Pragmatic Software Architecture Documentation

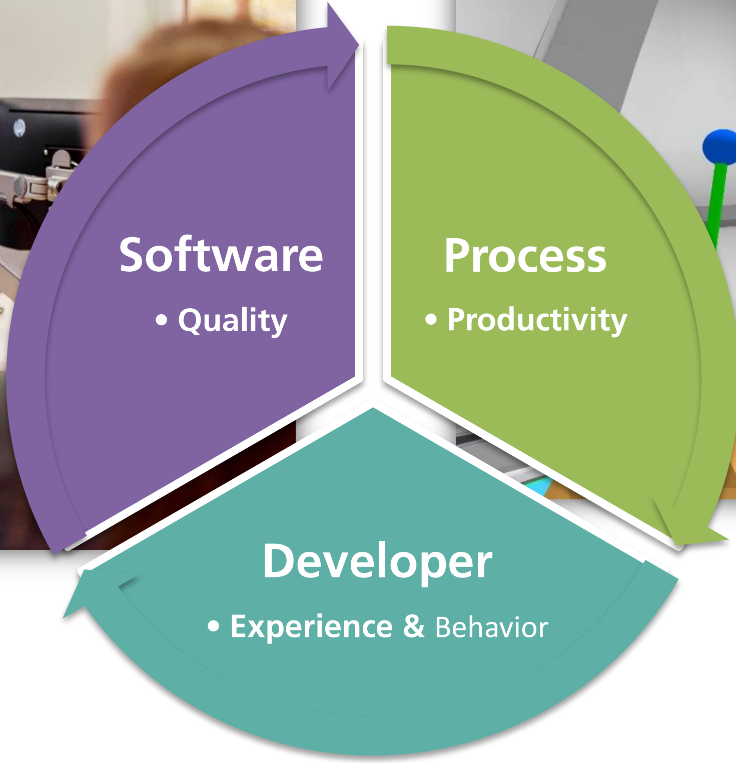
Tobias Schlauch

German Aerospace Center (DLR)
Simulation and Software Technology

05.12.2019, Hochschule Hannover

License hint

DLR Simulation and Software Technology
Group Software Engineering



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

What is software architecture?

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]*

What is software architecture? (Cont.)

- Software Architecture =
  Sum of all architectural decisions

- Architectural decisions =
  Fundamental decision which cannot be easily changed afterwards

What are architectural decisions?

Check questions:

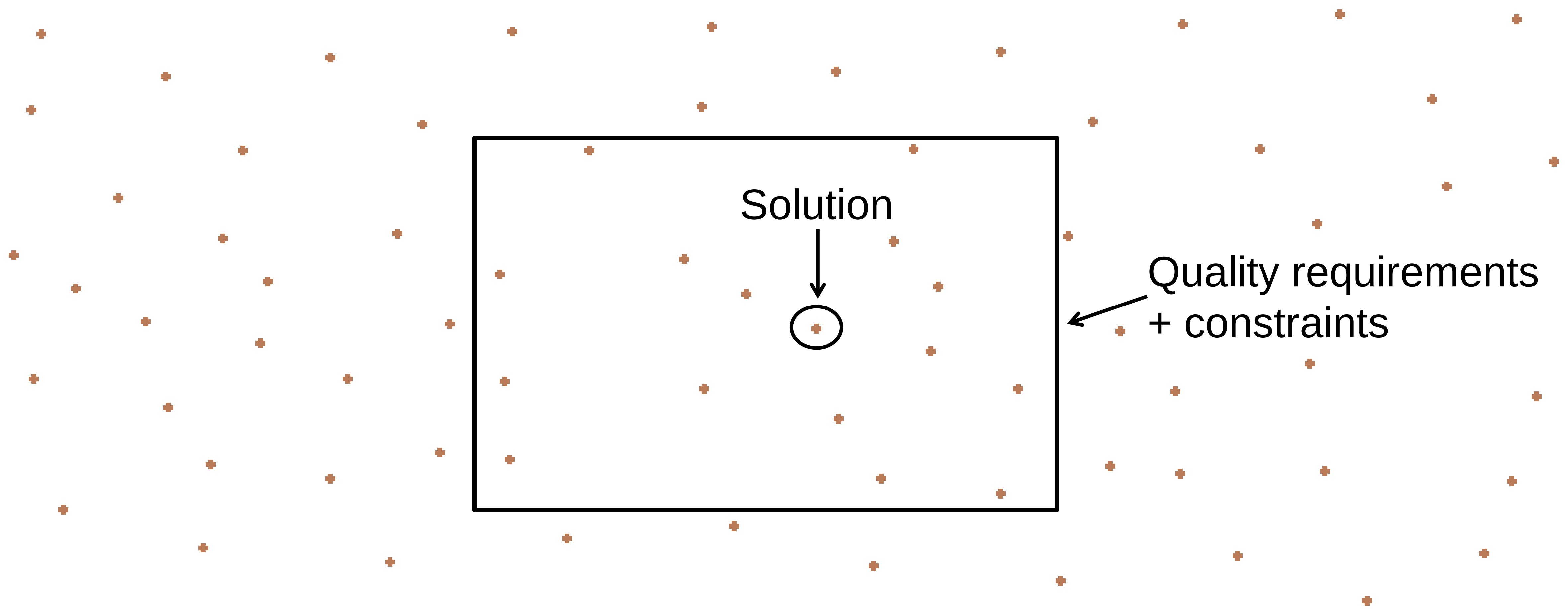
1. Is the decision hard to change later?
2. Is the implementation of the decision expensive?
3. Are there high quality requirements involved?
4. Is it hard to map requirements to already existing functionality?
5. Is your experience in the solution spectrum rather weak?

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

Software architecture guides selection of a suitable solution



Introduction to software architecture documentation

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!

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

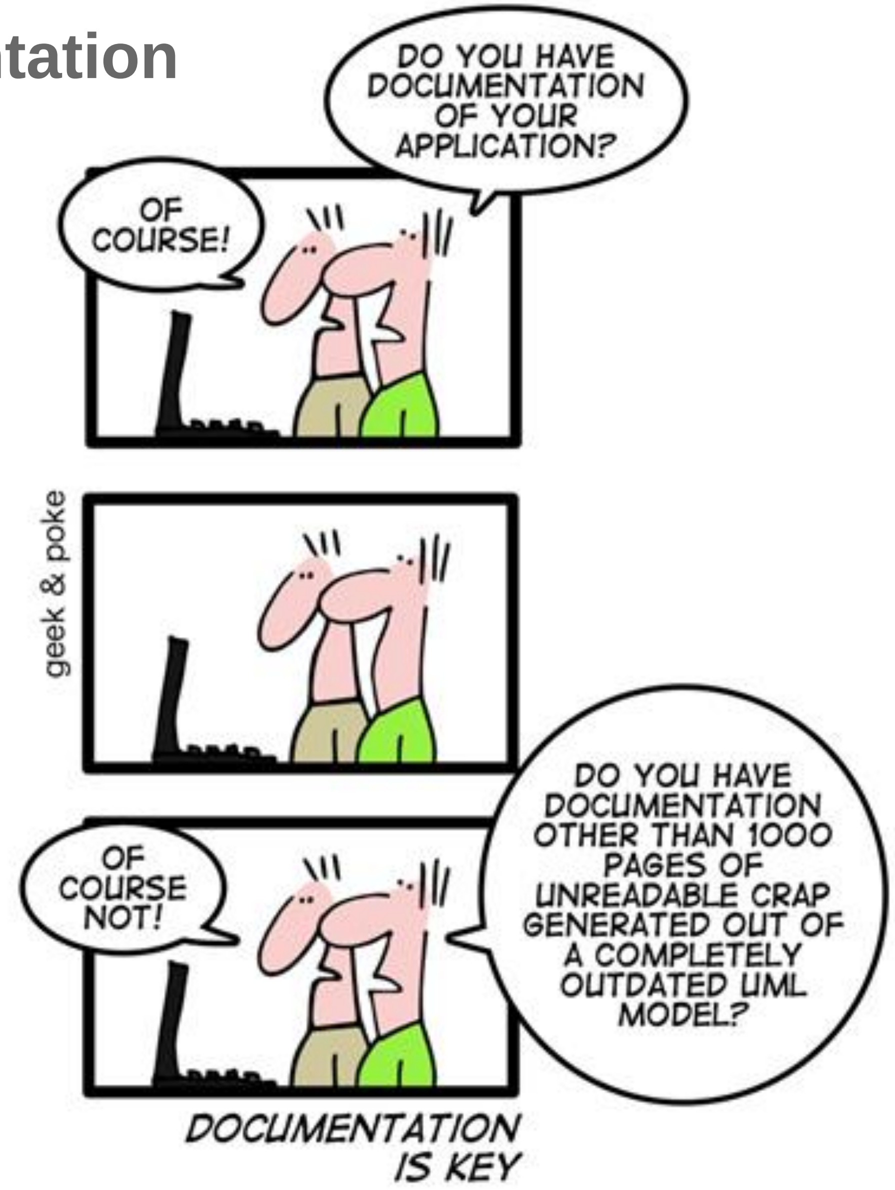
Source: Paul Clements et al., “Documenting Software Architectures: Views and Beyond”, Addison Wesley 2010

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

arc42 – A pragmatic template for software architecture documentation

DLR

arc42 – A pragmatic template for software architecture documentation

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

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.

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:

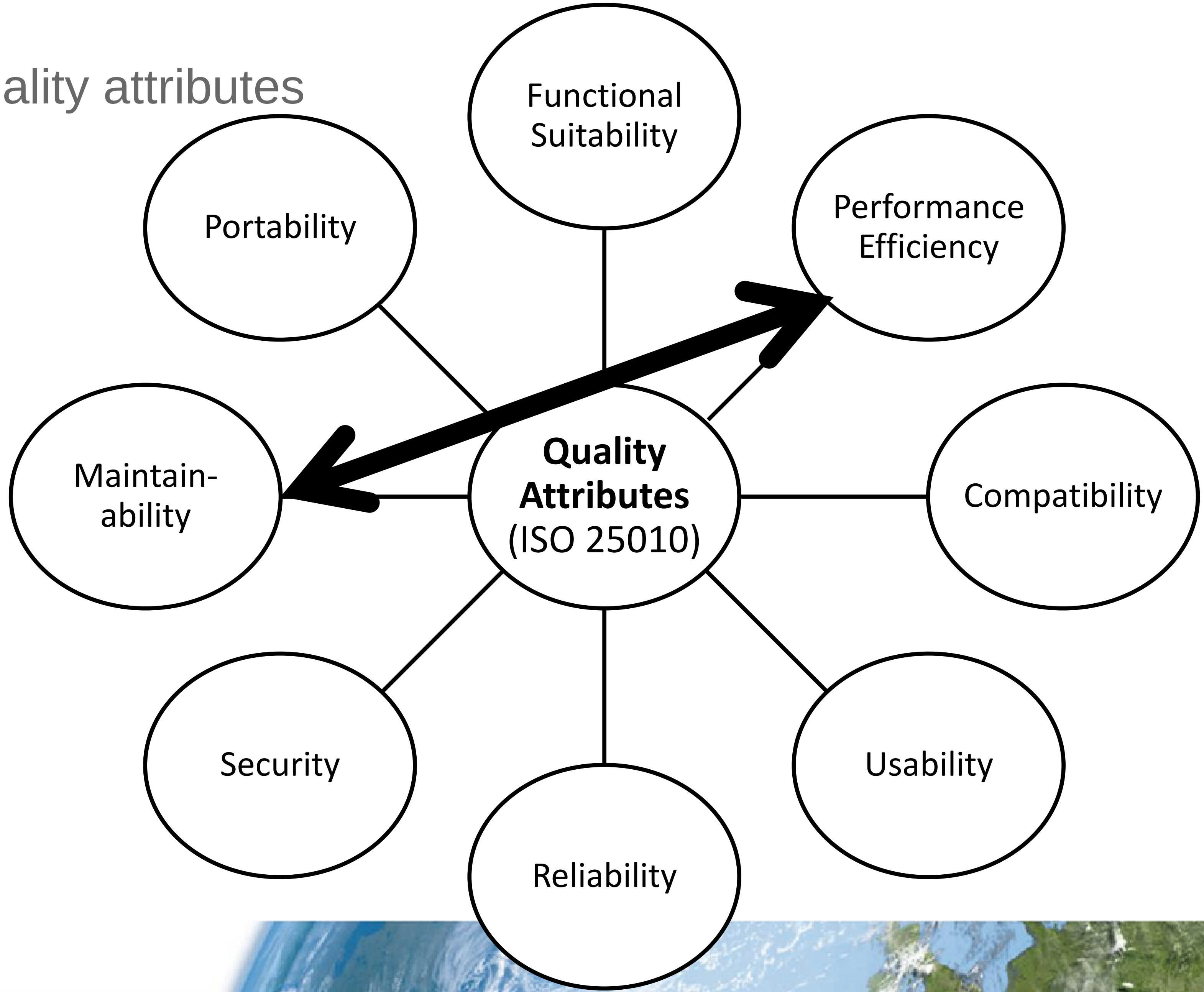
- Set up of the solar field and definition of standard operation procedures for solar fields up to 10000 heliostats and 10 receivers
- Autonomous performance of standard operation procedures
- Integrated monitoring, evaluation and alert functionalities
- Support of a wide range of heliostat and receiver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

Quality goals

01. Introduction and Goals

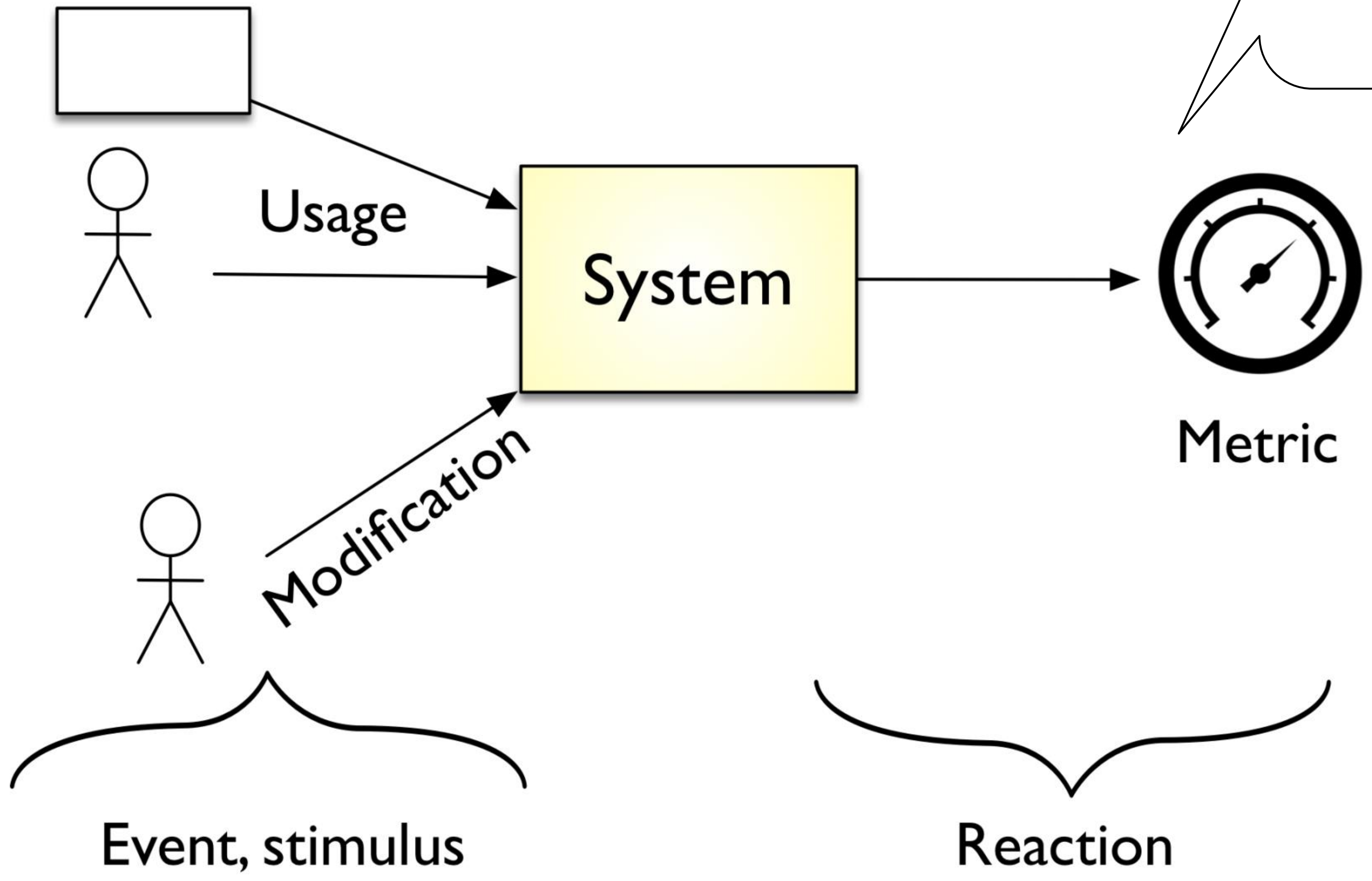
Quality Attribute	Goal	Quality Scenarios
Reliability	The system ensures the safe operation of the solar field. It guides the operators through the whole process and reliably protects them from operational errors. In addition, it takes into account typical operational conditions to prevent damages.	3,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

Quality goals
Background: Quality attributes



Quality goals
Background: Quality scenarios

Scenarios drive the architectural work!



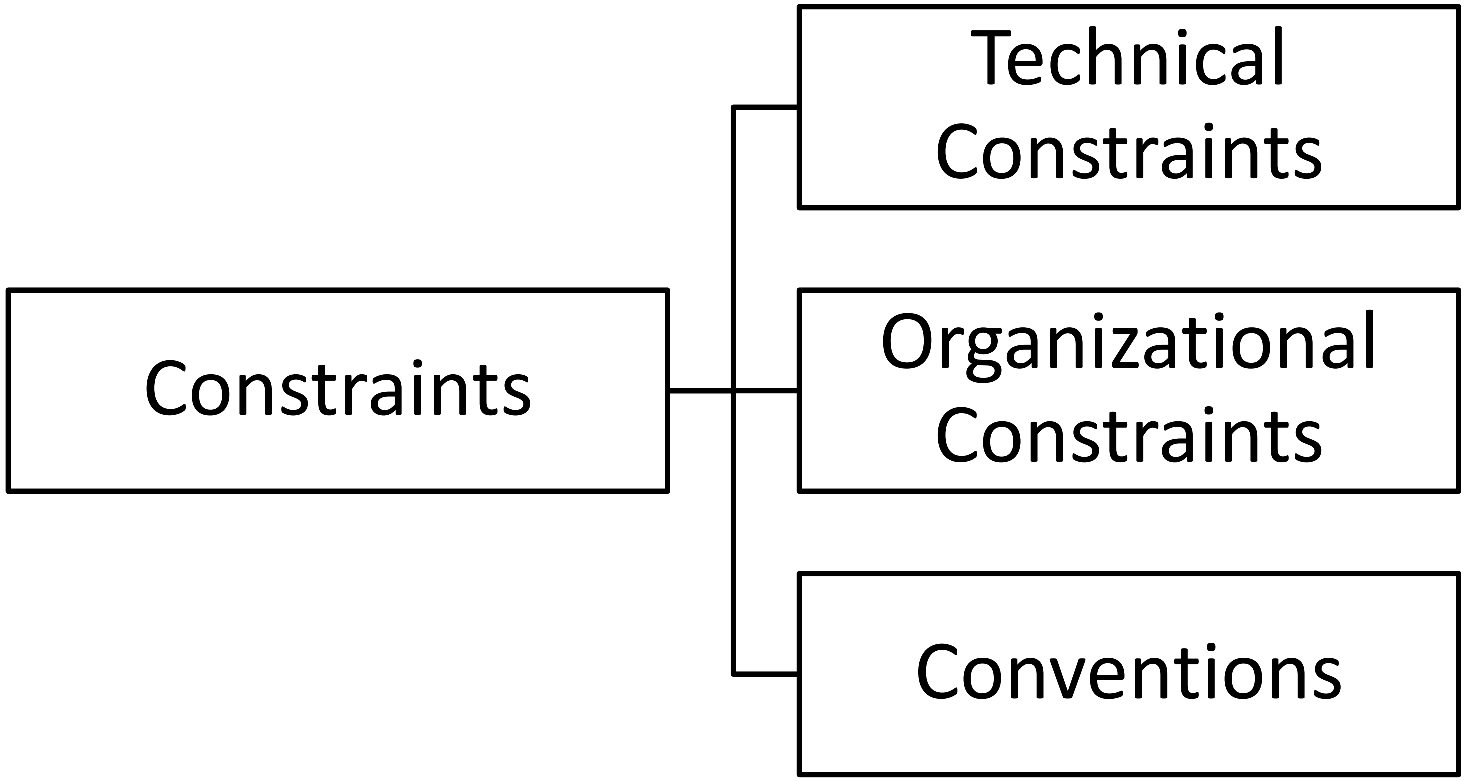
Constraints

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.
…	…

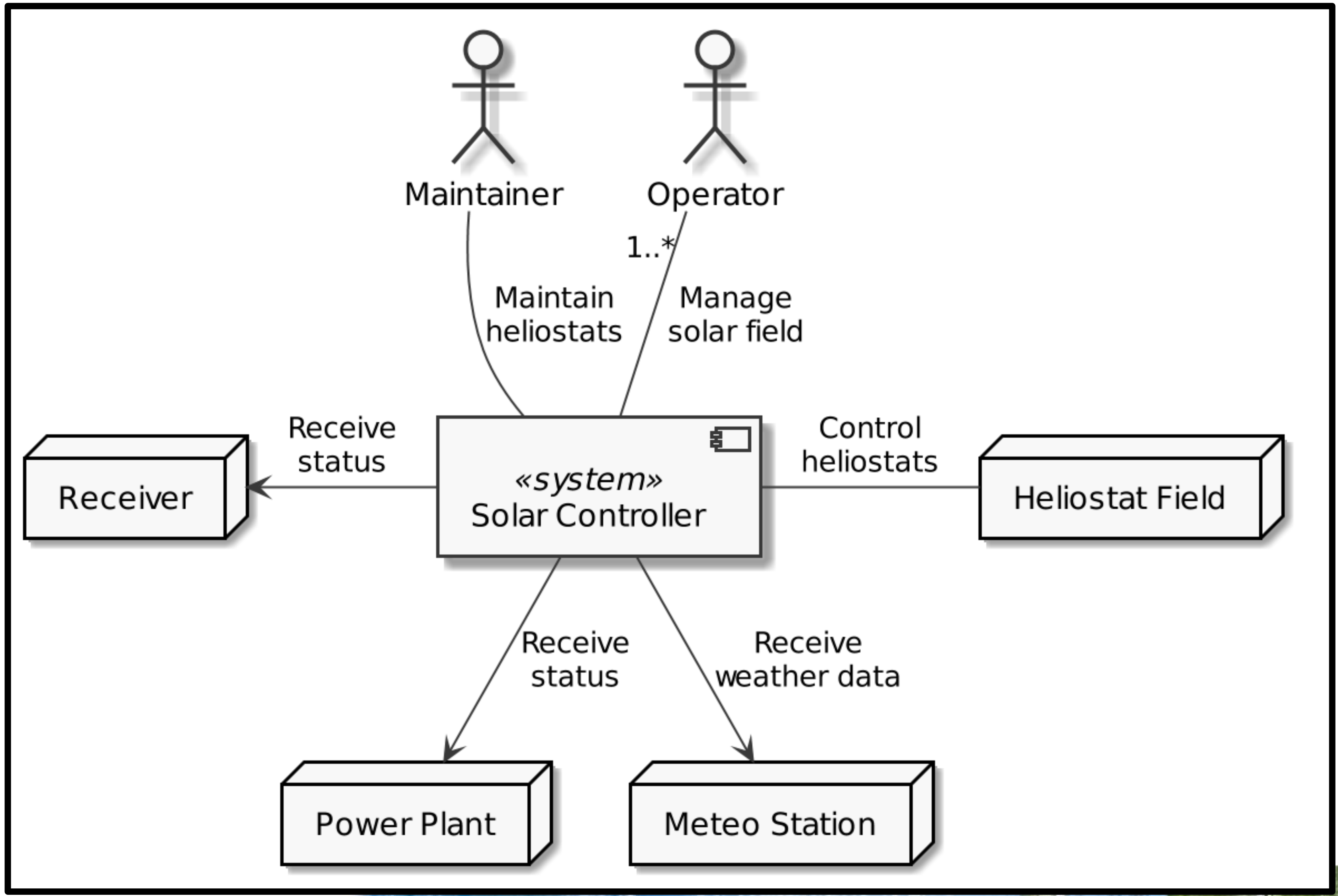
Constraints

Background: Different types of constraints



System context

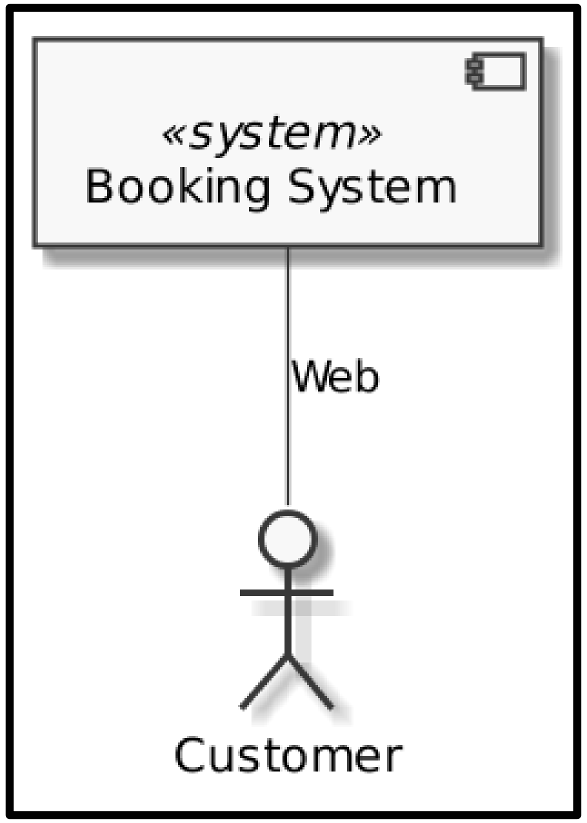
03. Context and Scope



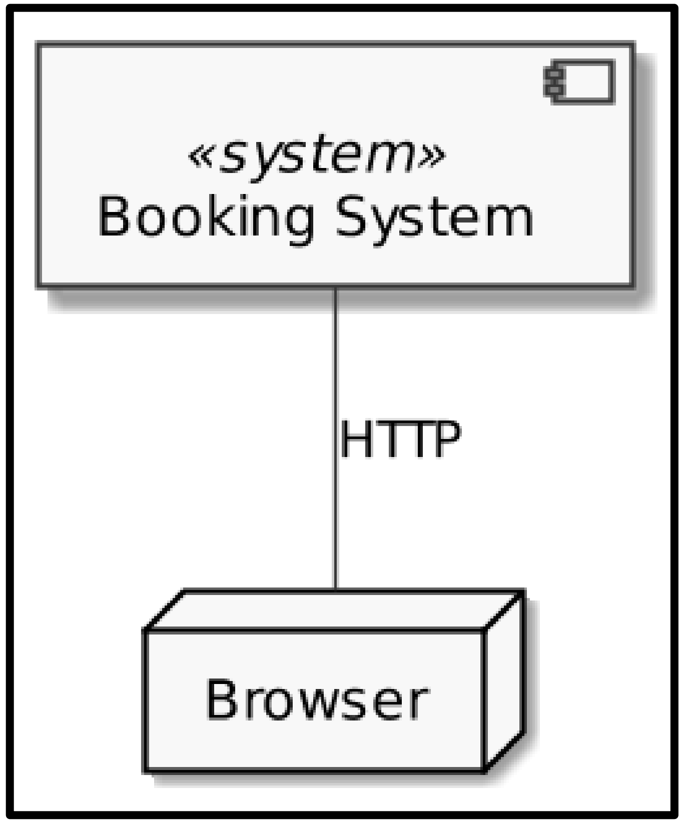
System context

Business vs. technical system context

Business system context

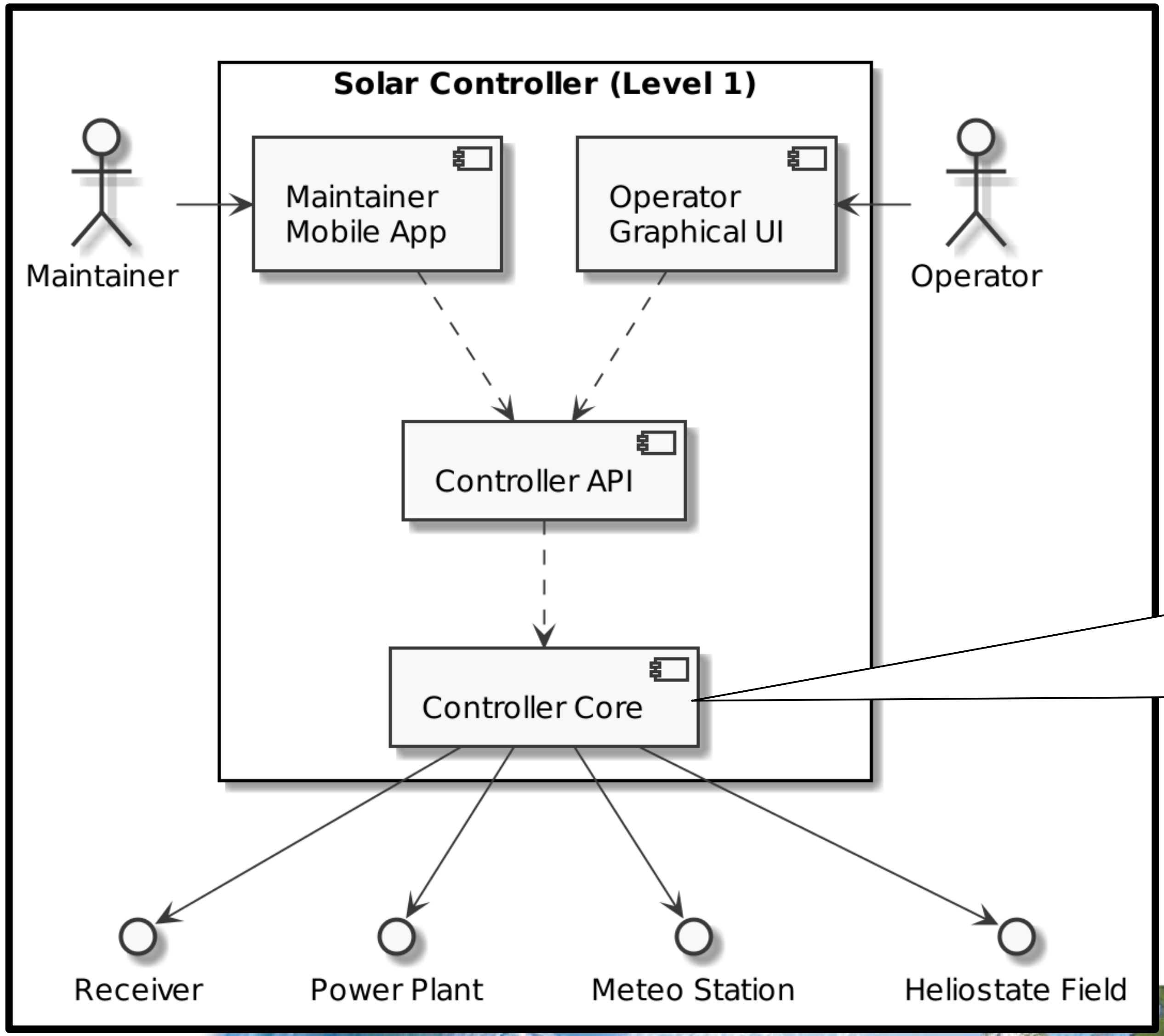


Technical system context



Building block view – level 1

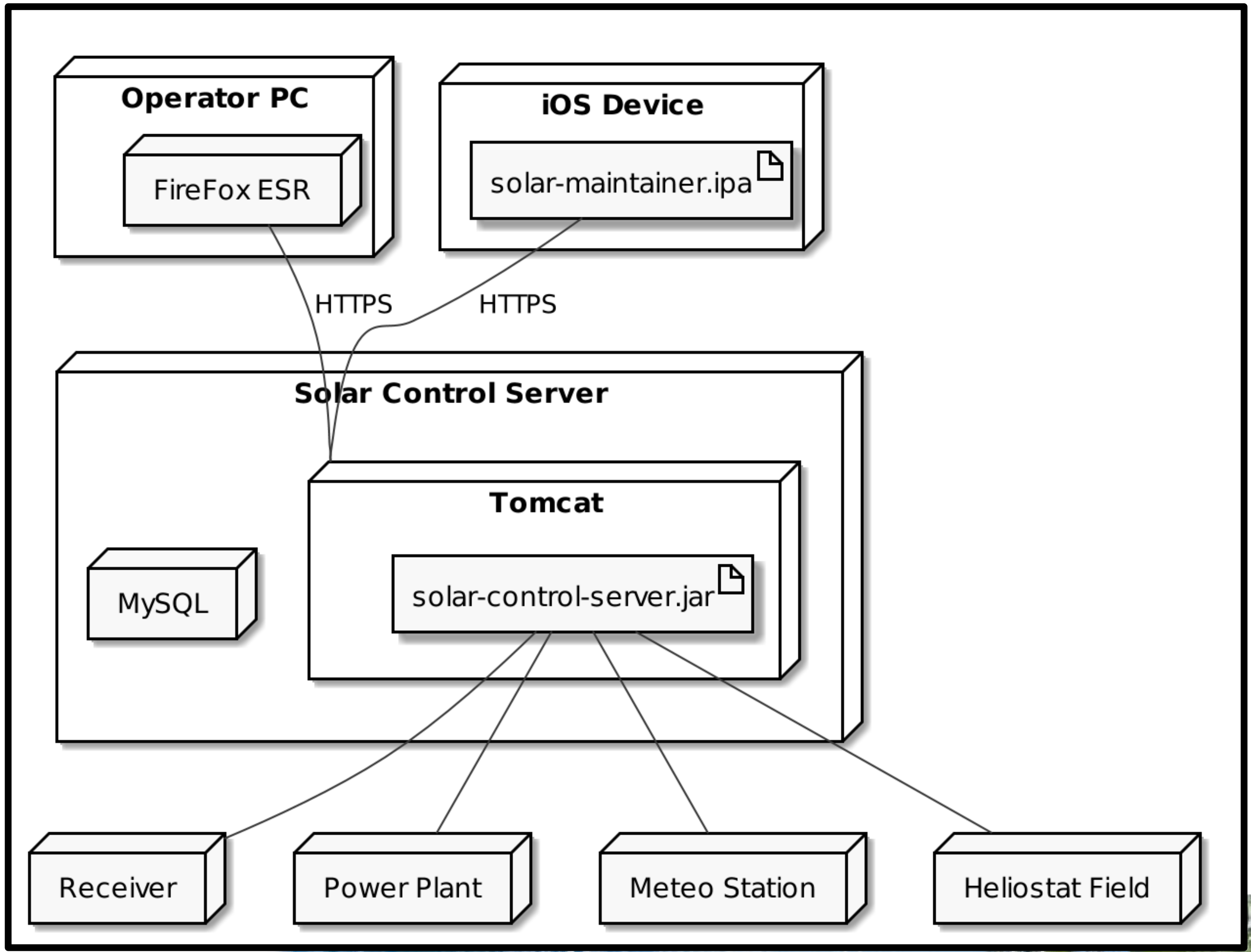
05. Building Block View



You can further decompose the components!

Deployment diagram

07. Deployment View



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.

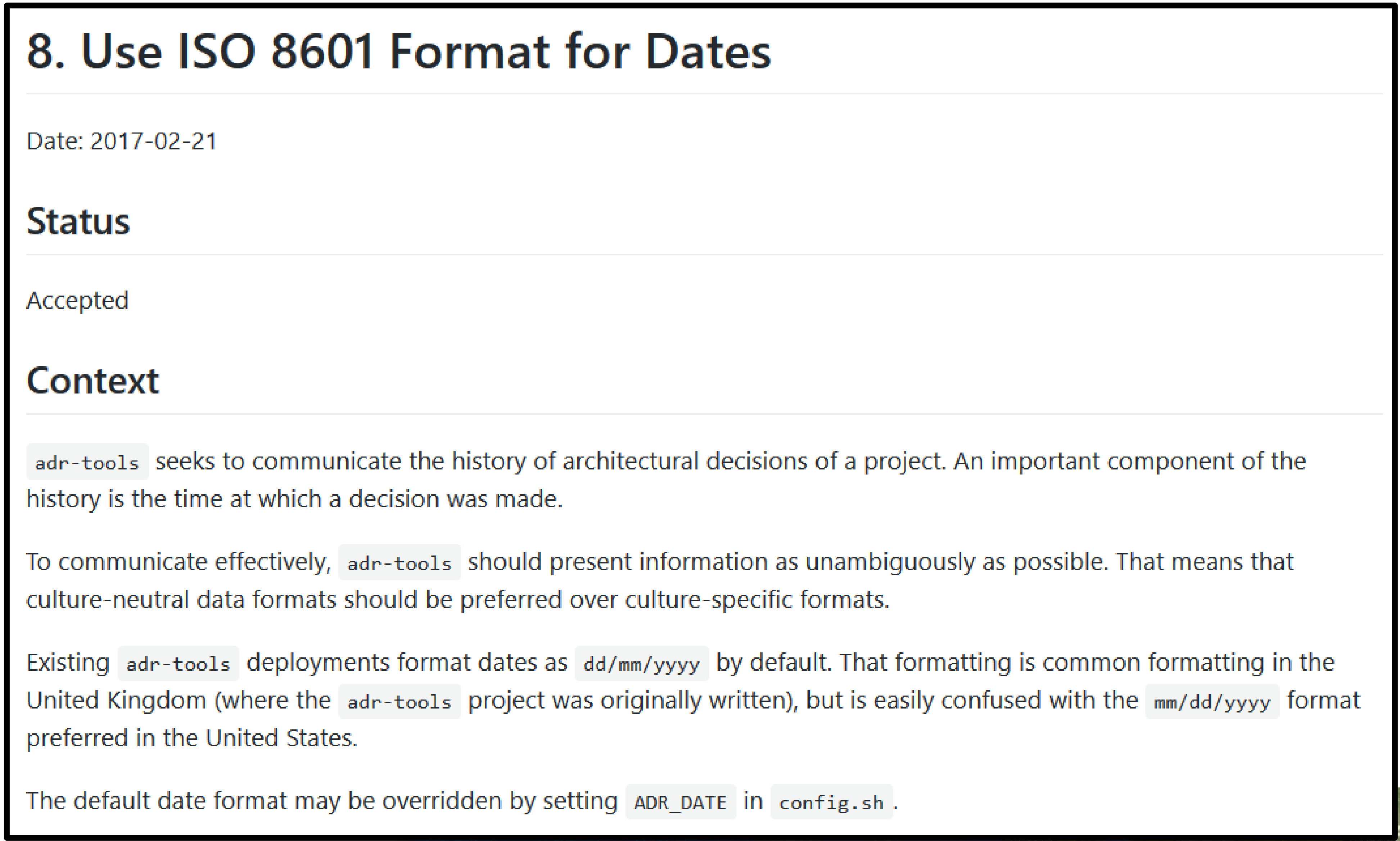
UML in the arc42 template

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

C4 model could be a good alternative or complement!

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.

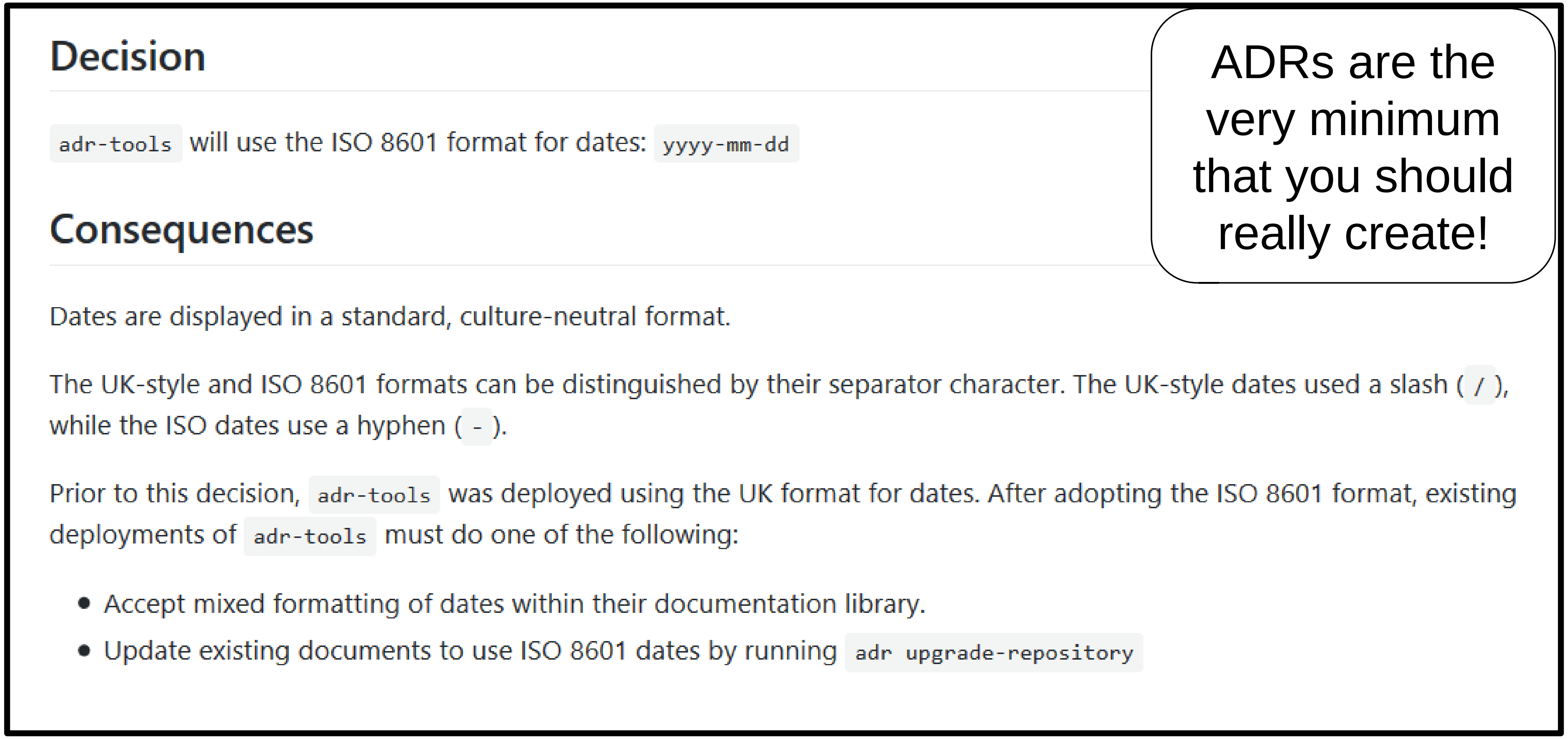
To communicate effectively, `adr-tools` should present information as unambiguously as possible. That means that culture-neutral data formats should be preferred over culture-specific formats.

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`.

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

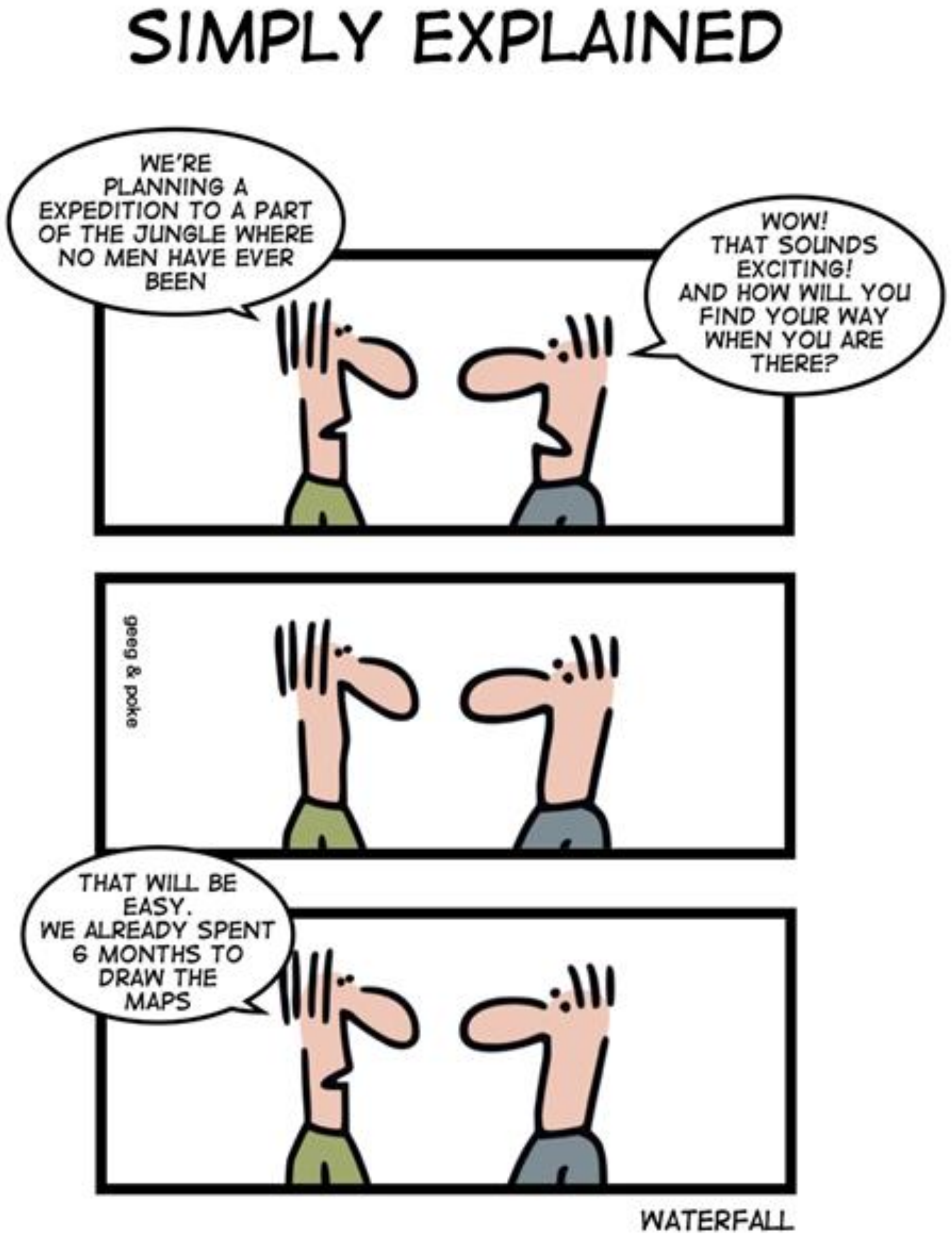
Software architecture documentation in the development process

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?
geeg & poke
THAT WILL BE EASY. WE ALREADY SPENT 6 MONTHS TO DRAW THE MAPS
WATERFALL

Architectural vision

What?

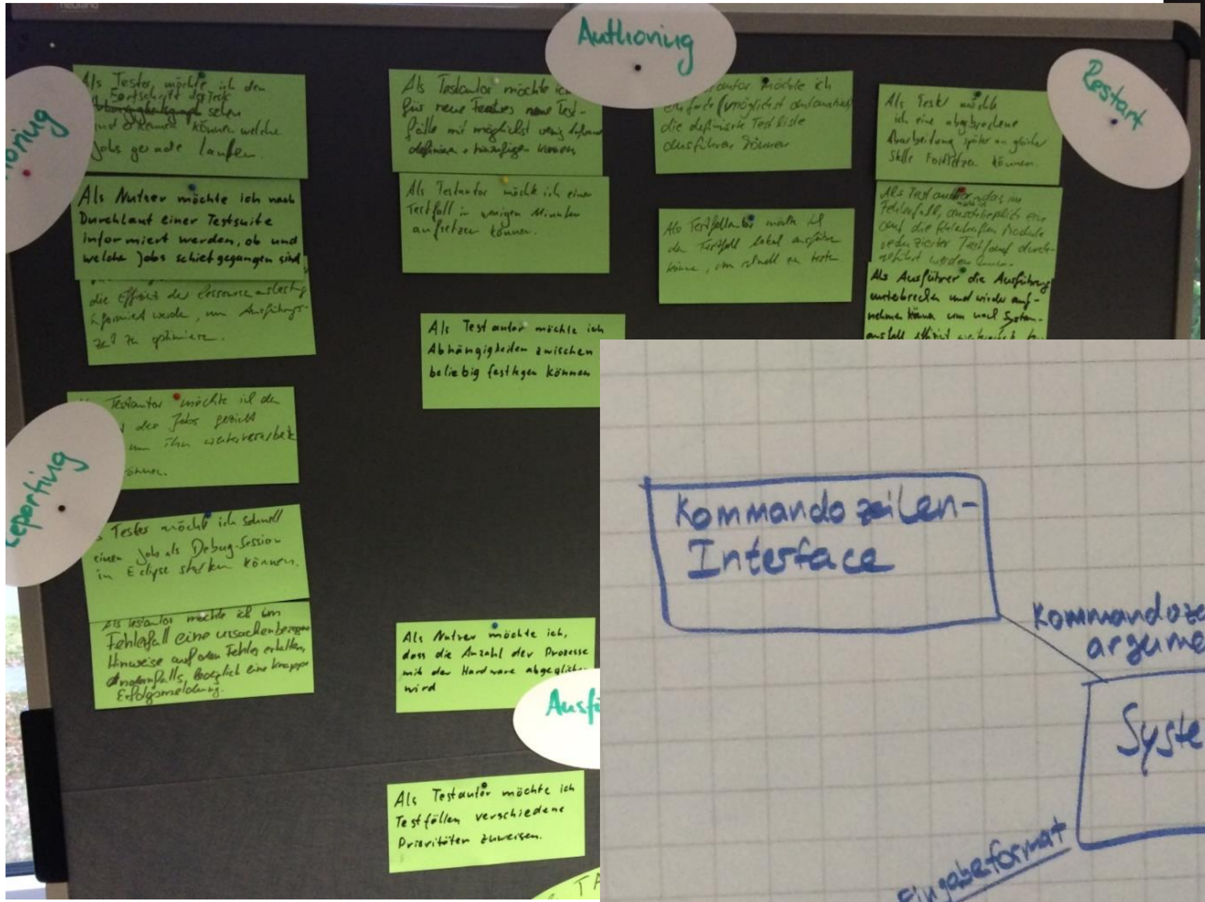
- ✓System context
- ✓Constraints
- ✓Quality goals
- –Quality Scenarios
- –Risks

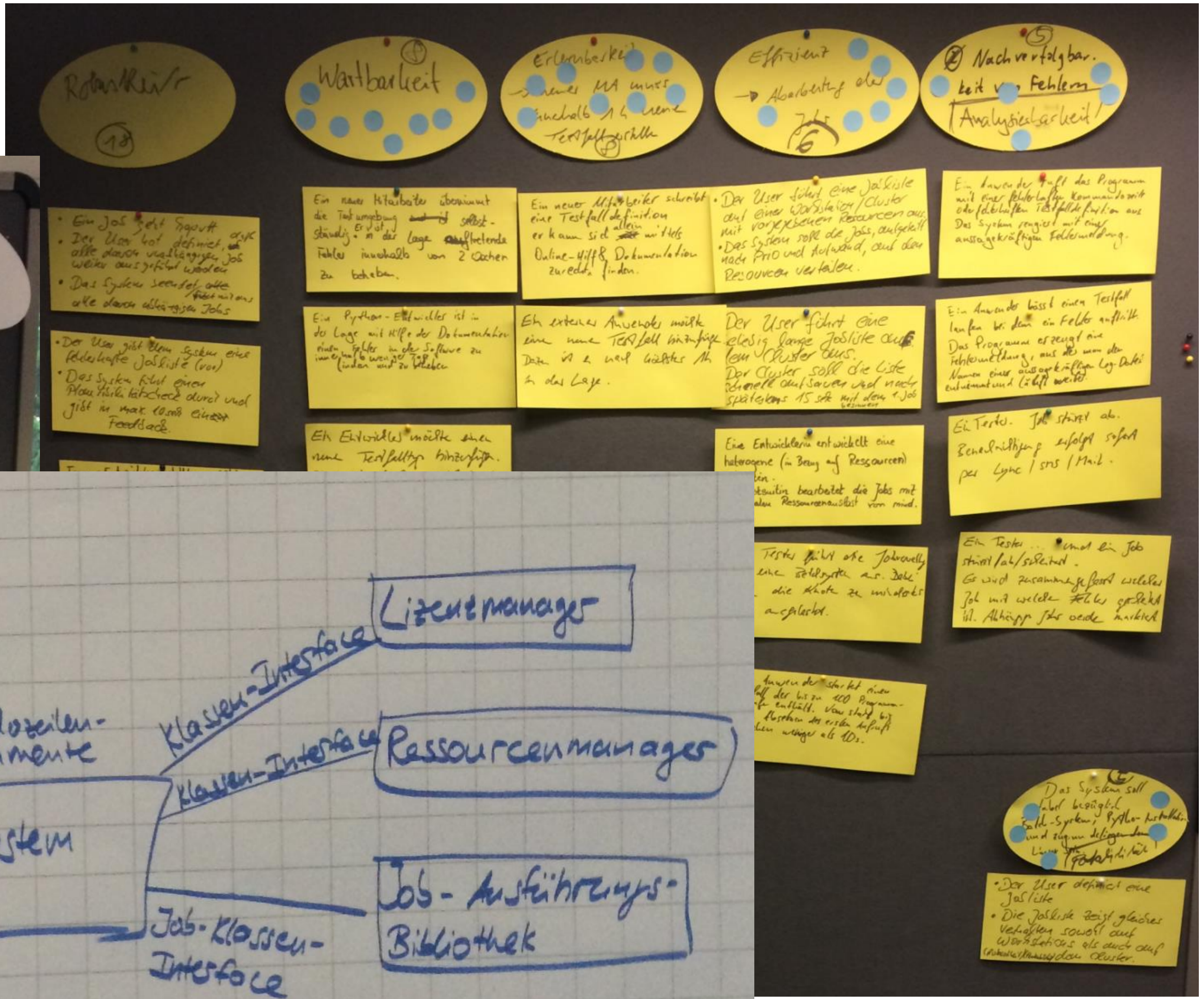
How?

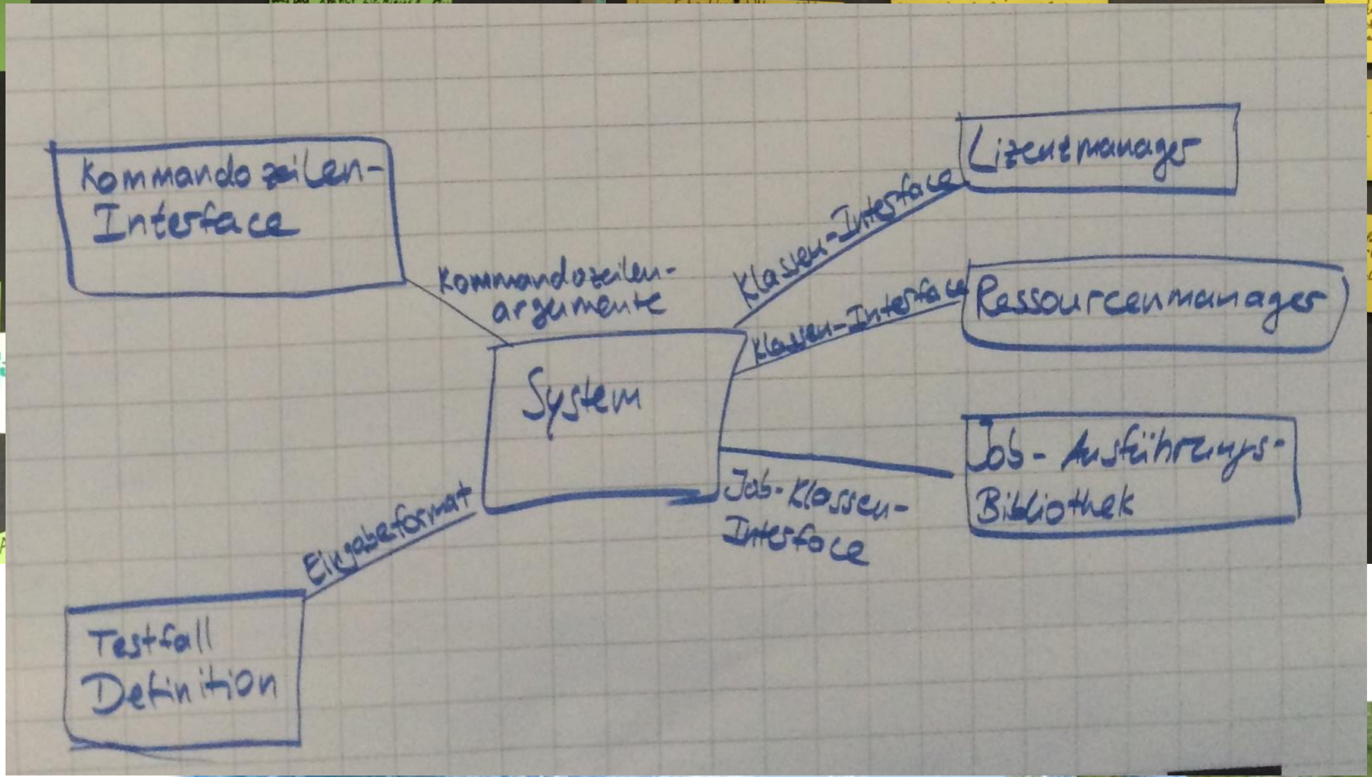
- ✓Technology stack
- ✓Architectural style
- ✓Design principles
- –Building block level 1
- –Domain model

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

Interactive workshops


Authoring
Restart
Reporting
Als Nutzer möchte ich nach Durchlauf einer Testsuite informiert werden, ob und welche Jobs schiefgegangen sind
Als Testautor möchte ich Abhängigkeiten zwischen beliebig festlegen können
Als Testautor möchte ich Testfällen verschiedene Prioritäten zuweisen.
Als Nutzer möchte ich, dass die Anzahl der Prozesse mit der Hardware abgeglichen wird



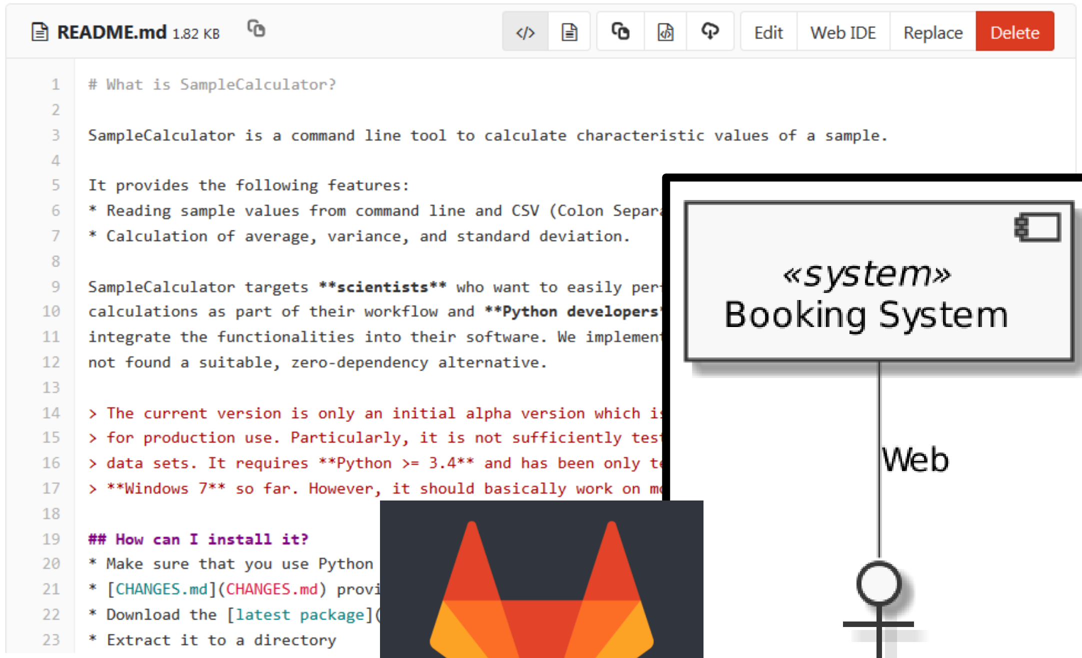


Documentation as code

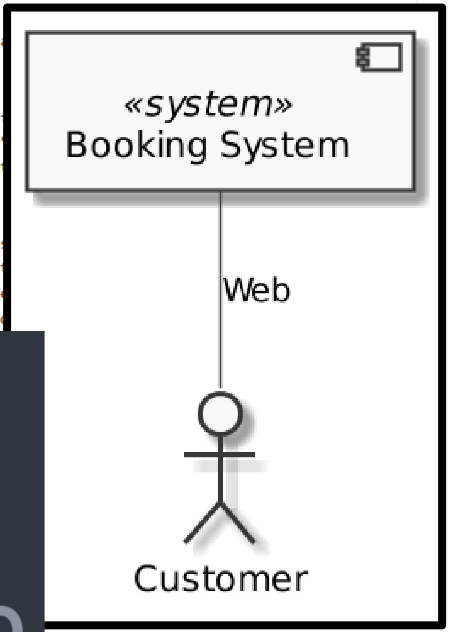
Basic approach:

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

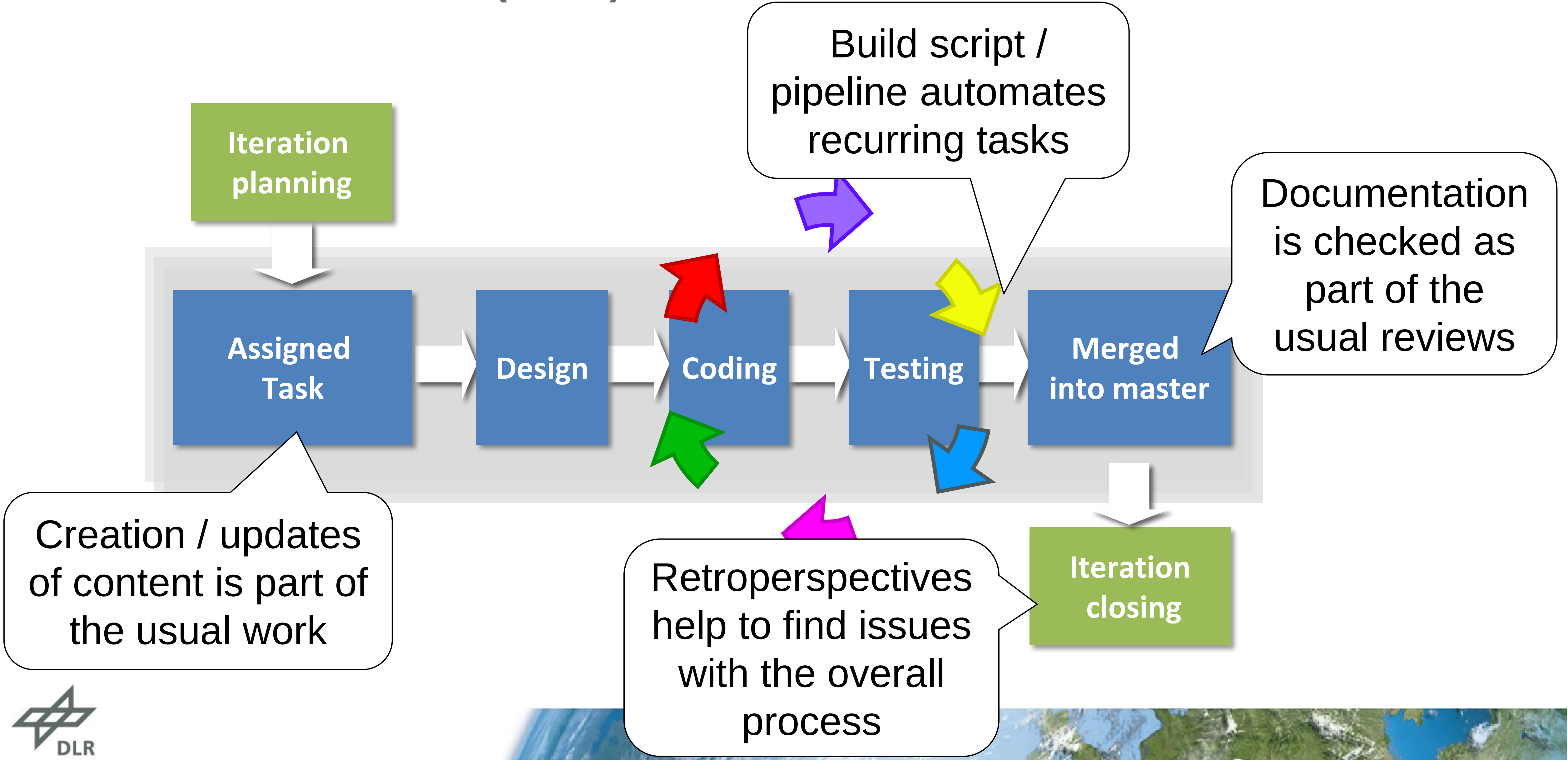
Handle documentation content like your code!



```
# 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
```



Documentation as code (Cont.)


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

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