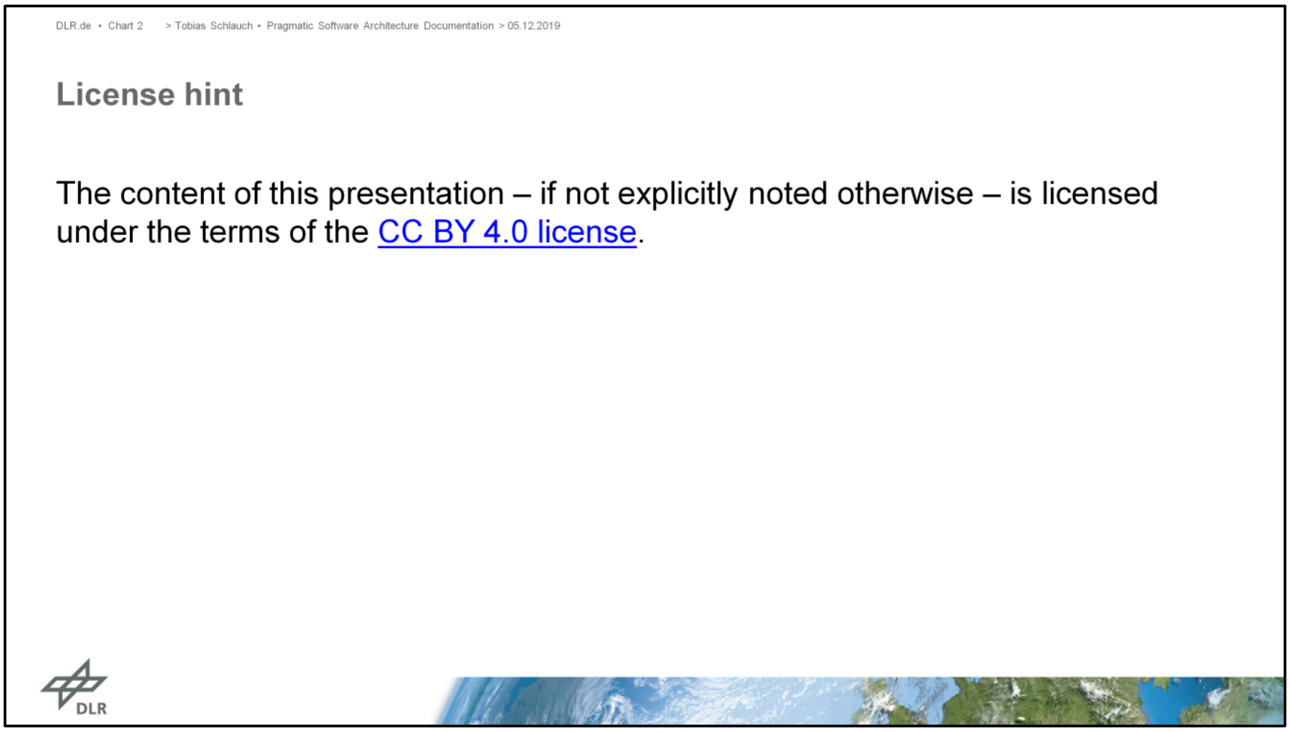
Pragmatic Software Architecture Documentation

Tobias Schlauch

German Aerospace Center (DLR)
Simulation and Software Technology

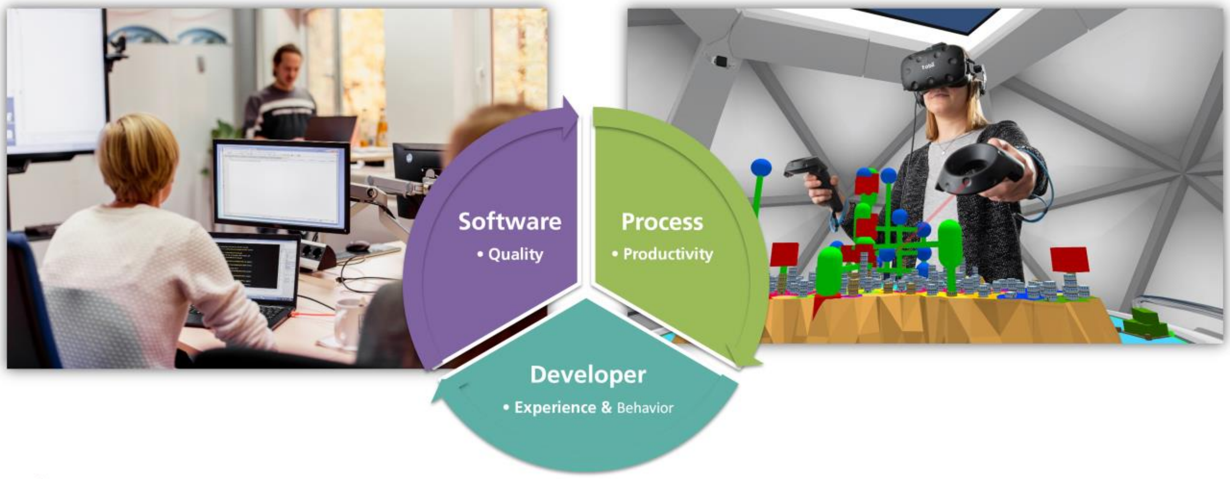
05.12.2019, Hochschule Hannover

Knowledge for Tomorrow



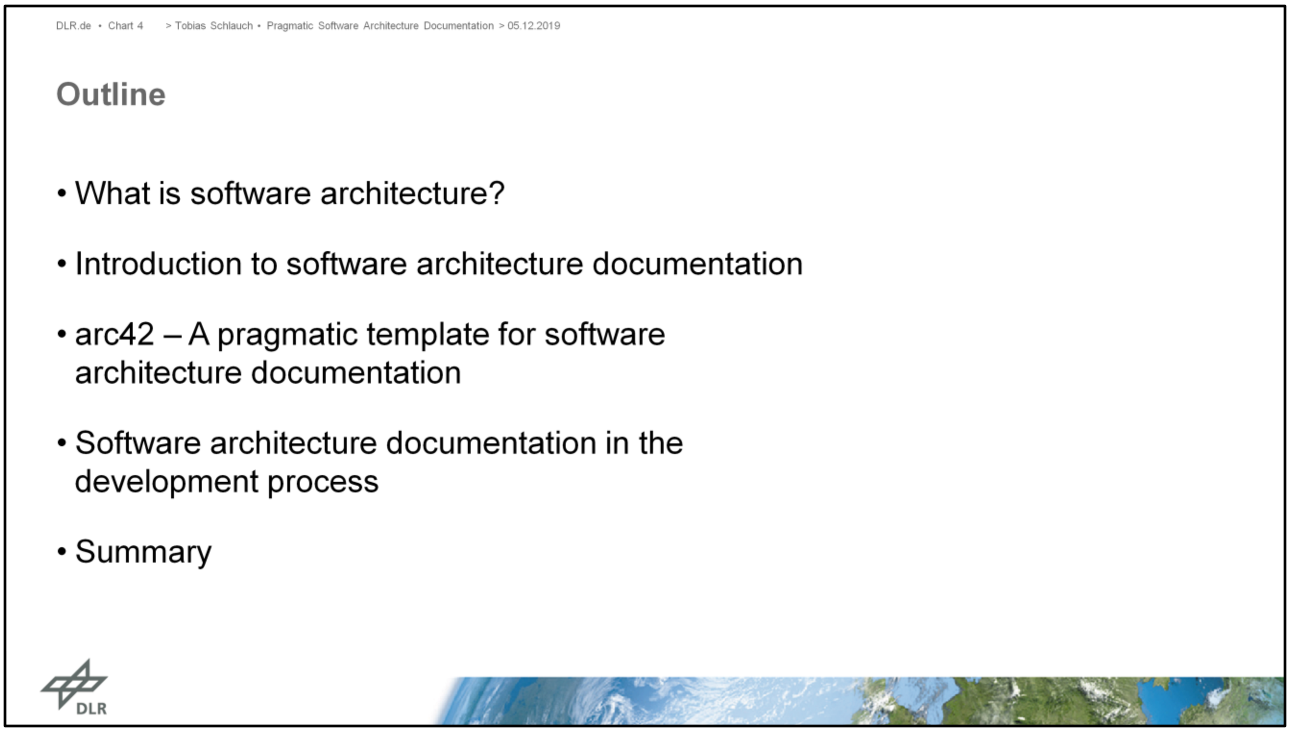
License hint

DLR Simulation and Software Technology
Group Software Engineering



DLR

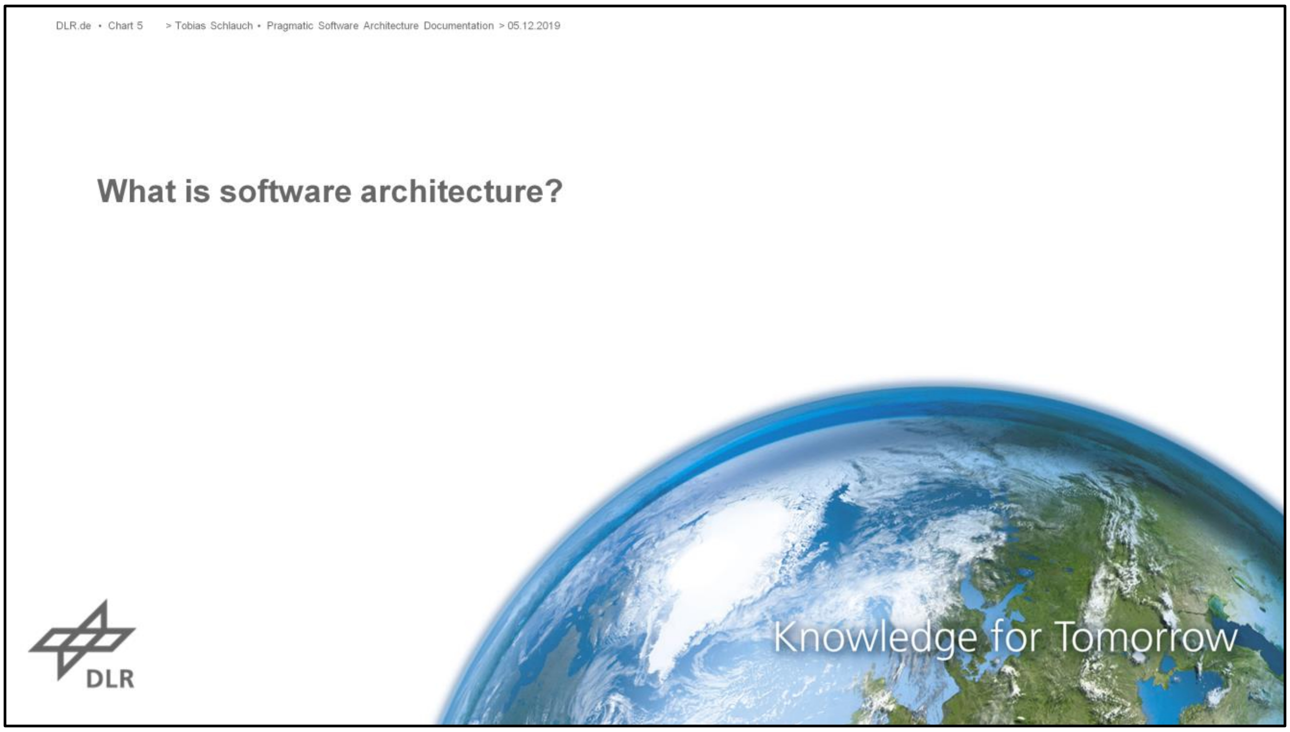
www.DLR.de/sc



Outline

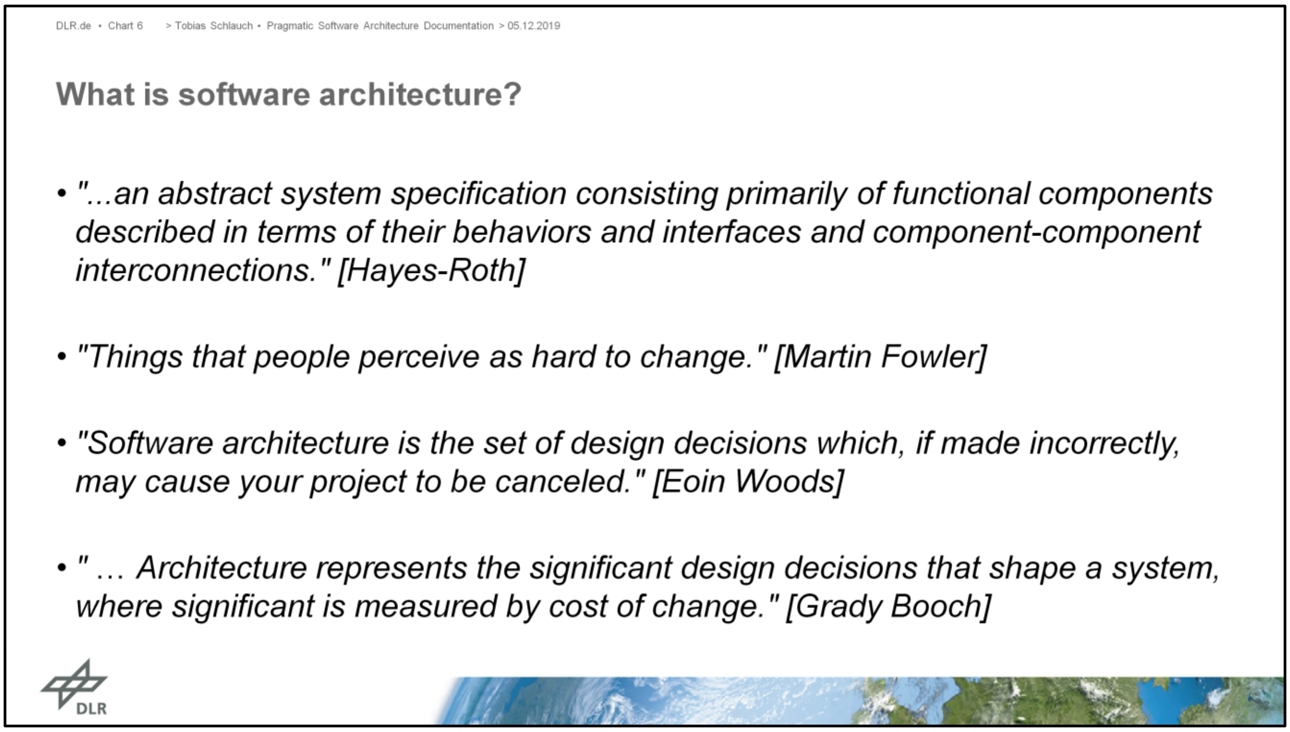
- What is software architecture?
- Introduction to software architecture documentation
- arc42 – A pragmatic template for software architecture documentation
- Software architecture documentation in the development process
- Summary

DLR



What is software architecture?

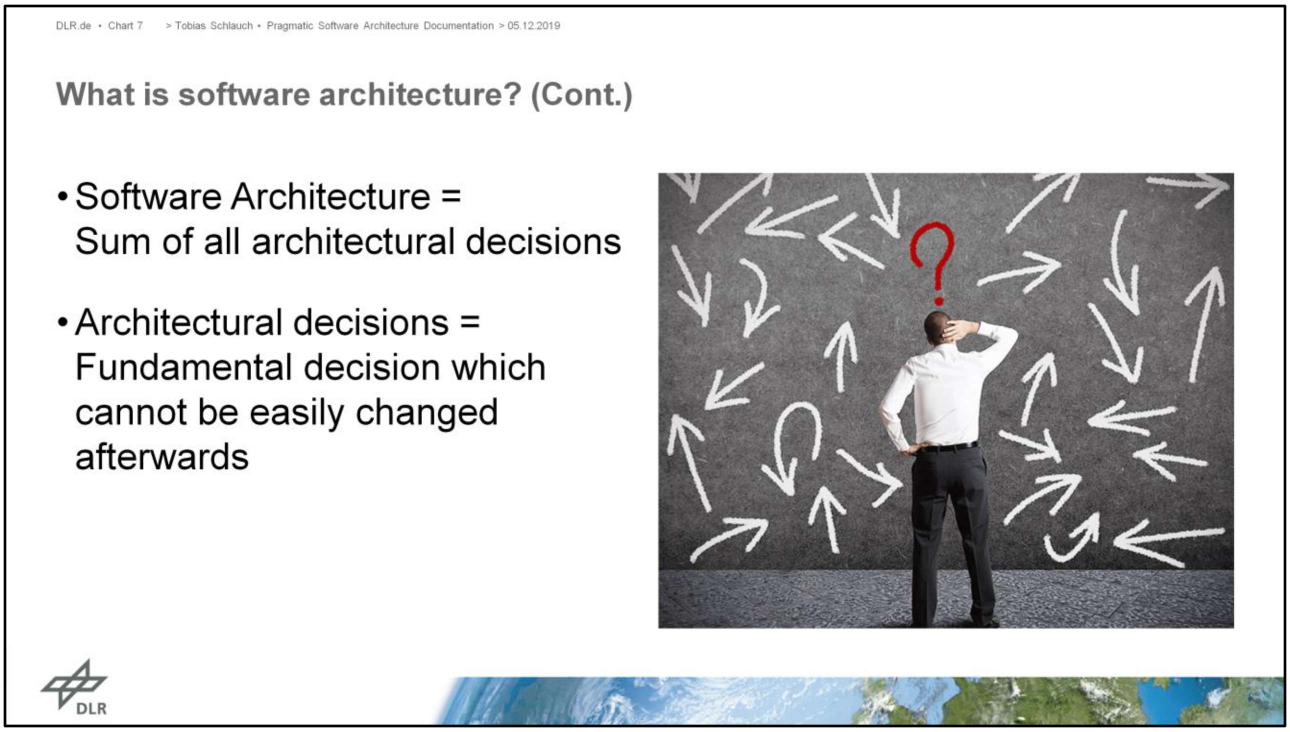
Knowledge for Tomorrow


DLR.de • Chart 6 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

What is software architecture?

- *"...an abstract system specification consisting primarily of functional components described in terms of their behaviors and interfaces and component-component interconnections." [Hayes-Roth]*
- *"Things that people perceive as hard to change." [Martin Fowler]*
- *"Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be canceled." [Eoin Woods]*
- *" … Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change." [Grady Booch]*

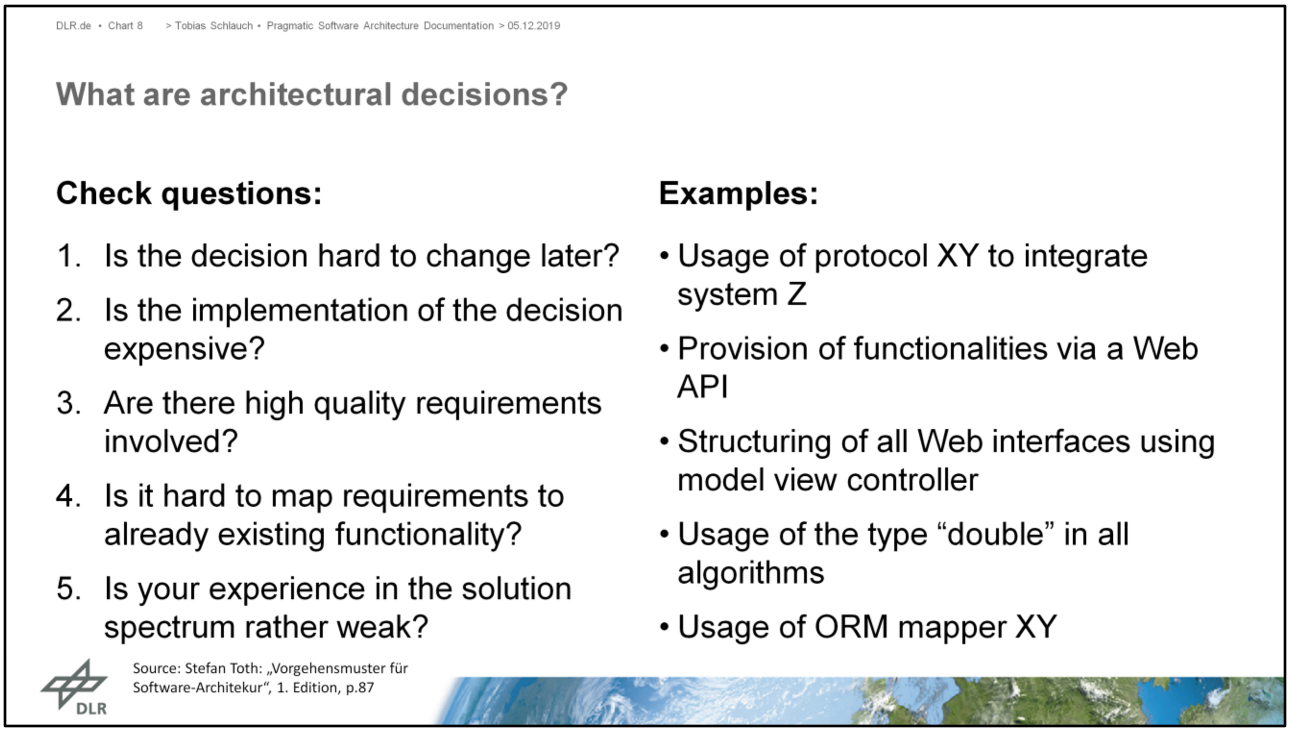
- There are many more definitions but there is no generally accepted one.
- But there are recurring themes such as system structure, hard to change things, decisions including their rationale



What is software architecture? (Cont.)

- Software Architecture =
  Sum of all architectural decisions
- Architectural decisions =
  Fundamental decision which cannot be easily changed afterwards

- But what are these architectural decisions? How can I find them?



What are architectural decisions?

Check questions:

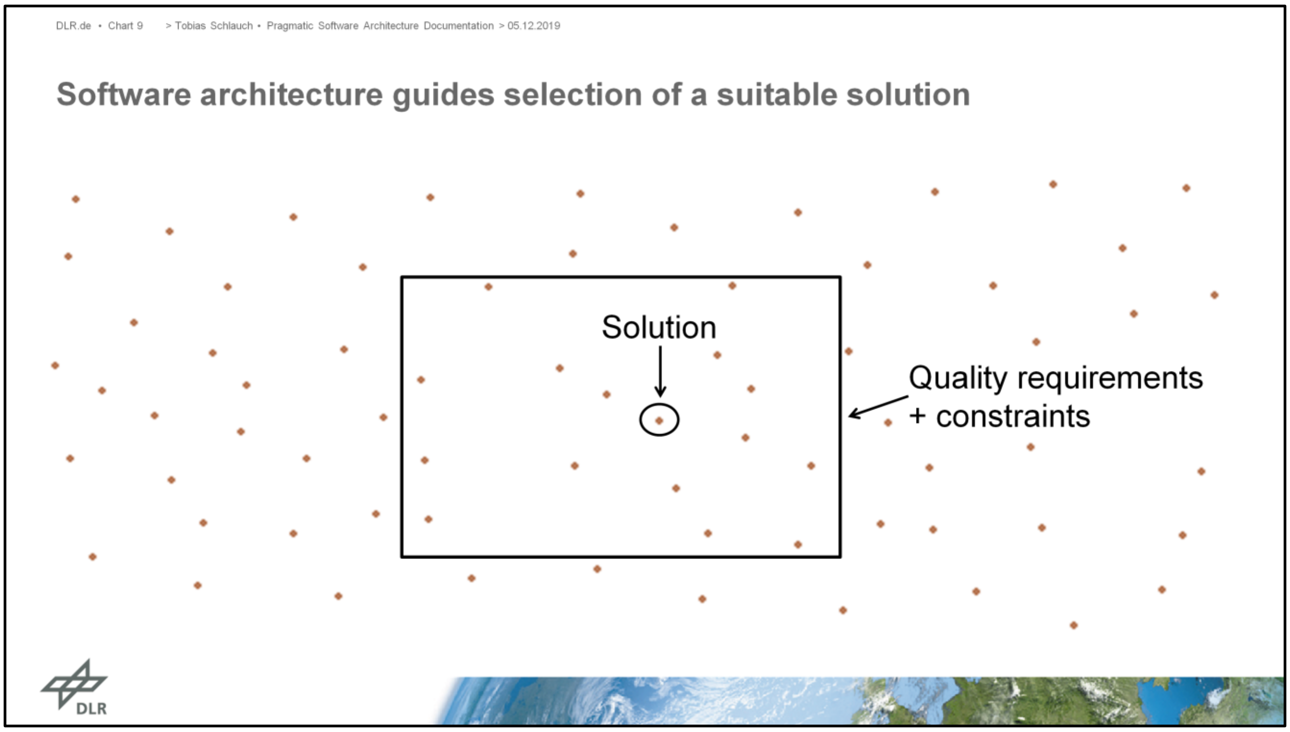
1. Is the decision hard to change later?
2. Is the implementation of the decision expensive?
3. Are there high quality requirements involved?
4. Is it hard to map requirements to already existing functionality?
5. Is your experience in the solution spectrum rather weak?

Examples:

- Usage of protocol XY to integrate system Z
- Provision of functionalities via a Web API
- Structuring of all Web interfaces using model view controller
- Usage of the type "double" in all algorithms
- Usage of ORM mapper XY

Source: Stefan Toth: „Vorgehensmuster für Software-Architekur", 1. Edition, p.87

- Control questions: a majority of „yes" answers indicates architectural significance
- Examples: It is **NOT** only about abstract things => its about important aspects including technology choices

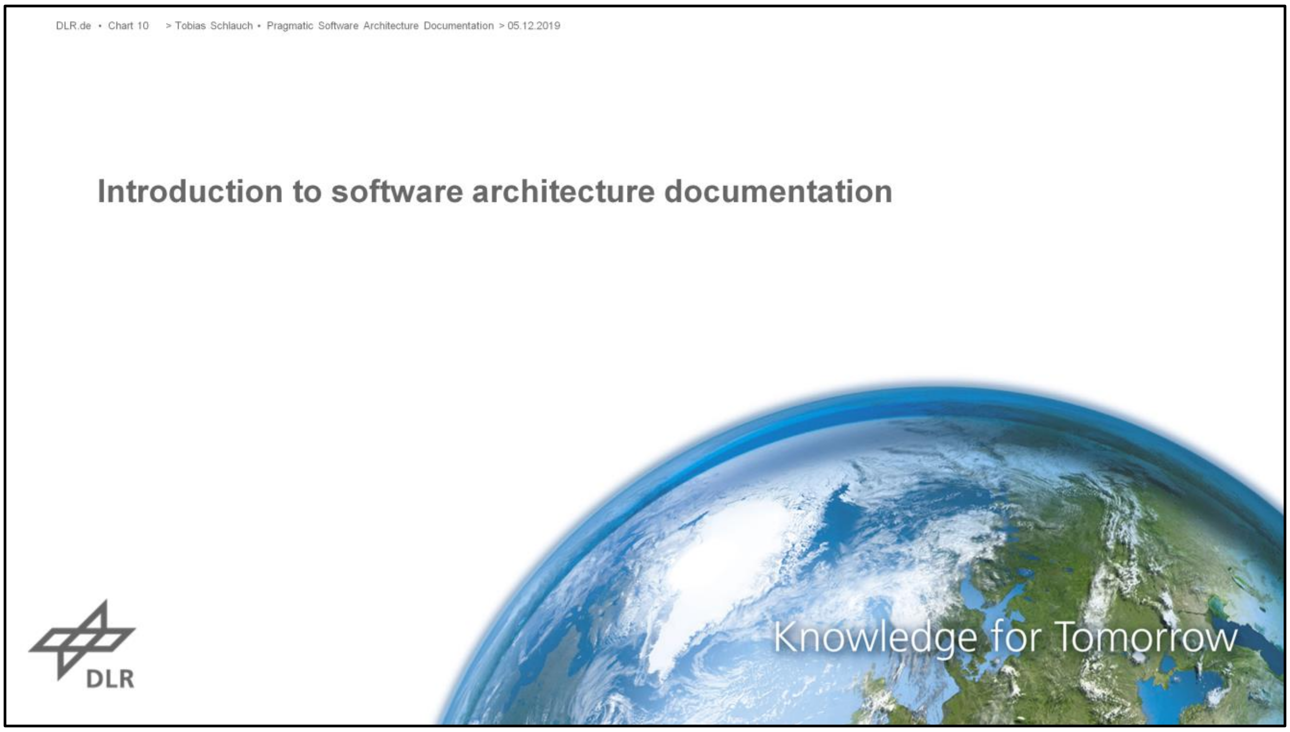


DLR.de • Chart 9 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

Software architecture guides selection of a suitable solution

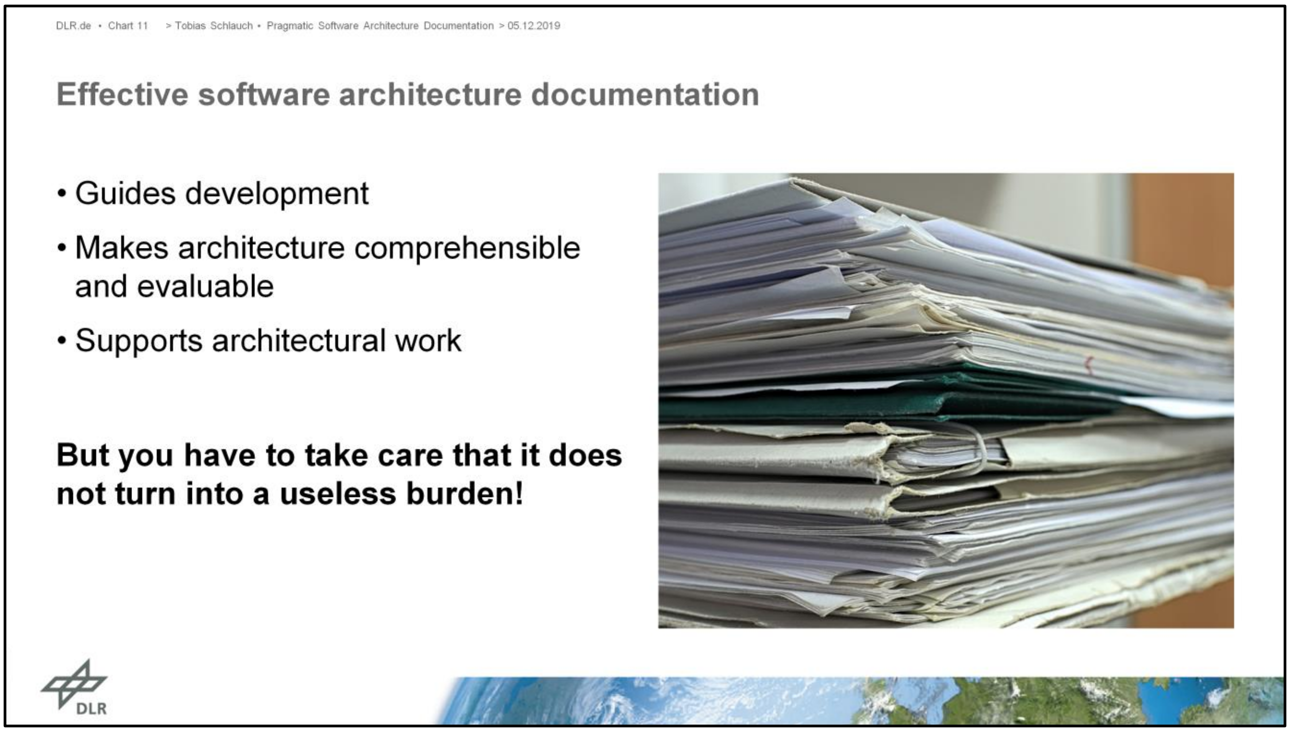
Solution
Quality requirements
+ constraints

- Software architecture – as a process/activity – guides the solution selection
- The decision making is typically heavily influenced by quality requirements and constraints => make sure you are aware of them!



Introduction to software architecture documentation

Knowledge for Tomorrow



Effective software architecture documentation

- Guides development
- Makes architecture comprehensible and evaluable
- Supports architectural work

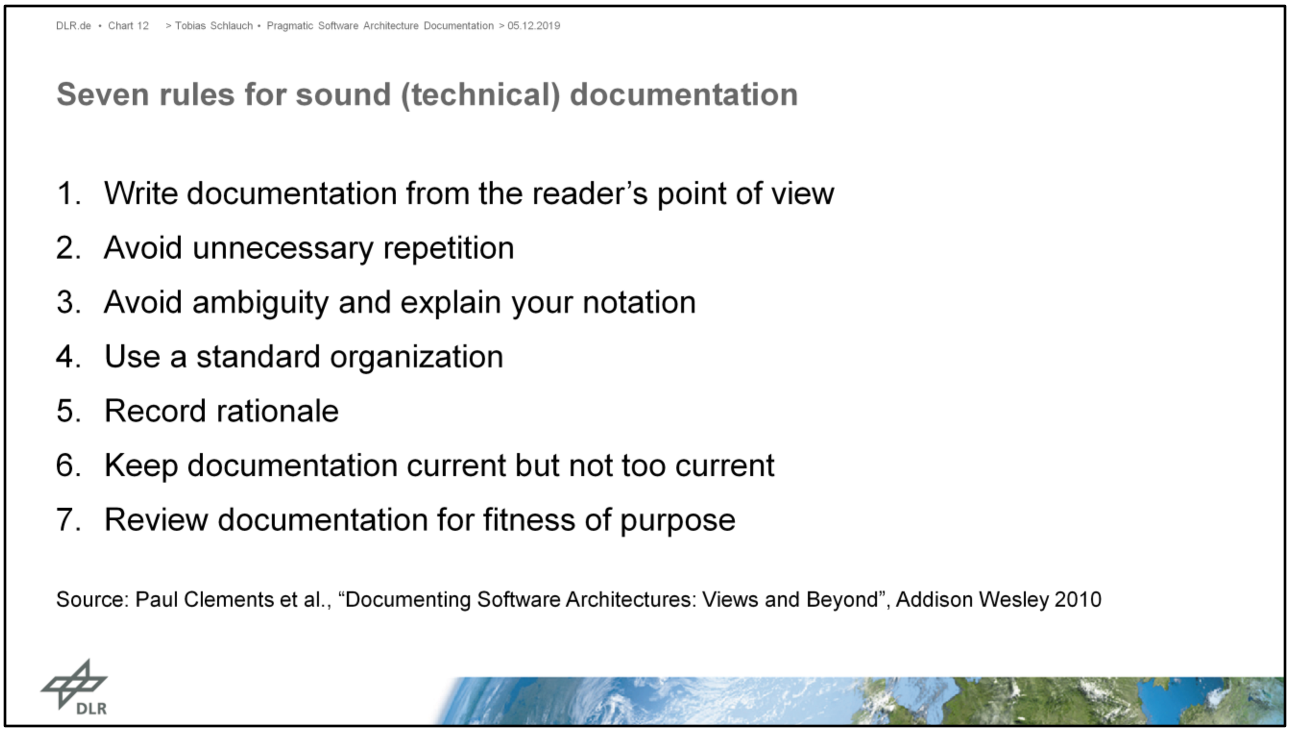
But you have to take care that it does not turn into a useless burden!

DLR

Good software architecture documentation:
- Provides hints to developers how to accomplish certain tasks
- Shows which decisions have been taken including their rationale.
- Provides sufficient information to help analyzing its suitability.

Good software architecture documentation is particularly value when:
- New team members join
- Important knowledge carrier leave the team
- Larger changes have to be carried out

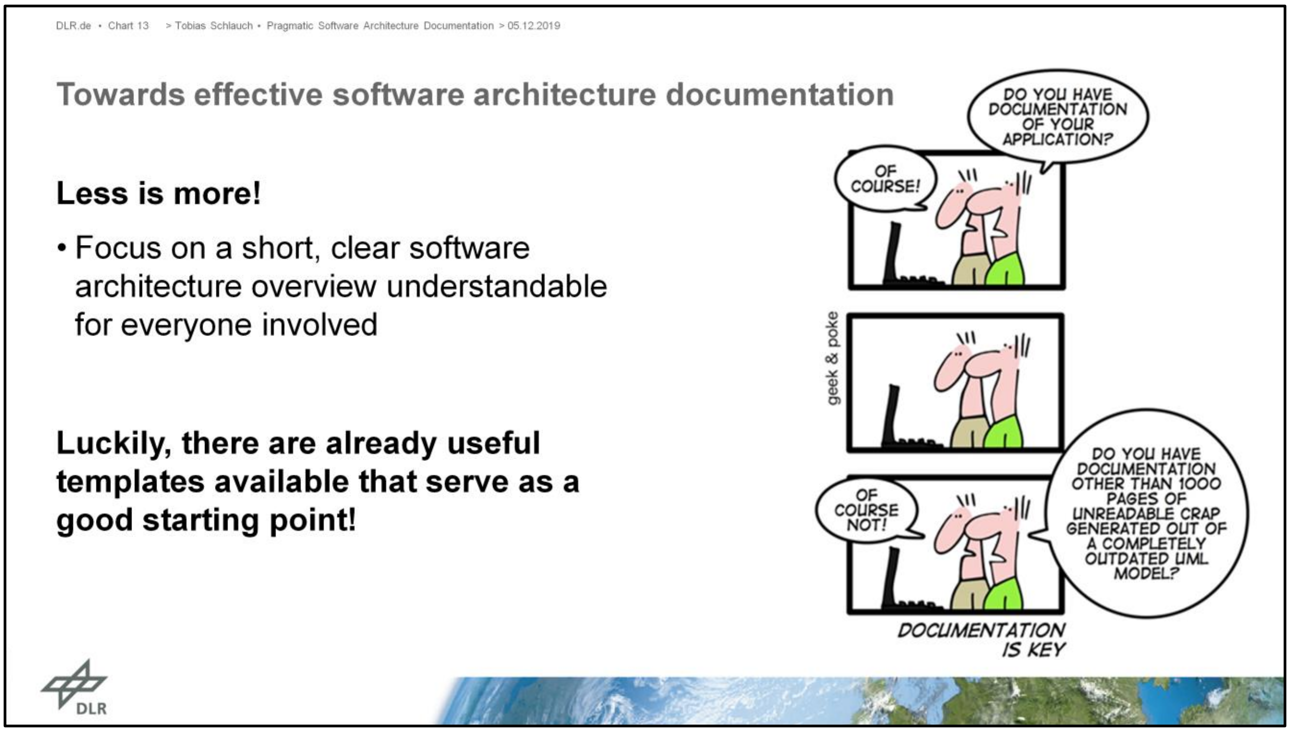
But typical problems include:
- Wrong, inconsistent, outdated or even useless information
- High creation and maintenance costs



Seven rules for sound (technical) documentation

1. Write documentation from the reader's point of view
2. Avoid unnecessary repetition
3. Avoid ambiguity and explain your notation
4. Use a standard organization
5. Record rationale
6. Keep documentation current but not too current
7. Review documentation for fitness of purpose

Source: Paul Clements et al., "Documenting Software Architectures: Views and Beyond", Addison Wesley 2010

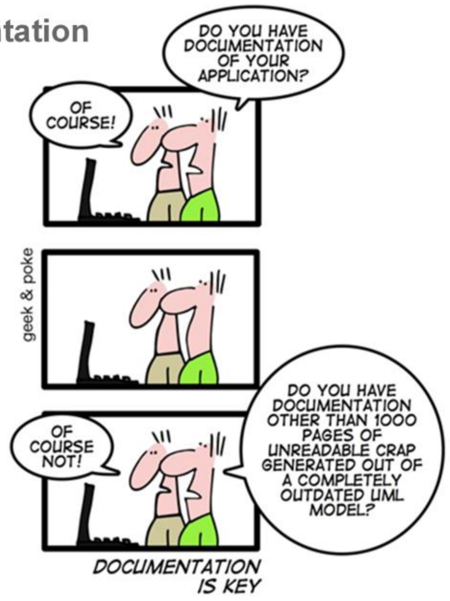


Towards effective software architecture documentation

Less is more!

- Focus on a short, clear software architecture overview understandable for everyone involved

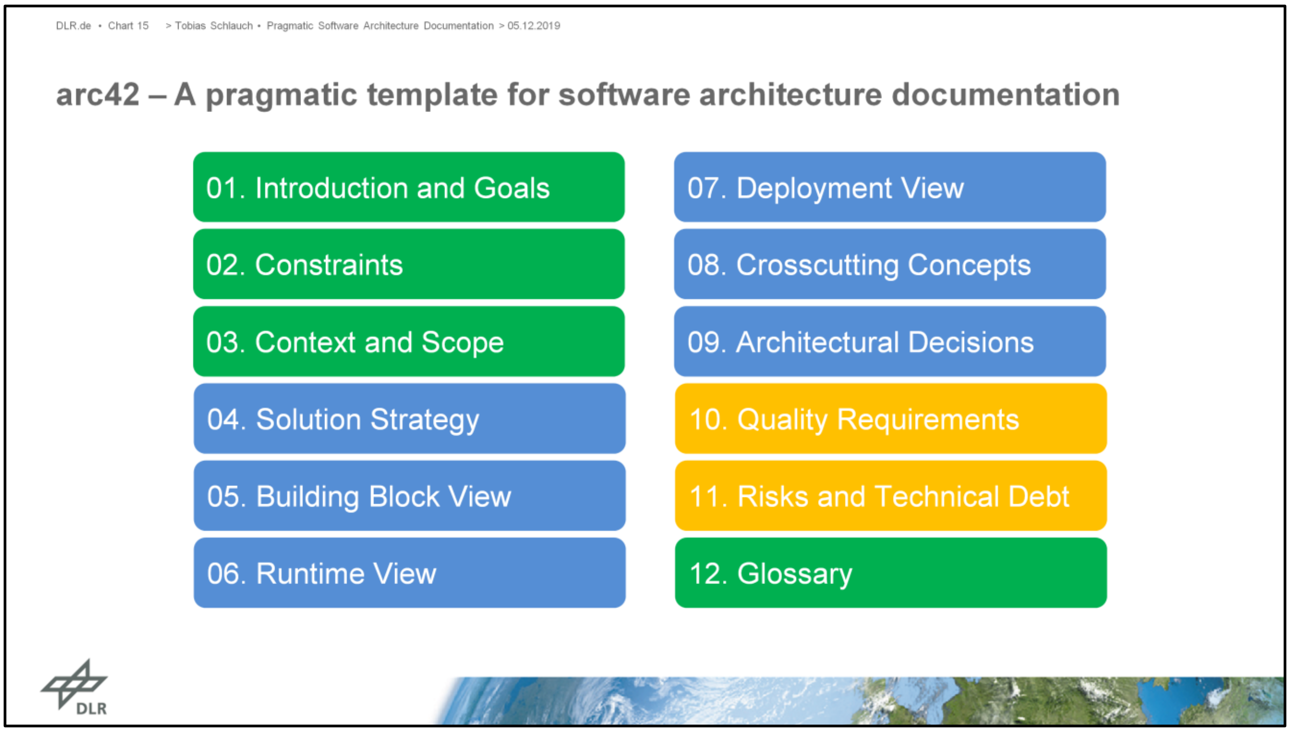
Luckily, there are already useful templates available that serve as a good starting point!


DO YOU HAVE DOCUMENTATION OF YOUR APPLICATION?
OF COURSE!
geek & poke
DO YOU HAVE DOCUMENTATION OTHER THAN 1000 PAGES OF UNREADABLE CRAP GENERATED OUT OF A COMPLETELY OUTDATED UML MODEL?
OF COURSE NOT!
DOCUMENTATION IS KEY

DLR

arc42 – A pragmatic template for software architecture documentation

Knowledge for Tomorrow

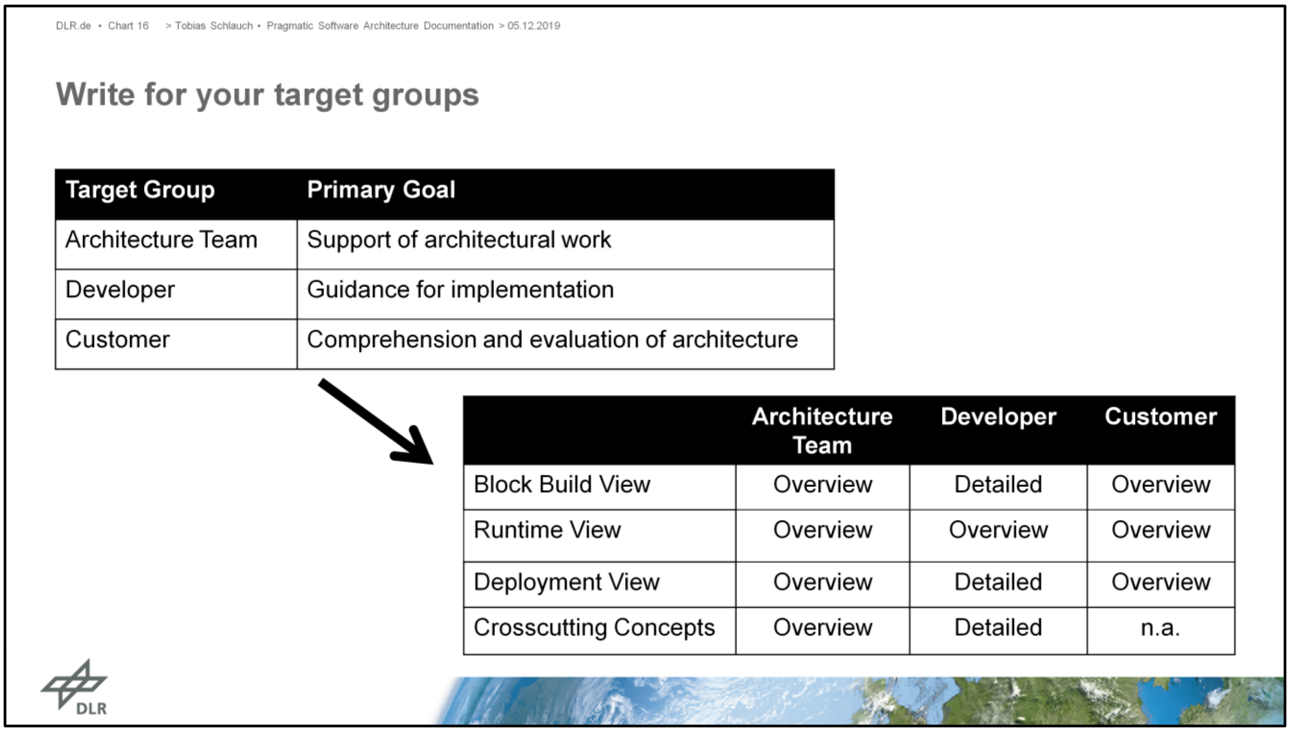


arc42 – A pragmatic template for software architecture documentation

01. Introduction and Goals	07. Deployment View
02. Constraints	08. Crosscutting Concepts
03. Context and Scope	09. Architectural Decisions
04. Solution Strategy	10. Quality Requirements
05. Building Block View	11. Risks and Technical Debt
06. Runtime View	12. Glossary

Overview:
- arc42: https://arc42.org
- Current version: 7.0
- Template for communication and documentation of software and system architectures
- Process- and tool-agnostic
- Made by practitioners
- Well documented, many examples and books are available
- Available in 4 different languages (DE, EN, ES, RU)
- Available in various formats: https://arc42.org/download
- Compatible with IEEE 1471
- Used in the iSAQB certification program
- License: CC BY-SA 4.0

Content structure / section meaning:
- For more information, please see: https://arc42.org/overview/
- Green => requirements-related sections
- Blue => solution-oriented sections
- Yellow => evaluation-oriented sections

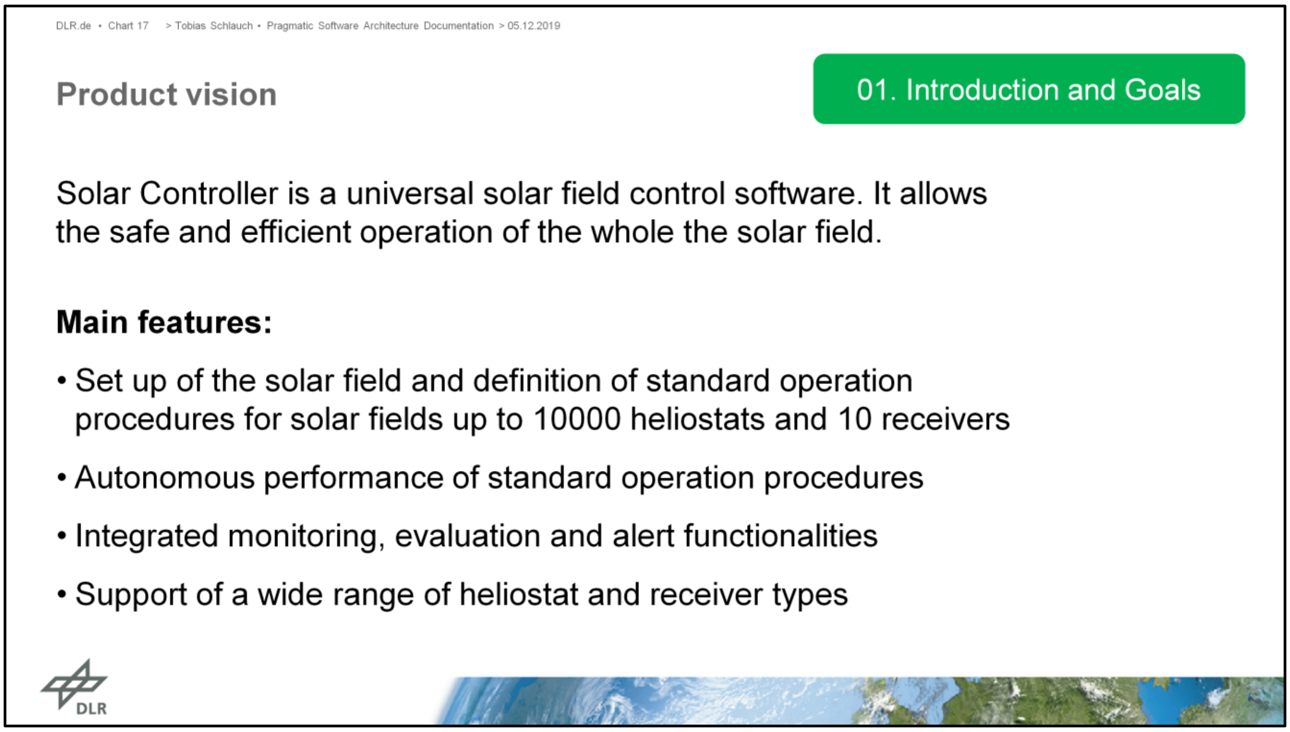


Write for your target groups

Target Group	Primary Goal
Architecture Team	Support of architectural work
Developer	Guidance for implementation
Customer	Comprehension and evaluation of architecture

	Architecture Team	Developer	Customer
Block Build View	Overview	Detailed	Overview
Runtime View	Overview	Overview	Overview
Deployment View	Overview	Detailed	Overview
Crosscutting Concepts	Overview	Detailed	n.a.

- The first table show exemplarily target groups and their goals with regard to the software architecture documentation.
- The second table depicts their potential interest in certain aspects of the software architecture documentation.



DLR.de • Chart 17 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

Product vision

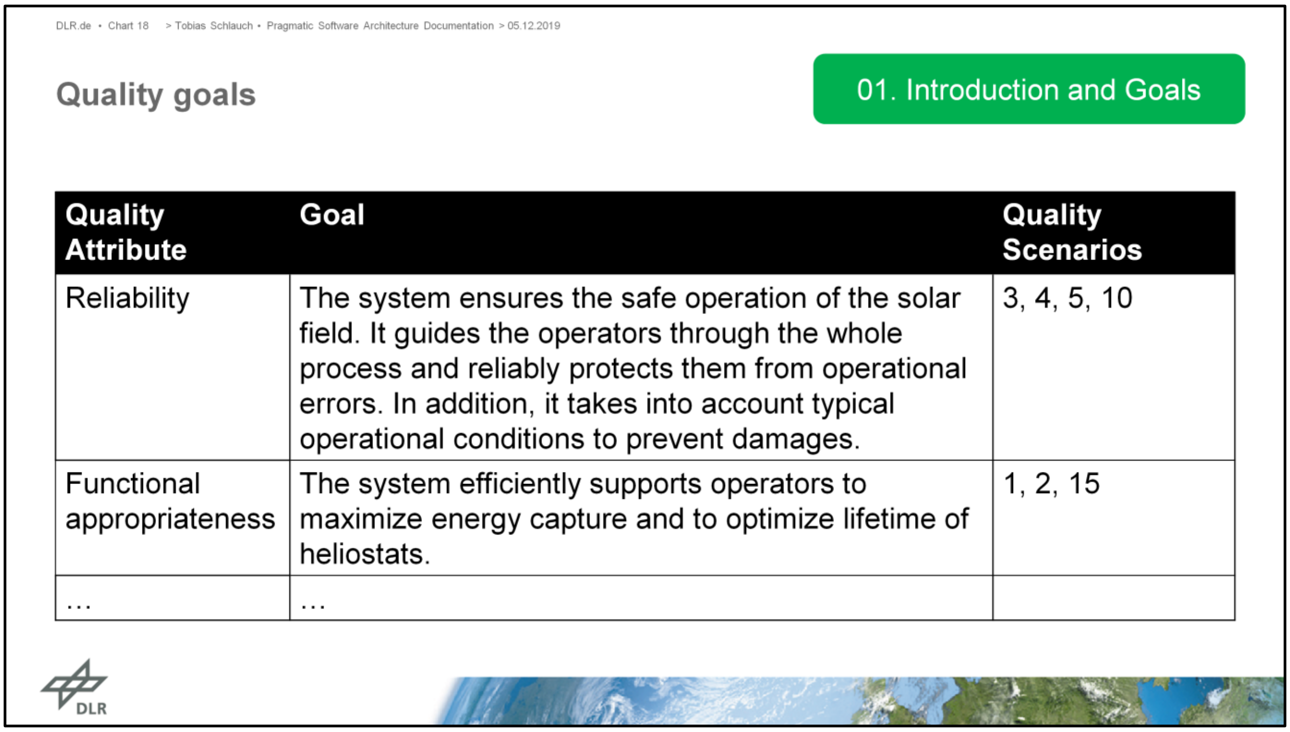
01. Introduction and Goals

Solar Controller is a universal solar field control software. It allows the safe and efficient operation of the whole the solar field.

Main features:

- Set up of the solar field and definition of standard operation procedures for solar fields up to 10000 heliostats and 10 receivers
- Autonomous performance of standard operation procedures
- Integrated monitoring, evaluation and alert functionalities
- Support of a wide range of heliostat and receiver types

- The product vision explains the purpose and the main features of the software.
- In general, it sums up the main selling arguments / reason for the software systems / why the system is built.
- In Open Source software, this information is typically shown in the README file.

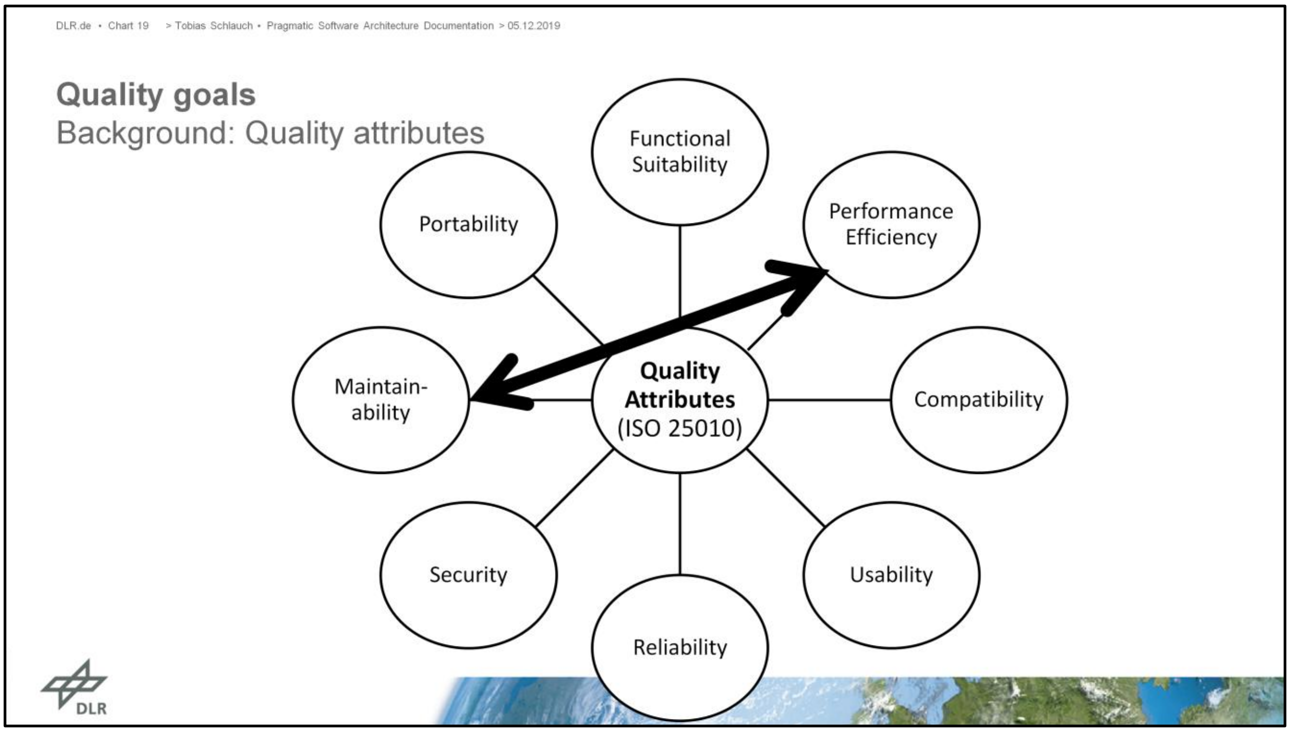


Quality goals

01. Introduction and Goals

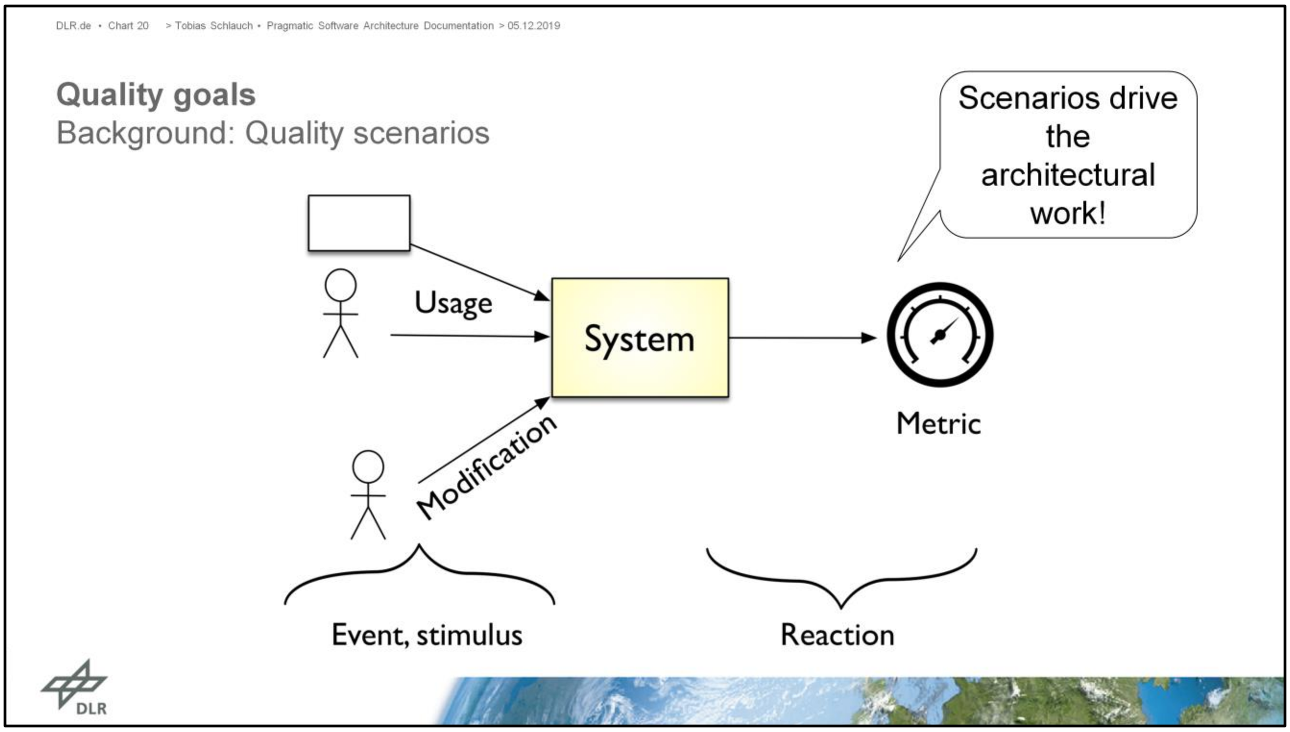
Quality Attribute	Goal	Quality Scenarios
Reliability	The system ensures the safe operation of the solar field. It guides the operators through the whole process and reliably protects them from operational errors. In addition, it takes into account typical operational conditions to prevent damages.	3, 4, 5, 10
Functional appropriateness	The system efficiently supports operators to maximize energy capture and to optimize lifetime of heliostats.	1, 2, 15
…	…	

- Software quality has to be explicitly constructed. Quality attributes offer a starting point but you need to interpret and prioritize them in context of your software. This is the purpose of quality goals.
- The tables shows how you can document the quality goals. You should reference the quality attribute and describe the goal in this regard. In addition, it is useful to reference relevant quality requirements (here captured as quality scenarios). These scenarios are more concrete and allow a better understanding of the actual requirements.
- The order of the goals implies their priority. The most important goal is defined first.
- You should limit the number of quality goals to 3 - 5.


DLR.de • Chart 19 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

Quality goals
Background: Quality attributes

- ISO/IEC 25010 defines the quality attributes. For further details, please see: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
- Beware that quality attributes influence each other. For that reason, it is important to prioritize them.

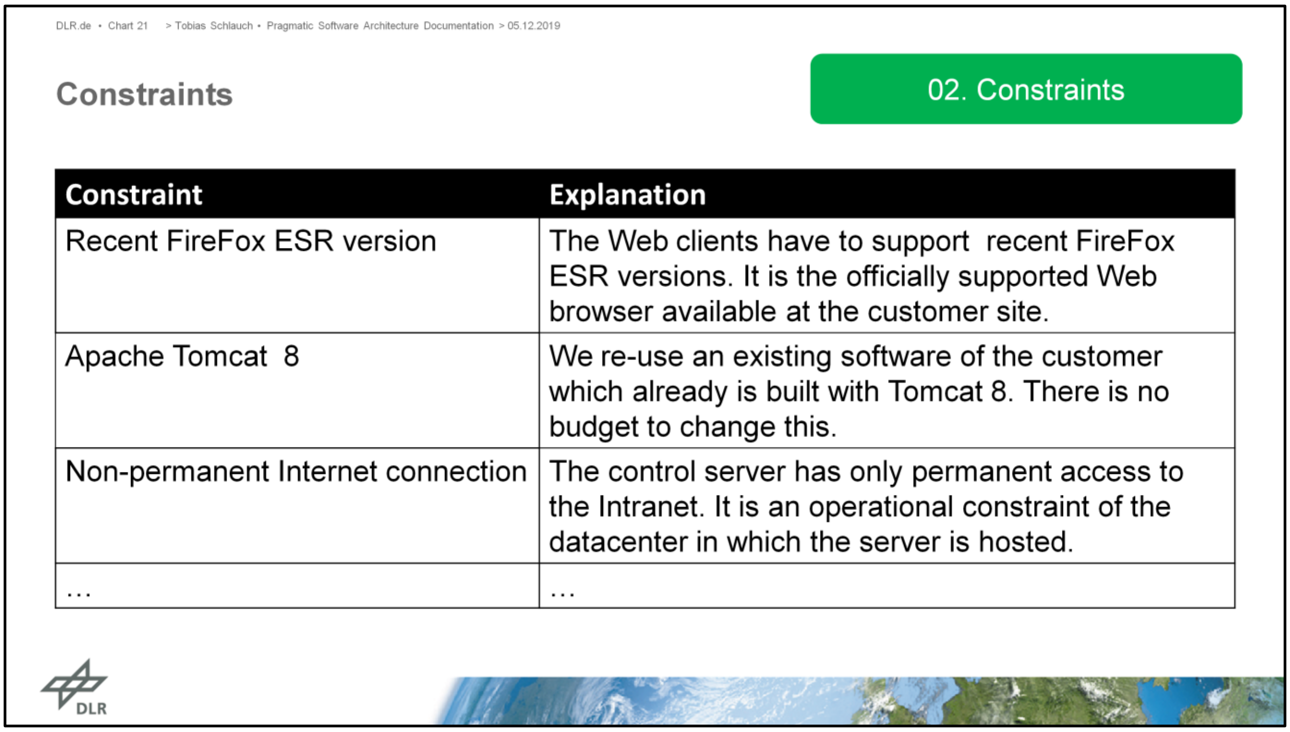

DLR.de • Chart 20 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

Quality goals
Background: Quality scenarios

- Quality scenarios describe the desired quality attributes in an informal, discussable way. They drive the architectural work and capture the quality requirements.
- Event/stimulus is the source of change.
- System is the (software) system which we actual want to describe.
- Reaction illustrates the response of our system to the incoming event/stimulus.
- Metric provides a measure for the response.

There different types of quality scenarios:
- Usage scenarios describe how the system is used and how it behaves in such situations. Example: “The customer uses the article search (event). The search (system) shows first matching articles (reaction) after one second (metric).”
- Change scenarios describe changes of the system`s environment or operational infrastructure. Example: “The number of users doubles (event). The system (system) does not show significant performance decreases for typical actions (reaction + metric).”
- Failure scenarios describe failure situations of neighbor systems or the system`s operational infrastructure. “Example: “A power plant indicates an overheat event (event). The control system (system) triggers the safe-mode immediately (metric).

- For more information, please see: https://faq.arc42.org/questions/C-10-2/ and https://github.com/arc42/quality-requirements



DLR.de • Chart 21 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

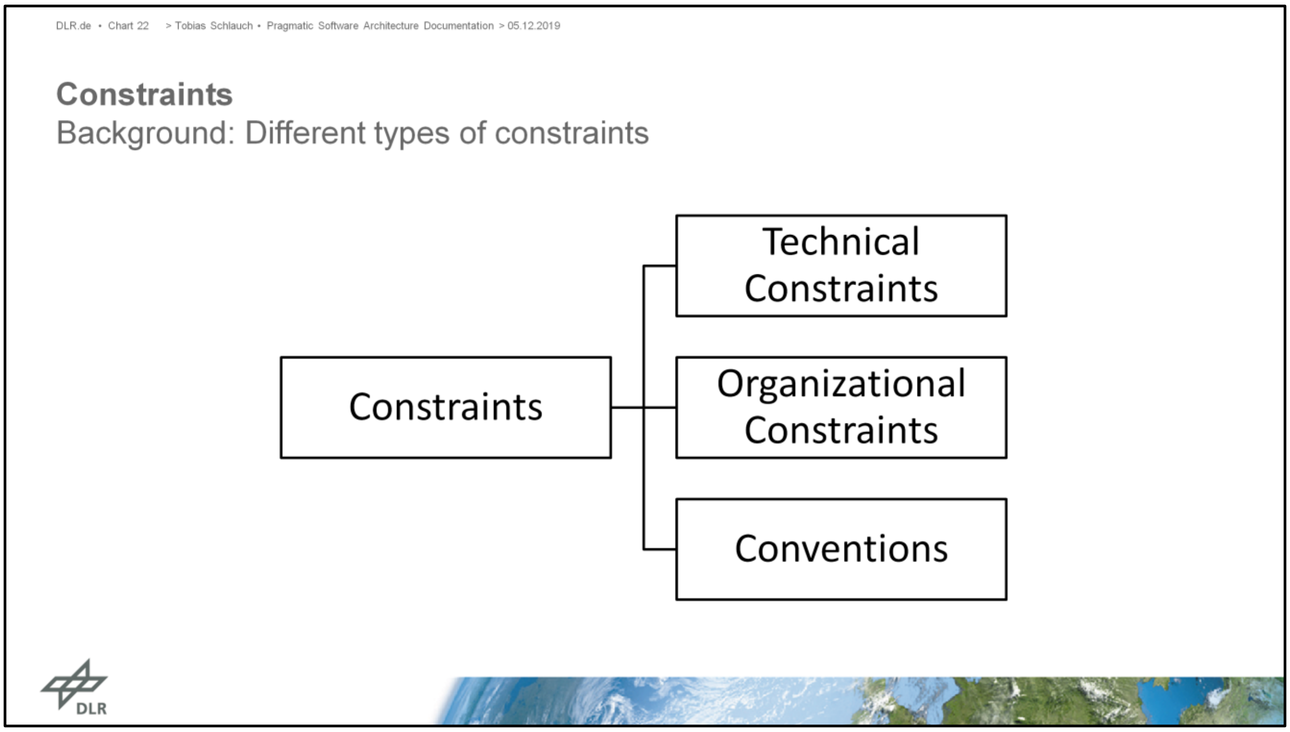
Constraints

02. Constraints

Constraint	Explanation
Recent FireFox ESR version	The Web clients have to support recent FireFox ESR versions. It is the officially supported Web browser available at the customer site.
Apache Tomcat 8	We re-use an existing software of the customer which already is built with Tomcat 8. There is no budget to change this.
Non-permanent Internet connection	The control server has only permanent access to the Intranet. It is an operational constraint of the datacenter in which the server is hosted.
…	…

DLR

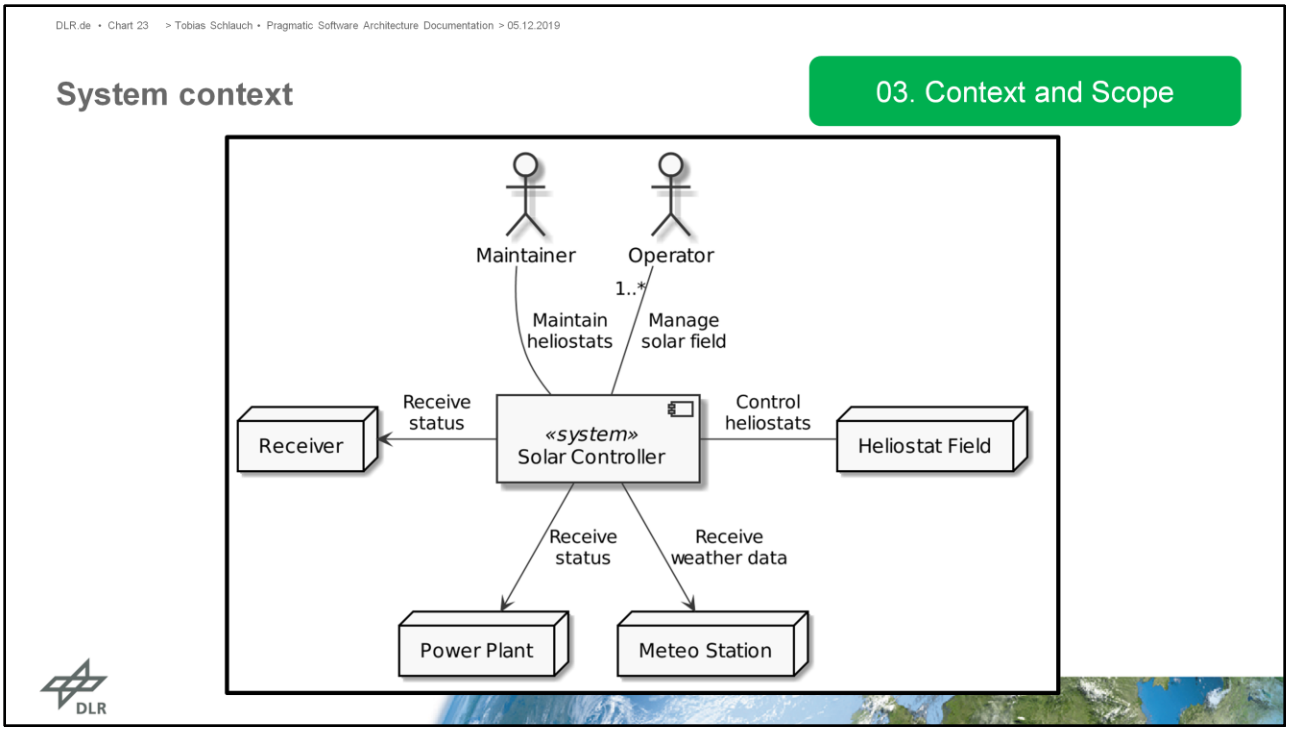
- Constraints are requirements which limit your choices in building / architecting the software system.
- It is important to collect, discuss, and communicate them. Otherwise the system might never find its way into production.
- **Beware:** Some constraints might only be a preference of someone involved!
- The table illustrates how you can document constraints. In this case three technical constraints are listed. Try to keep it short and reference other documents as needed. Particularly, you should explain the rational of the constraint.


DLR.de • Chart 22 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

Constraints
Background: Different types of constraints

Technical Constraints
Constraints
Organizational Constraints
Conventions
DLR

- Technical constraints are typically specifications concerning software, hardware, or operations.
- Organizational constraints might concern the organization/structure of the project or the available resources. In addition, they might require you to comply with organizational standards or legal aspects.
- Conventions include aspects such as programing or documentation guidelines.
- For further details, please see the arc42 documentation and Gernot Starke: "Effektive Software-Architekturen", Auflage 8, 2017.


DLR.de • Chart 23 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

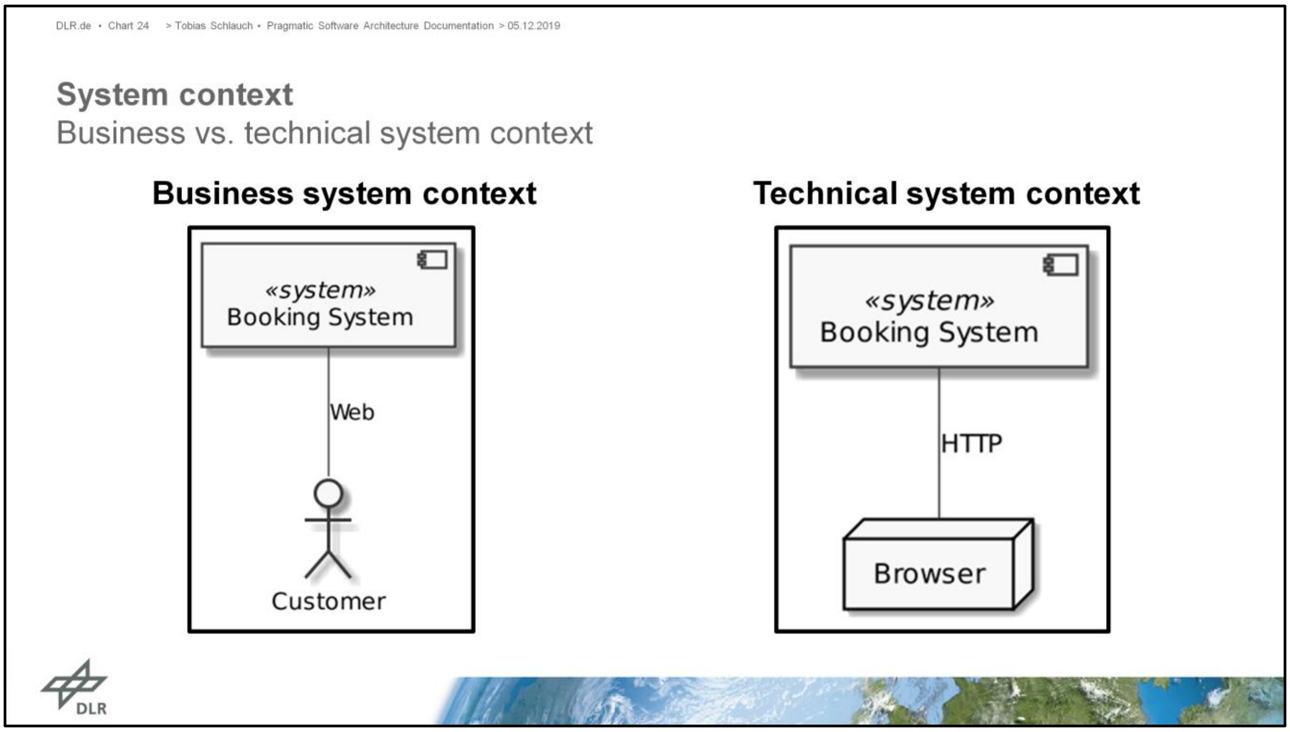
System context

03. Context and Scope

- The system context is used to differentiate want belongs into the system and what is outside the system. Particularly, it shows the involved stakeholders and third-party systems with which our software system has to interact.
- It is a very valuable view which should not change so often after the system is basically into production.
- The system context is a good starting point for architectural decisions.
- Make sure that you properly describe the diagram within your documentation!

Explanation of the diagram:

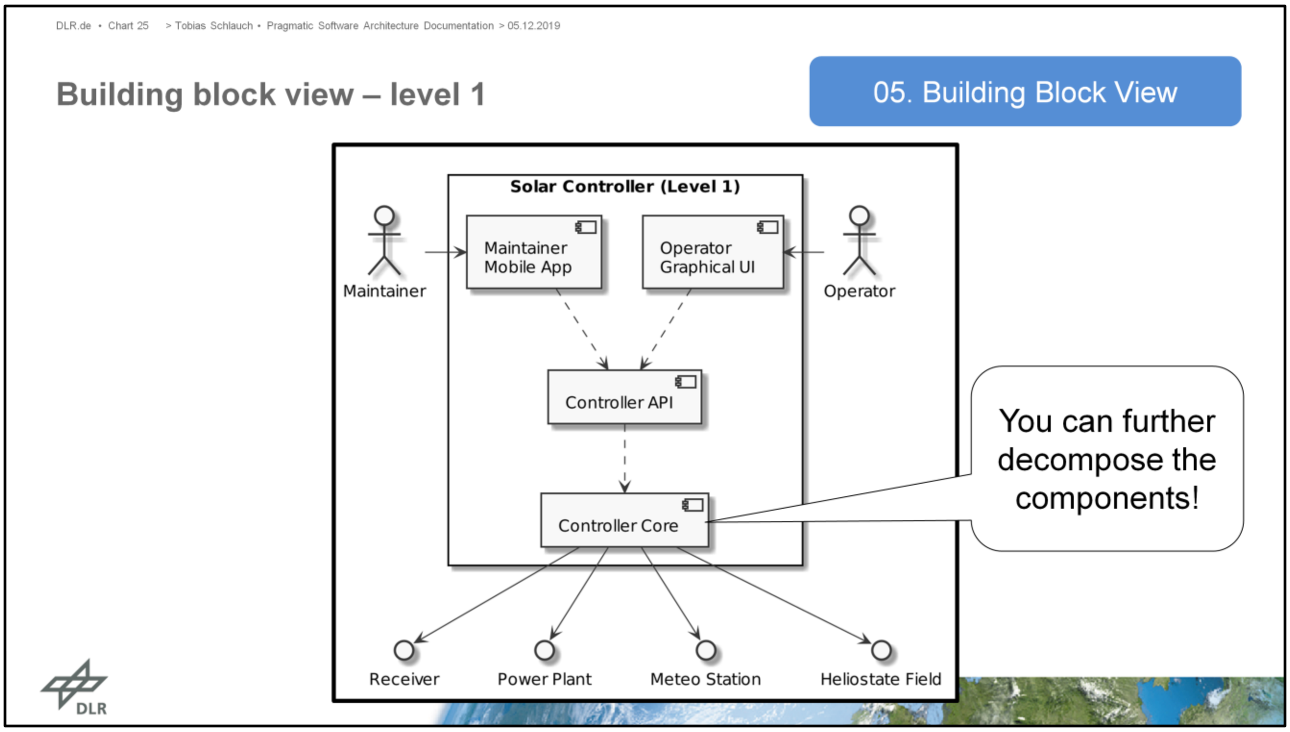
- In the middle is the software system we want to build (UML component).
- Lines show interactions with users/third-party systems outside the system.



DLR.de • Chart 24 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

System context
Business vs. technical system context

- You can differentiate between the business and the technical system context.
- Business context focuses on the domain.
- Technical context focuses on the technical protocols / details.



DLR.de • Chart 25 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

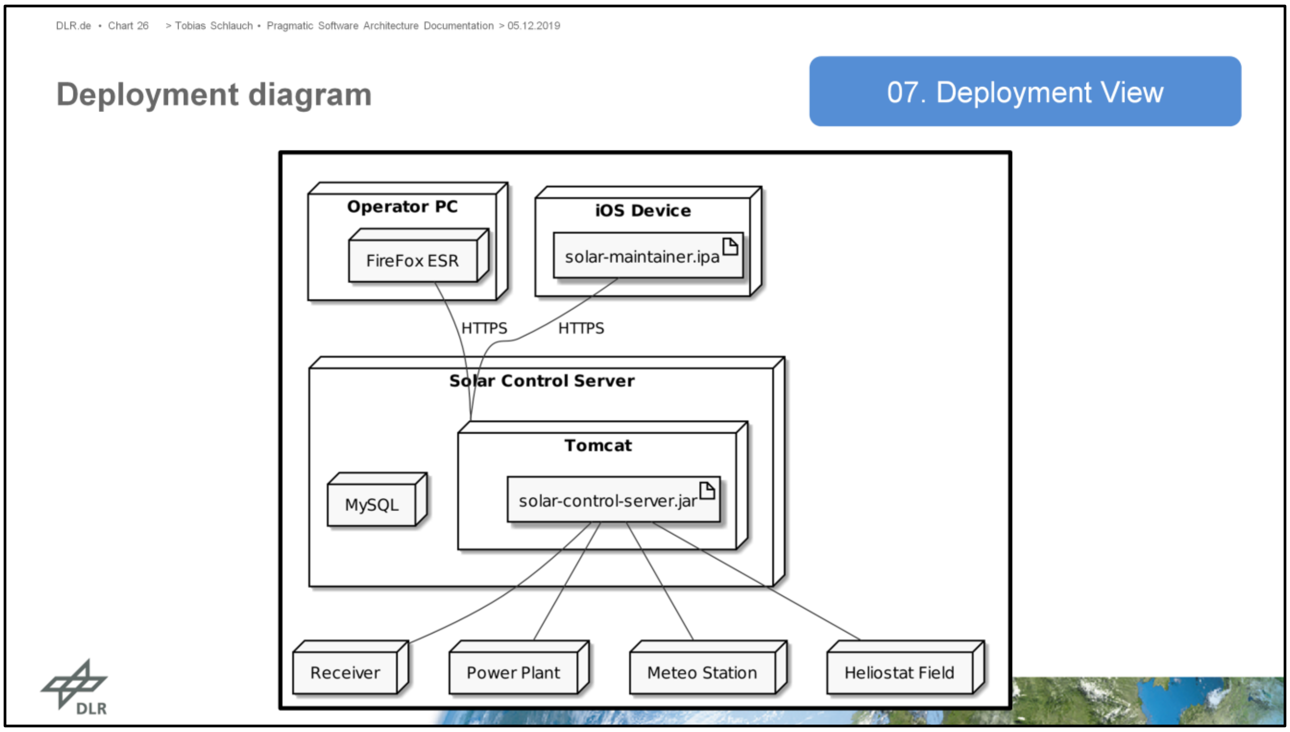
Building block view – level 1

05. Building Block View

- The building block view shows the decomposition of system. Its static structure. Here we have used the UML component diagram syntax.
- You can define further levels showing the details of a component. For instance, we could show the further decomposition of the component “Controller Core” which would be part of building block level 2.
- There are different ways to approach the decomposition: domain-specific vs. technical decomposition. Here you see a typical technical decomposition.

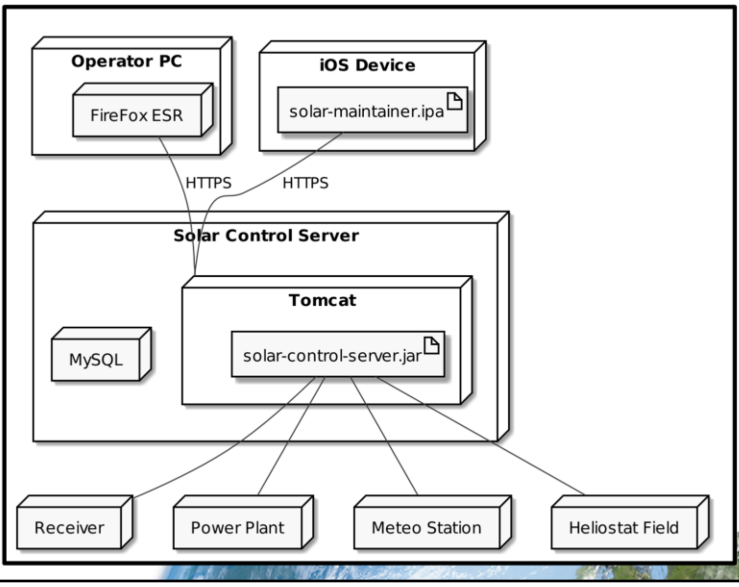
Tips:

- Make sure that the names of the component can be directly mapped to your source code.
- You should avoid too many decomposition levels here - particularly when you create them manually. 1-2 level should be sufficient. Otherwise it is hard to keep them up-to-date.
- The C4 model (https://c4model.com/) is a good alternative as it allows you to generate those views from your source code.
- Make sure that you properly describe the diagram within your documentation!

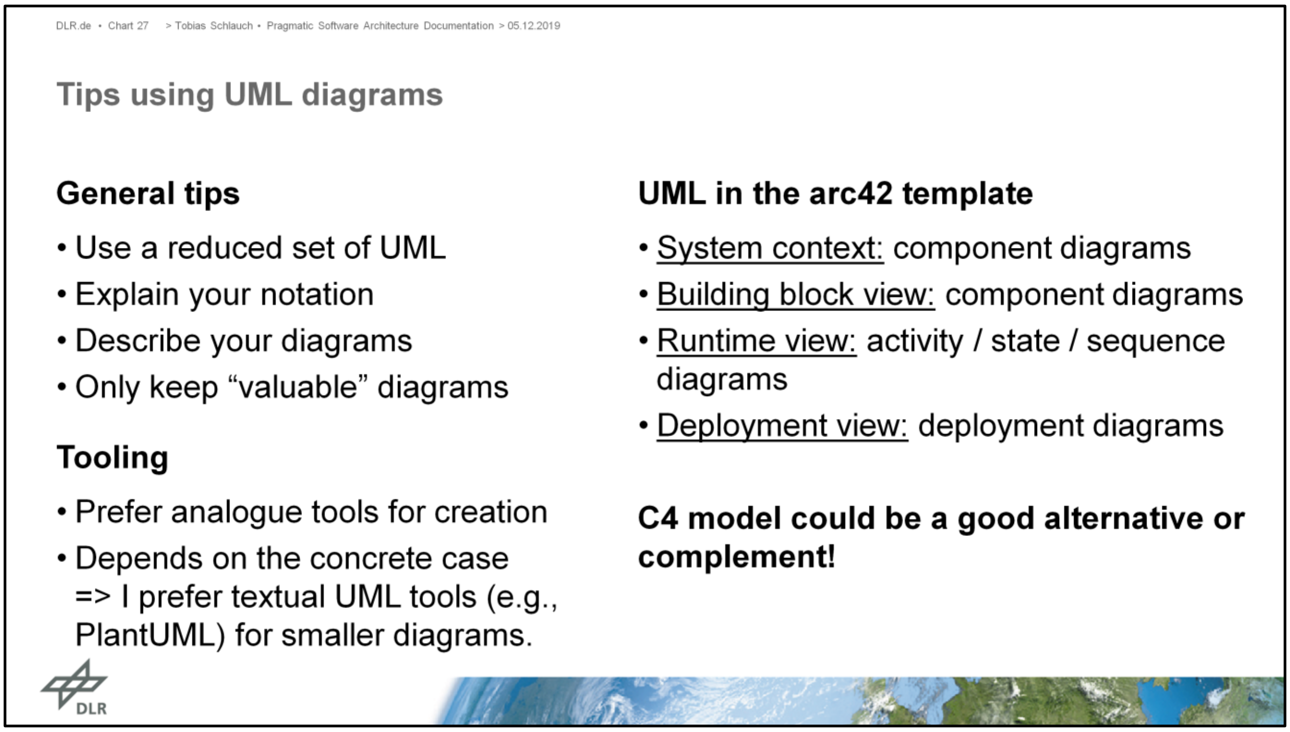


Deployment diagram

07. Deployment View



- The diagrams shows a UML deployment diagram which shows how the system is operated.


DLR.de • Chart 27 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

Tips using UML diagrams

General tips

- Use a reduced set of UML
- Explain your notation
- Describe your diagrams
- Only keep “valuable” diagrams

Tooling

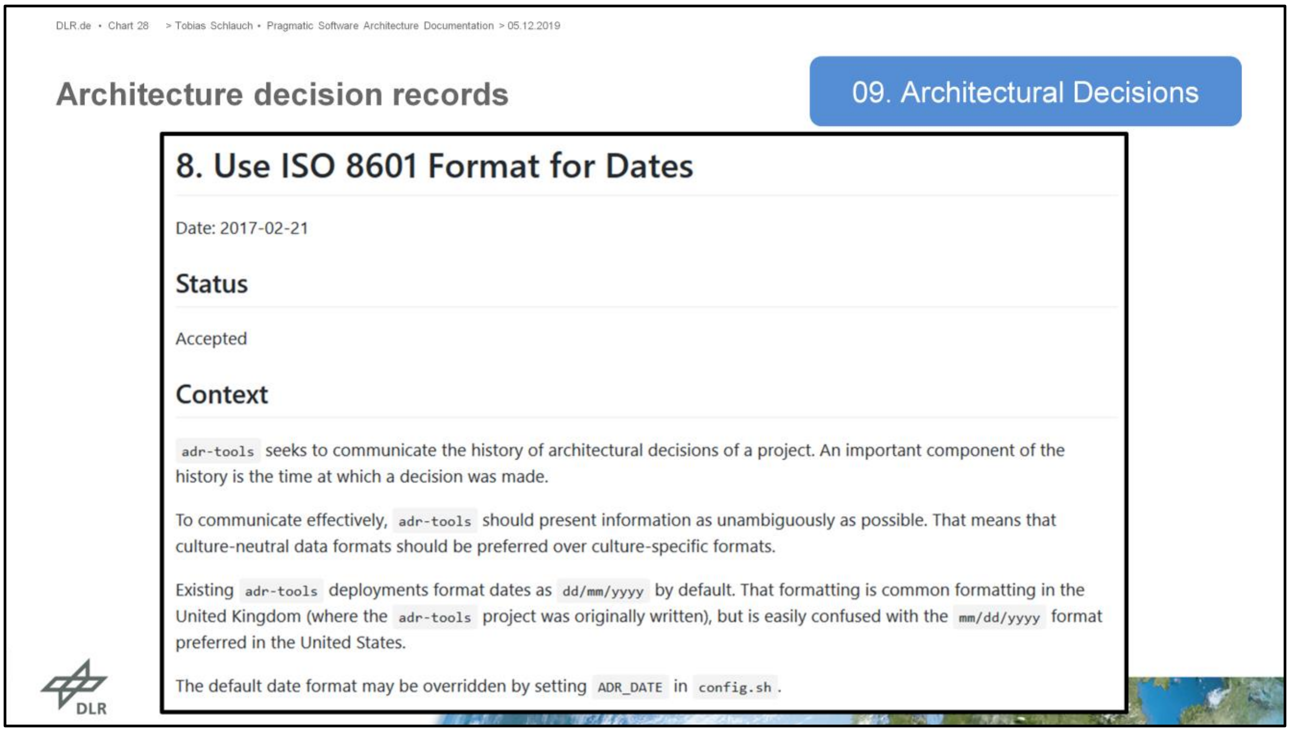
- Prefer analogue tools for creation
- Depends on the concrete case => I prefer textual UML tools (e.g., PlantUML) for smaller diagrams.

UML in the arc42 template

- System context: component diagrams
- Building block view: component diagrams
- Runtime view: activity / state / sequence diagrams
- Deployment view: deployment diagrams

C4 model could be a good alternative or complement!

- In general, arc42 makes no assumption whether you use or you do not use UML.
- There are also alternatives available such as C4 model https://c4model.com/ (focus on structure)
- PlantUML is a textual UML tool which is widely supported by different tools (e.g., editors/IDEs, development platforms, Wiki): https://plantuml.com/


DLR.de • Chart 28 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

Architecture decision records

09. Architectural Decisions

8. Use ISO 8601 Format for Dates

Date: 2017-02-21

Status

Accepted

Context

`adr-tools` seeks to communicate the history of architectural decisions of a project. An important component of the history is the time at which a decision was made.

To communicate effectively, `adr-tools` should present information as unambiguously as possible. That means that culture-neutral data formats should be preferred over culture-specific formats.

Existing `adr-tools` deployments format dates as `dd/mm/yyyy` by default. That formatting is common formatting in the United Kingdom (where the `adr-tools` project was originally written), but is easily confused with the `mm/dd/yyyy` format preferred in the United States.

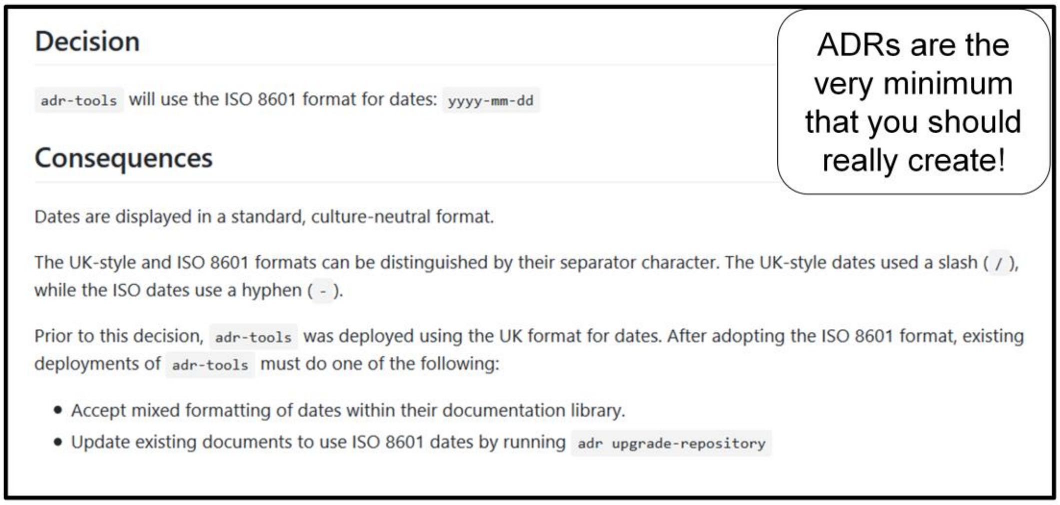
The default date format may be overridden by setting `ADR_DATE` in `config.sh`.

DLR

- Architecture decision records offer a compact/lean format to document architectural decisions. In addition, they offer you a way/template to approach those decisions.
- For more information, see Michael Nygard blog: http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions
- A good collection for ADR tools/templates is available here: https://github.com/joelparkerhenderson/architecture_decision_record

Architecture decision records (Cont.)

09. Architectural Decisions



Decision

`adr-tools` will use the ISO 8601 format for dates: `yyyy-mm-dd`

Consequences

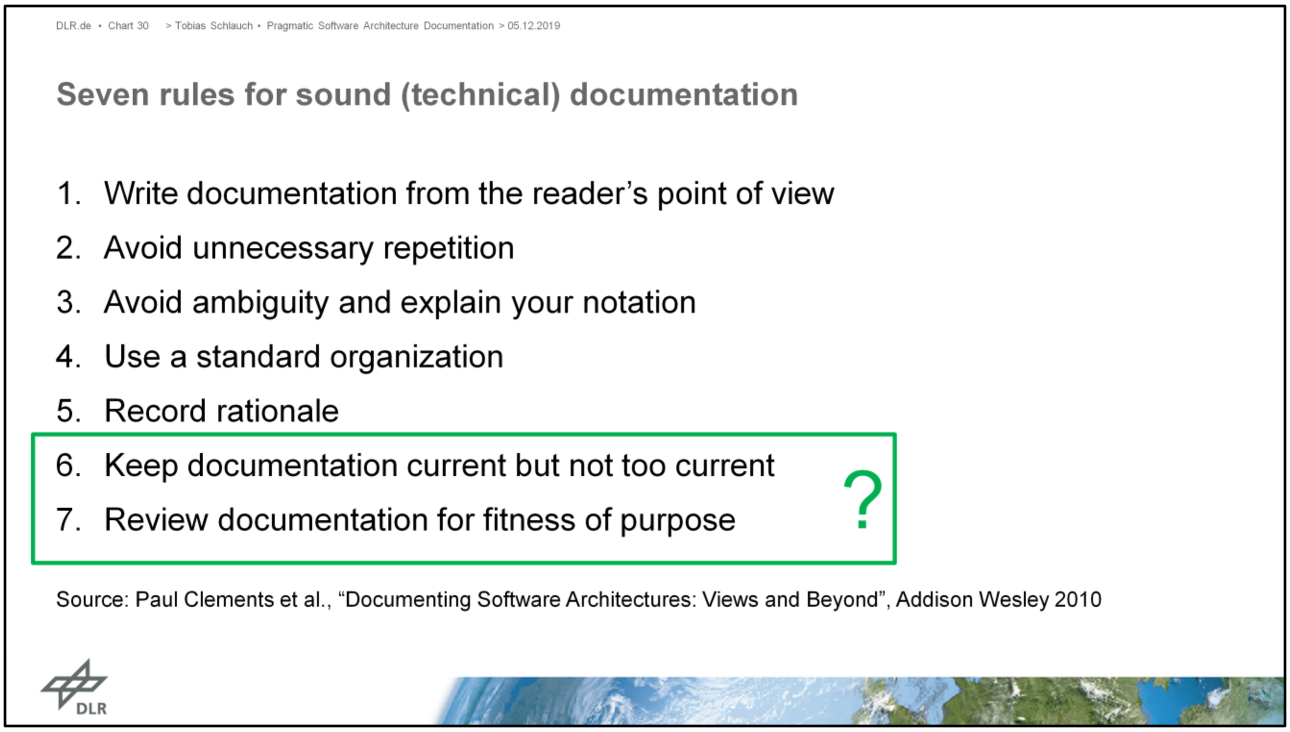
Dates are displayed in a standard, culture-neutral format.

The UK-style and ISO 8601 formats can be distinguished by their separator character. The UK-style dates used a slash (`/`), while the ISO dates use a hyphen (`-`).

Prior to this decision, `adr-tools` was deployed using the UK format for dates. After adopting the ISO 8601 format, existing deployments of `adr-tools` must do one of the following:

- Accept mixed formatting of dates within their documentation library.
- Update existing documents to use ISO 8601 dates by running `adr upgrade-repository`

ADRs are the very minimum that you should really create!

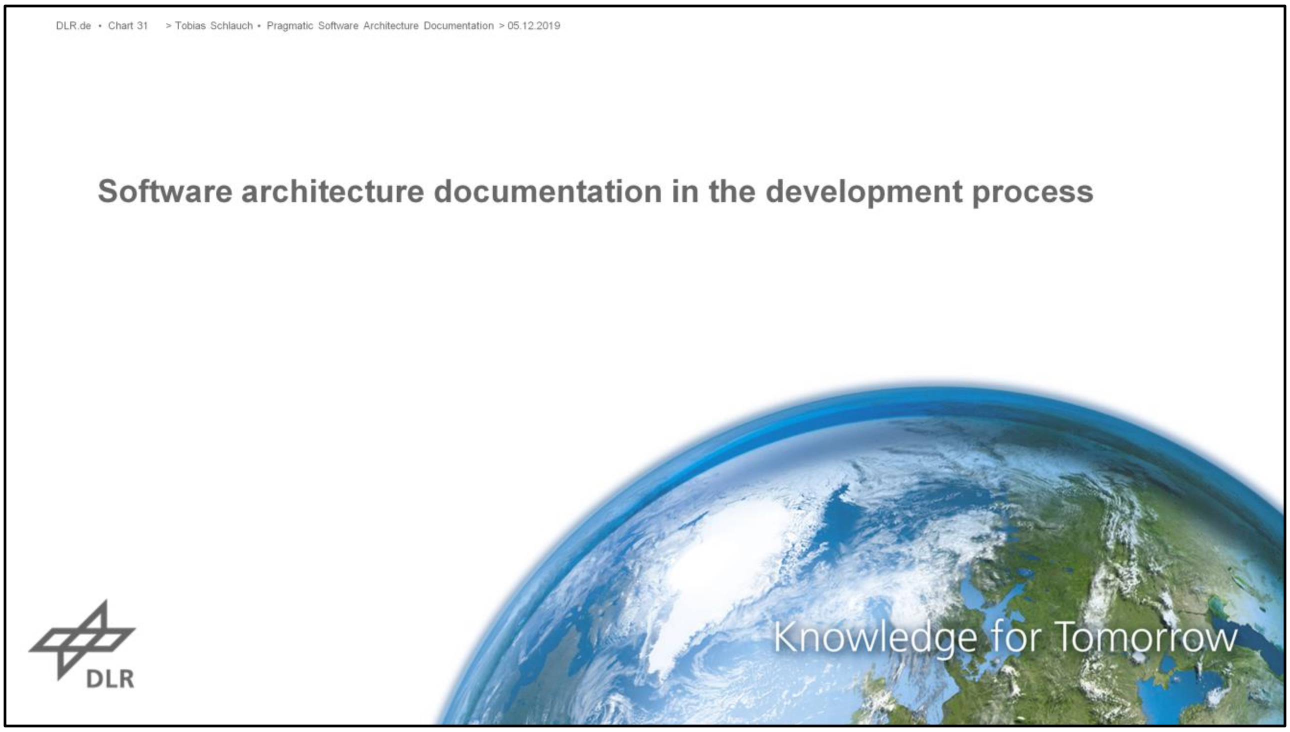


Seven rules for sound (technical) documentation

1. Write documentation from the reader's point of view
2. Avoid unnecessary repetition
3. Avoid ambiguity and explain your notation
4. Use a standard organization
5. Record rationale
6. Keep documentation current but not too current
7. Review documentation for fitness of purpose

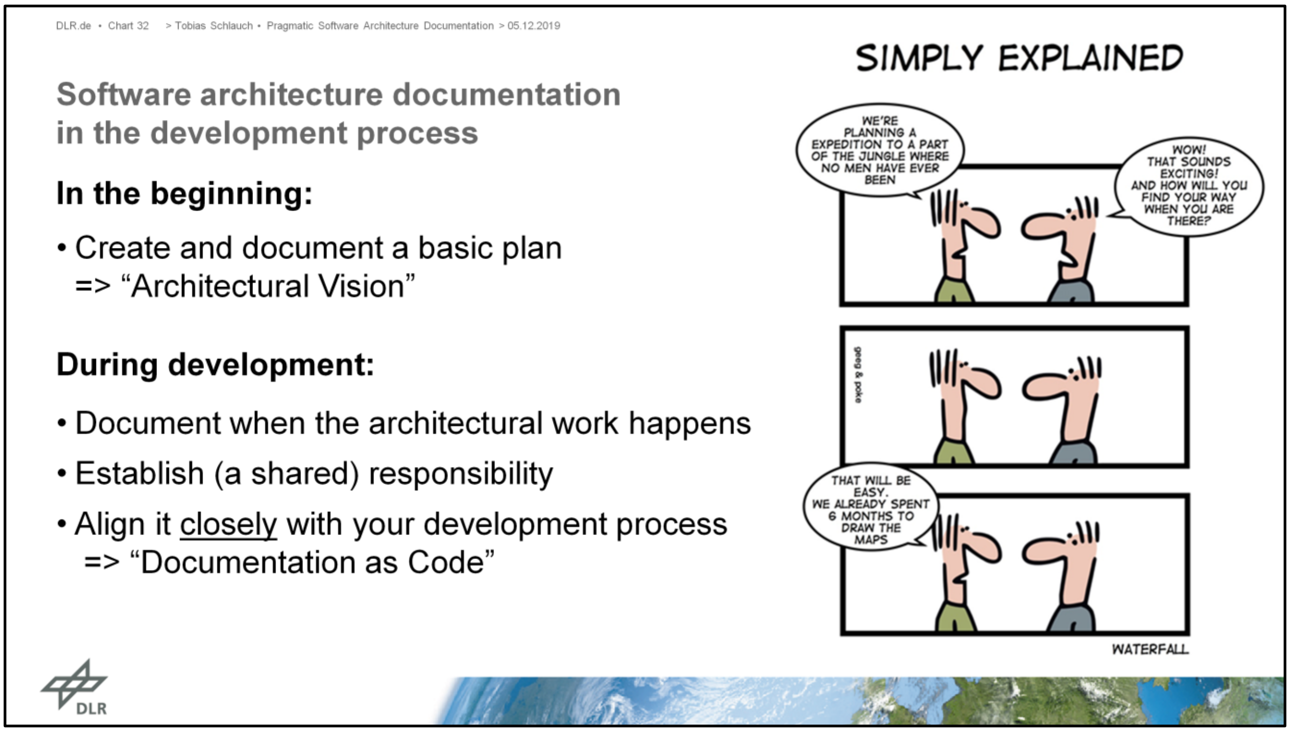
?

Source: Paul Clements et al., "Documenting Software Architectures: Views and Beyond", Addison Wesley 2010



Software architecture documentation in the development process

DLR



DLR.de • Chart 32 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

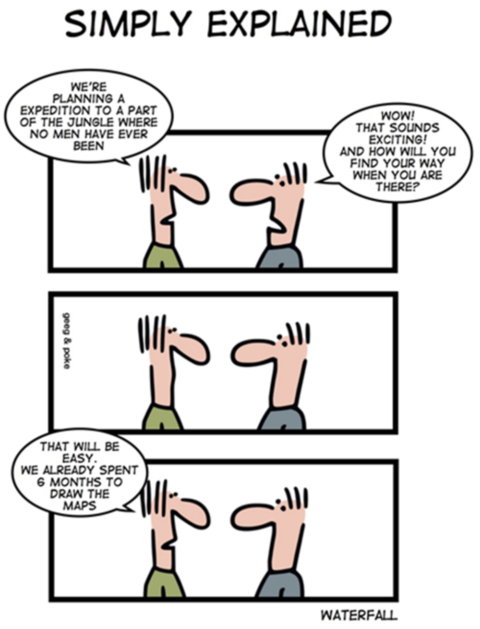
Software architecture documentation in the development process

In the beginning:

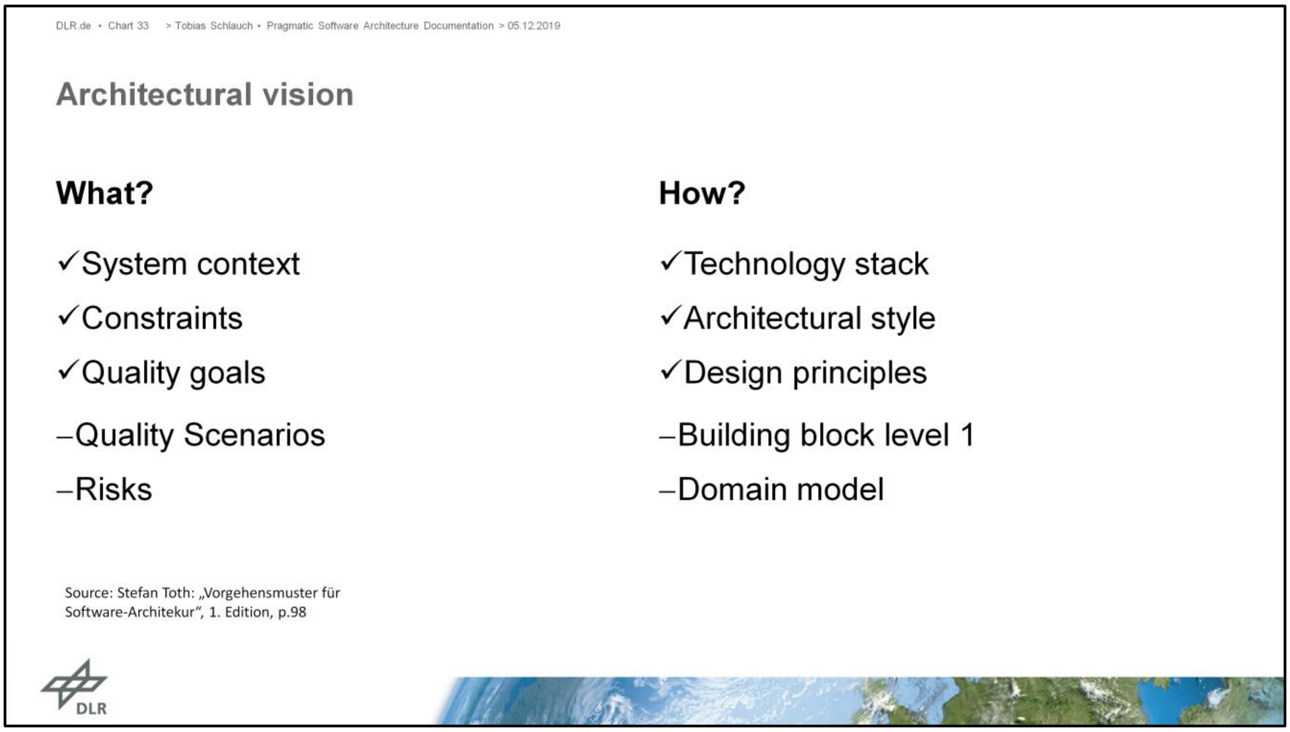
- Create and document a basic plan
  => “Architectural Vision”

During development:

- Document when the architectural work happens
- Establish (a shared) responsibility
- Align it closely with your development process
  => “Documentation as Code”


SIMPLY EXPLAINED
WE'RE PLANNING A EXPEDITION TO A PART OF THE JUNGLE WHERE NO MEN HAVE EVER BEEN
WOW! THAT SOUNDS EXCITING! AND HOW WILL YOU FIND YOUR WAY WHEN YOU ARE THERE?
THAT WILL BE EASY. WE ALREADY SPENT 6 MONTHS TO DRAW THE MAPS
geek & poke
WATERFALL

- In the 90 the focus has been on rigorous planning (BUFD) which could easily lead to analysis paralysis.
- The introduction of agile methods shifted the focus. Unfortunately, in the direction of having no plan at all.
- To sump: You should have a plan (see also: iteration 0 etc.) but should keep its creation lean.



Architectural vision

What?

✓System context

✓Constraints

✓Quality goals

–Quality Scenarios

–Risks

How?

✓Technology stack

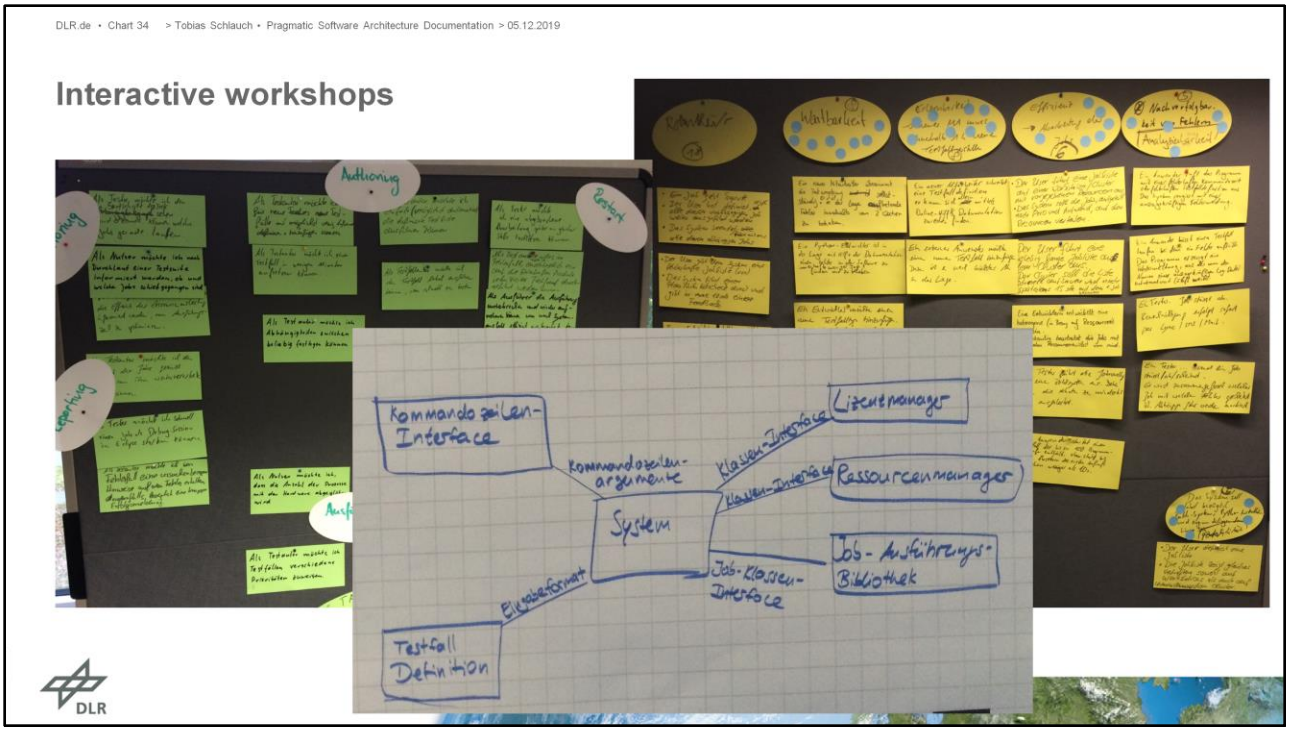
✓Architectural style

✓Design principles

–Building block level 1

–Domain model

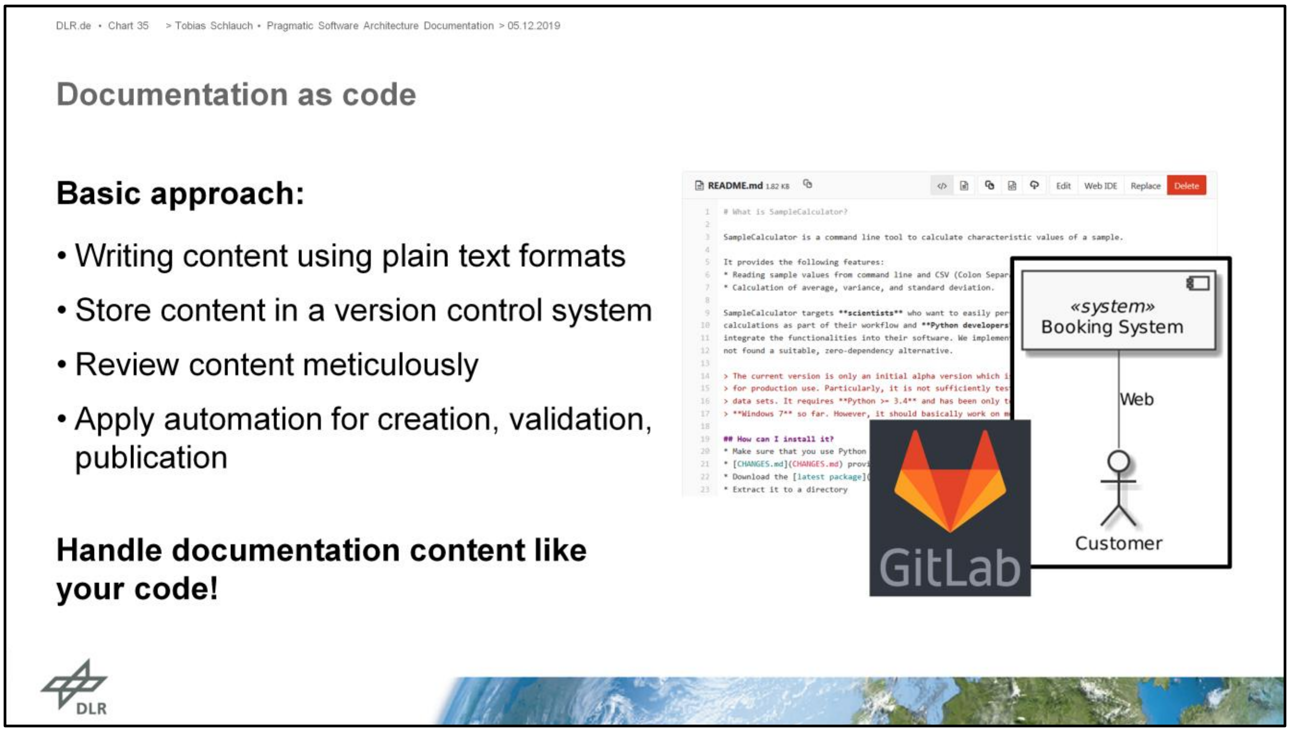
Source: Stefan Toth: „Vorgehensmuster für Software-Architekur“, 1. Edition, p.98



DLR.de • Chart 34 > Tobias Schlauch • Pragmatic Software Architecture Documentation > 05.12.2019

Interactive workshops

- Interactive workshops for requirement collection and performing architectural work are very effective ways to collect/create architectural content.
- You should involve all important stakeholders and should focus on interactive formats such as group work, brain writing etc.
- These workshops are particularly effective for prioritization and decision making.
- Stefan Toth: „Vorgehensmuster für Software-Architekur“ provides some good ideas in this regard.

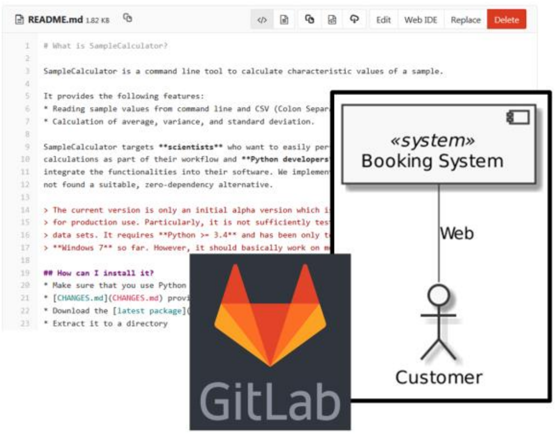


Documentation as code

Basic approach:

- Writing content using plain text formats
- Store content in a version control system
- Review content meticulously
- Apply automation for creation, validation, publication

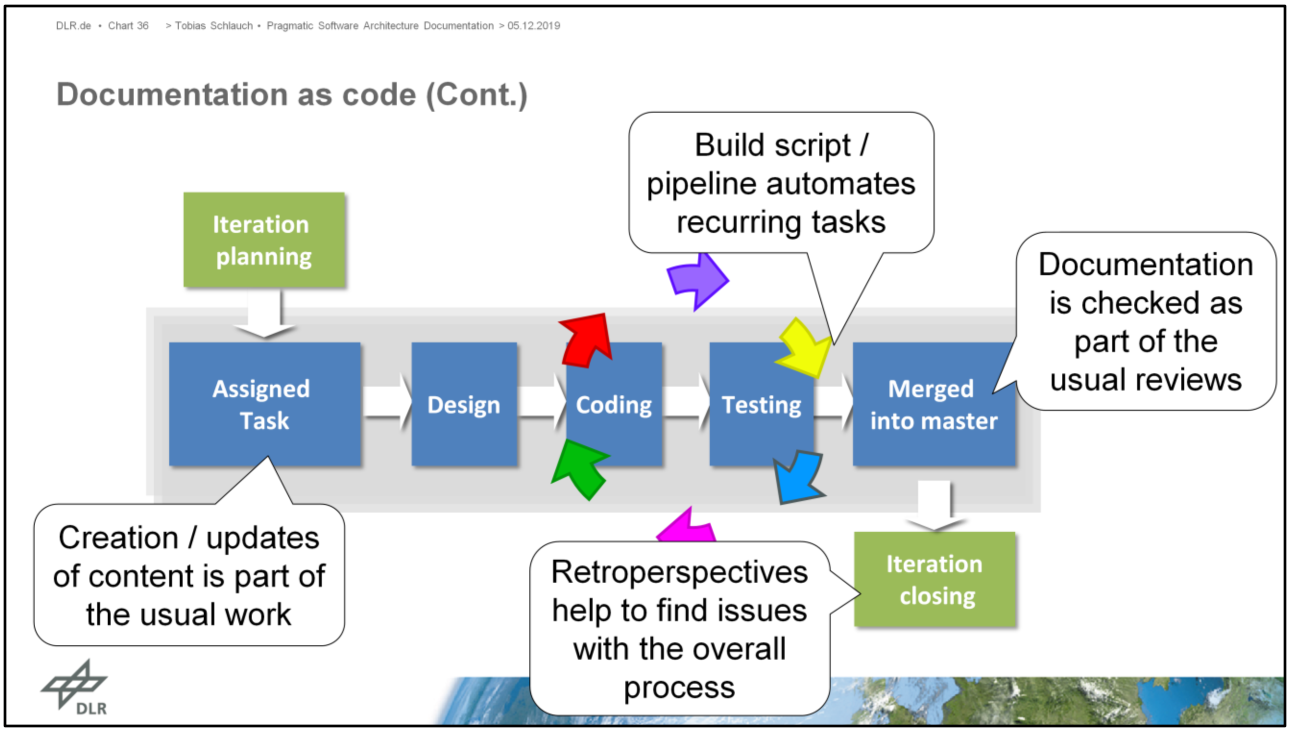
Handle documentation content like your code!

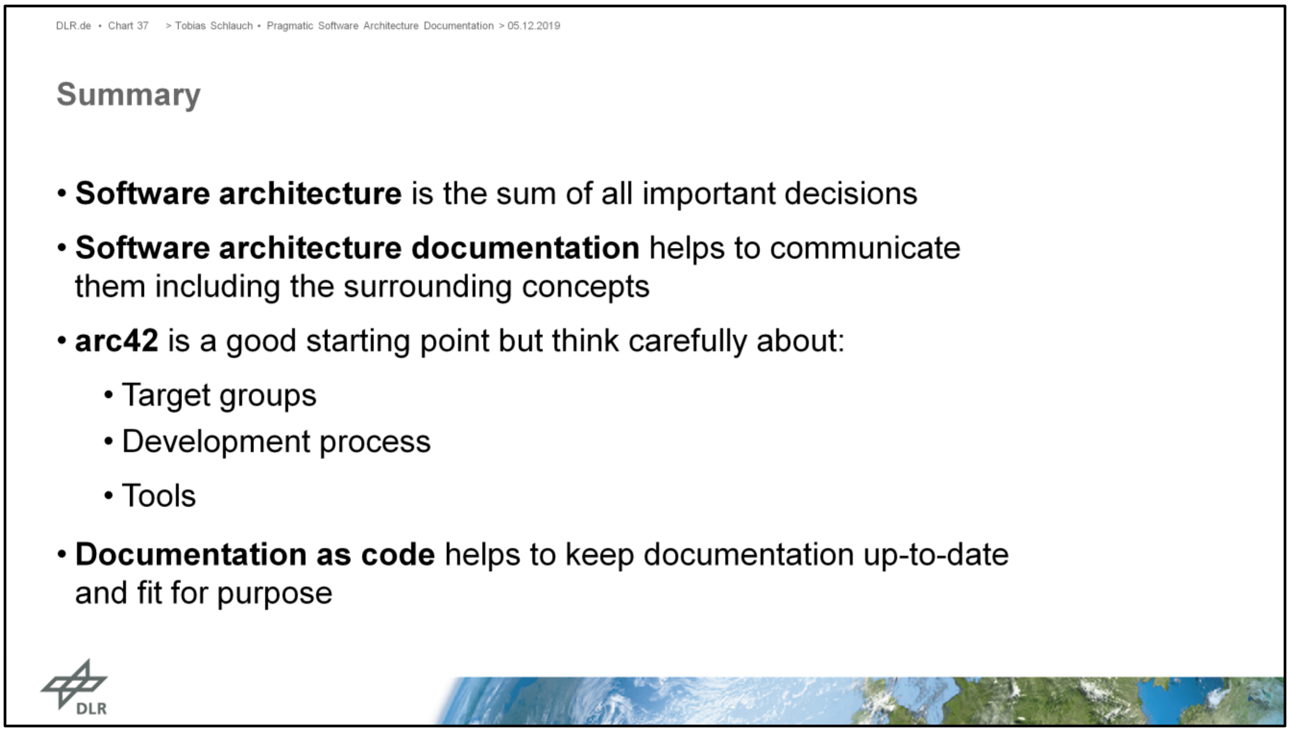


```
# What is SampleCalculator?

SampleCalculator is a command line tool to calculate characteristic values of a sample.

It provides the following features:
* Reading sample values from command line and CSV (Colon Separ
* Calculation of average, variance, and standard deviation.

SampleCalculator targets **scientists** who want to easily per
calculations as part of their workflow and **Python developers
integrate the functionalities into their software. We implemen
not found a suitable, zero-dependency alternative.

> The current version is only an initial alpha version which i
> for production use. Particularly, it is not sufficiently tes
> data sets. It requires **Python >= 3.4** and has been only t
> **Windows 7** so far. However, it should basically work on m

## How can I install it?
* Make sure that you use Python
* [CHANGES.md](CHANGES.md) provi
* Download the [latest package](
* Extract it to a directory
```

- The “Write the Docs” community provides a good overview about it: https://www.writethedocs.org/guide/docs-as-code/
- Suitable markup languages include AsciiDoc, MarkDown, and RestructredText.
- docToolchain provides an implementation of this approach: https://doctoolchain.github.io/docToolchain/

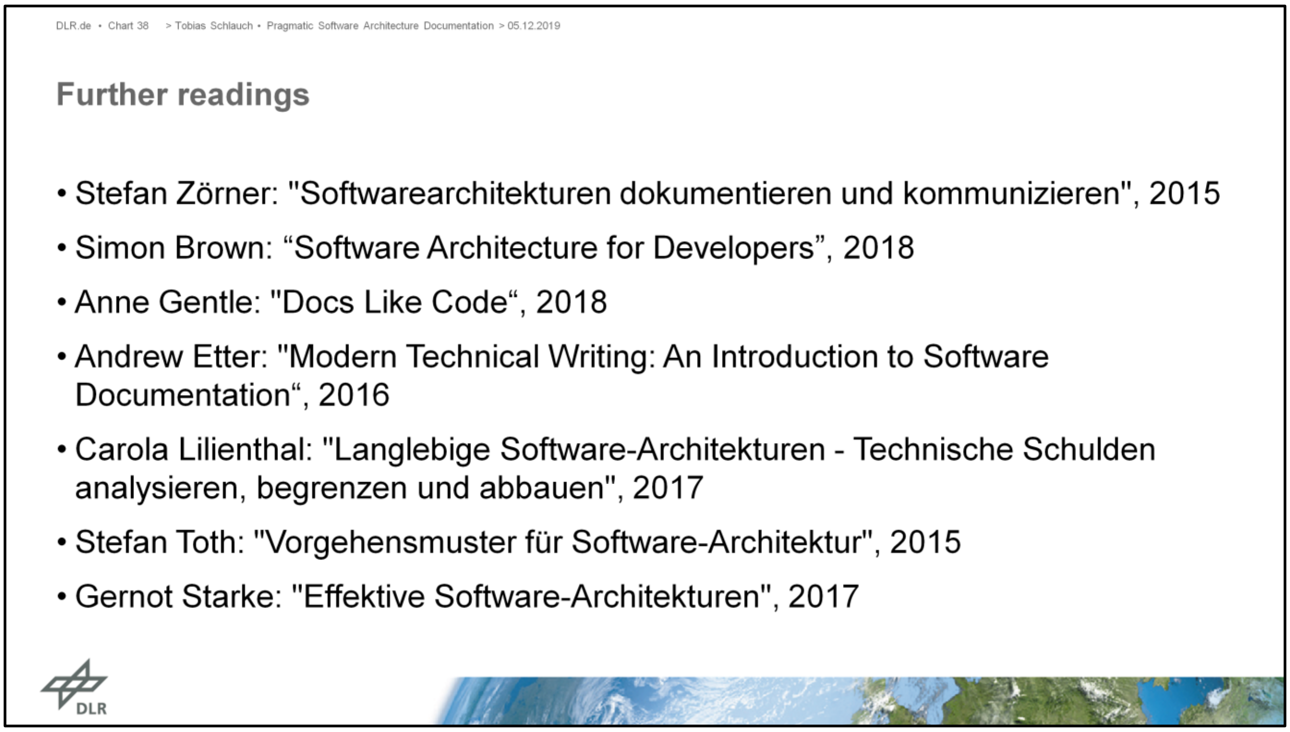
Documentation as code (Cont.)


Build script / pipeline automates recurring tasks
Iteration planning
Documentation is checked as part of the usual reviews
Assigned Task
Design
Coding
Testing
Merged into master
Iteration closing
Creation / updates of content is part of the usual work
Retroperspectives help to find issues with the overall process
DLR



Summary

- **Software architecture** is the sum of all important decisions
- **Software architecture documentation** helps to communicate them including the surrounding concepts
- **arc42** is a good starting point but think carefully about:
  - Target groups
  - Development process
  - Tools
- **Documentation as code** helps to keep documentation up-to-date and fit for purpose



Further readings

- Stefan Zörner: "Softwarearchitekturen dokumentieren und kommunizieren", 2015
- Simon Brown: “Software Architecture for Developers”, 2018
- Anne Gentle: "Docs Like Code“, 2018
- Andrew Etter: "Modern Technical Writing: An Introduction to Software Documentation“, 2016
- Carola Lilienthal: "Langlebige Software-Architekturen - Technische Schulden analysieren, begrenzen und abbauen", 2017
- Stefan Toth: "Vorgehensmuster für Software-Architektur", 2015
- Gernot Starke: "Effektive Software-Architekturen", 2017

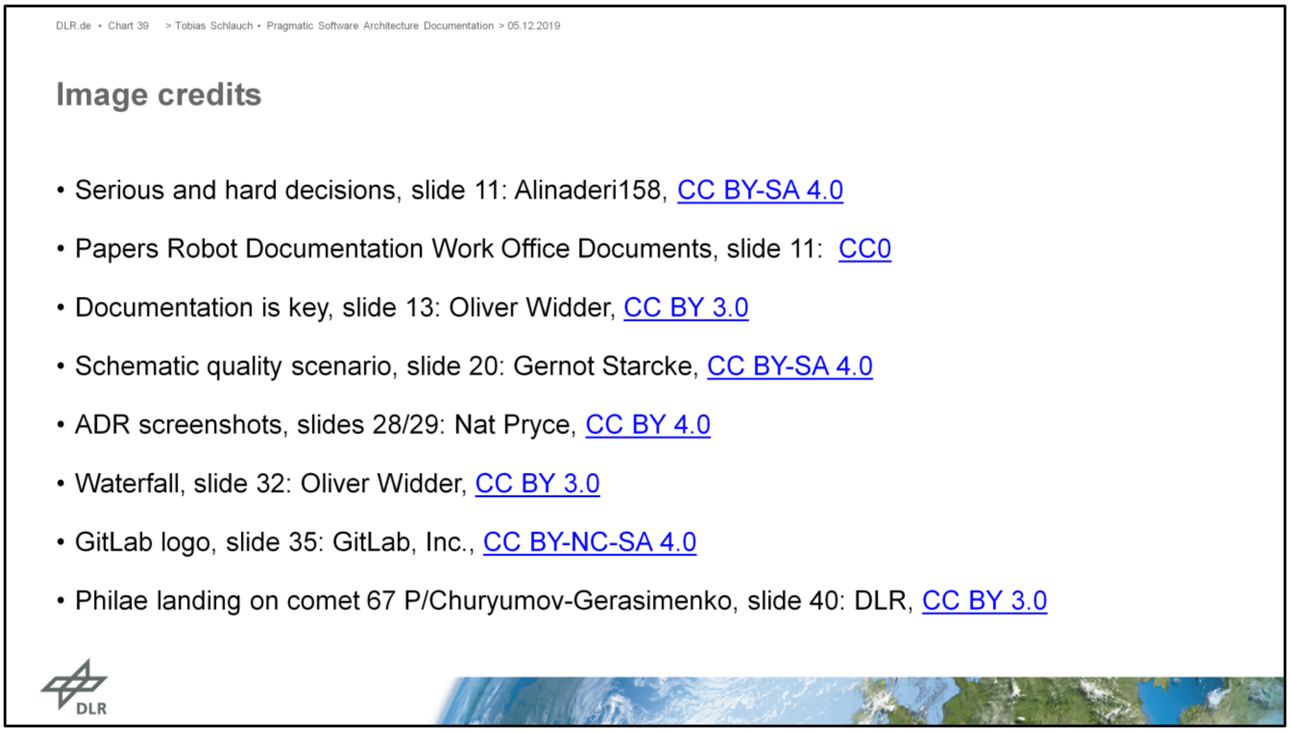


Image credits

- Serious and hard decisions, slide 11: Alinaderi158, CC BY-SA 4.0
- Papers Robot Documentation Work Office Documents, slide 11: CC0
- Documentation is key, slide 13: Oliver Widder, CC BY 3.0
- Schematic quality scenario, slide 20: Gernot Starcke, CC BY-SA 4.0
- ADR screenshots, slides 28/29: Nat Pryce, CC BY 4.0
- Waterfall, slide 32: Oliver Widder, CC BY 3.0
- GitLab logo, slide 35: GitLab, Inc., CC BY-NC-SA 4.0
- Philae landing on comet 67 P/Churyumov-Gerasimenko, slide 40: DLR, CC BY 3.0

Thank you!

Questions?

Tobias.Schlauch@dlr.de

www.DLR.de/sc | @TobiasSchlauch