https://leanpub.com/isaqbglglossary>

(German:) arc42 in Aktion

The German original of the previously mentioned eBook... for those who both understand German *and* like to hold books in their hands...



You get it at your local bookstore or [online¹⁸²](#).

¹⁸²<https://www.amazon.de/arc42-Aktion-Praktische-Tipps-Architekturdokumentation/dp/3446448012>