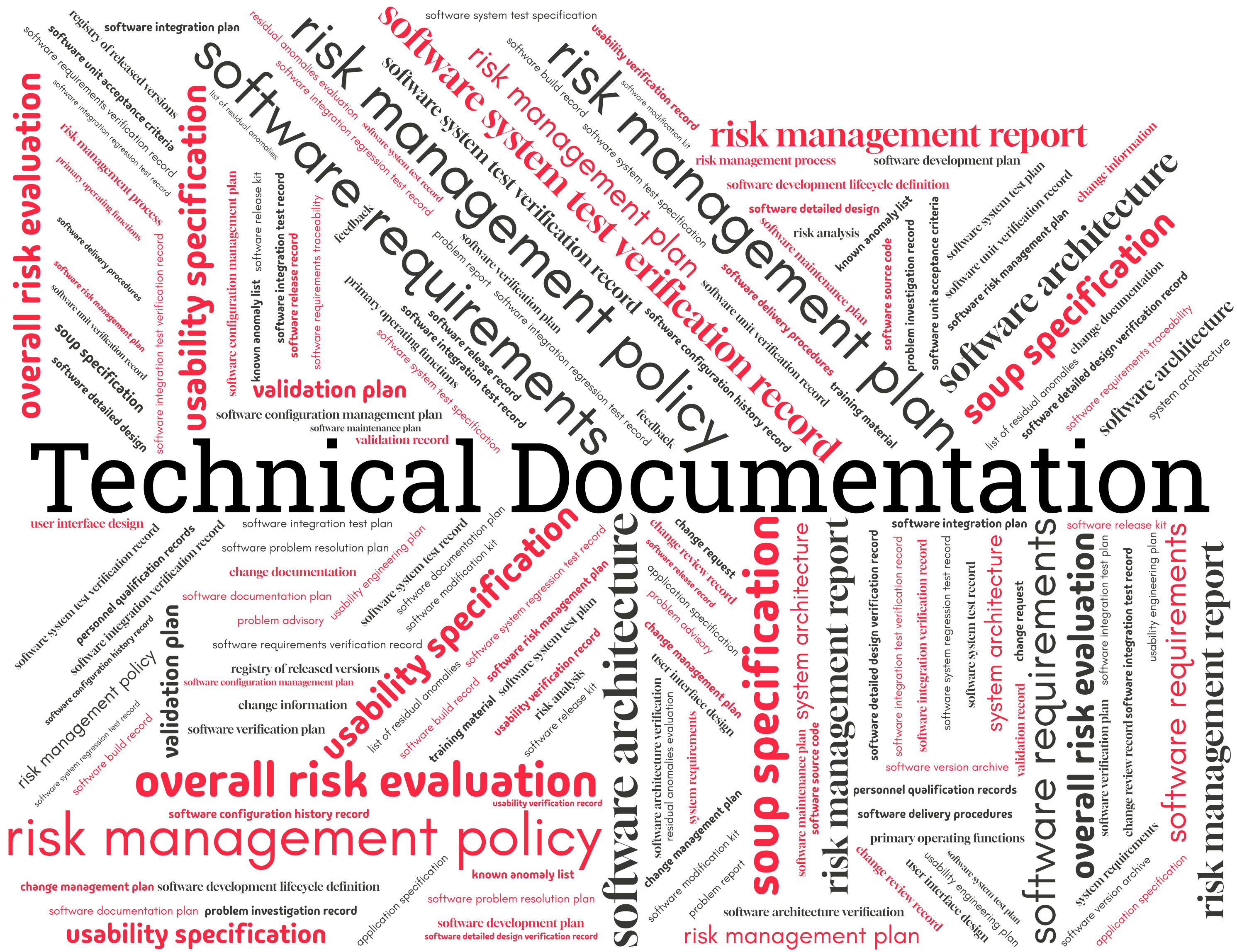
Pragmatisch Dokumentieren

Entwicklungsergebnisse effizient festhalten

Matthias Hölzer-Klüpfel

Dokumentationsanforderungen



Die richtigen Werkzeuge

Projektalltag

Medienbrüche

SW-Anforderungen

SW-Systemprüfung

SW-Architektur

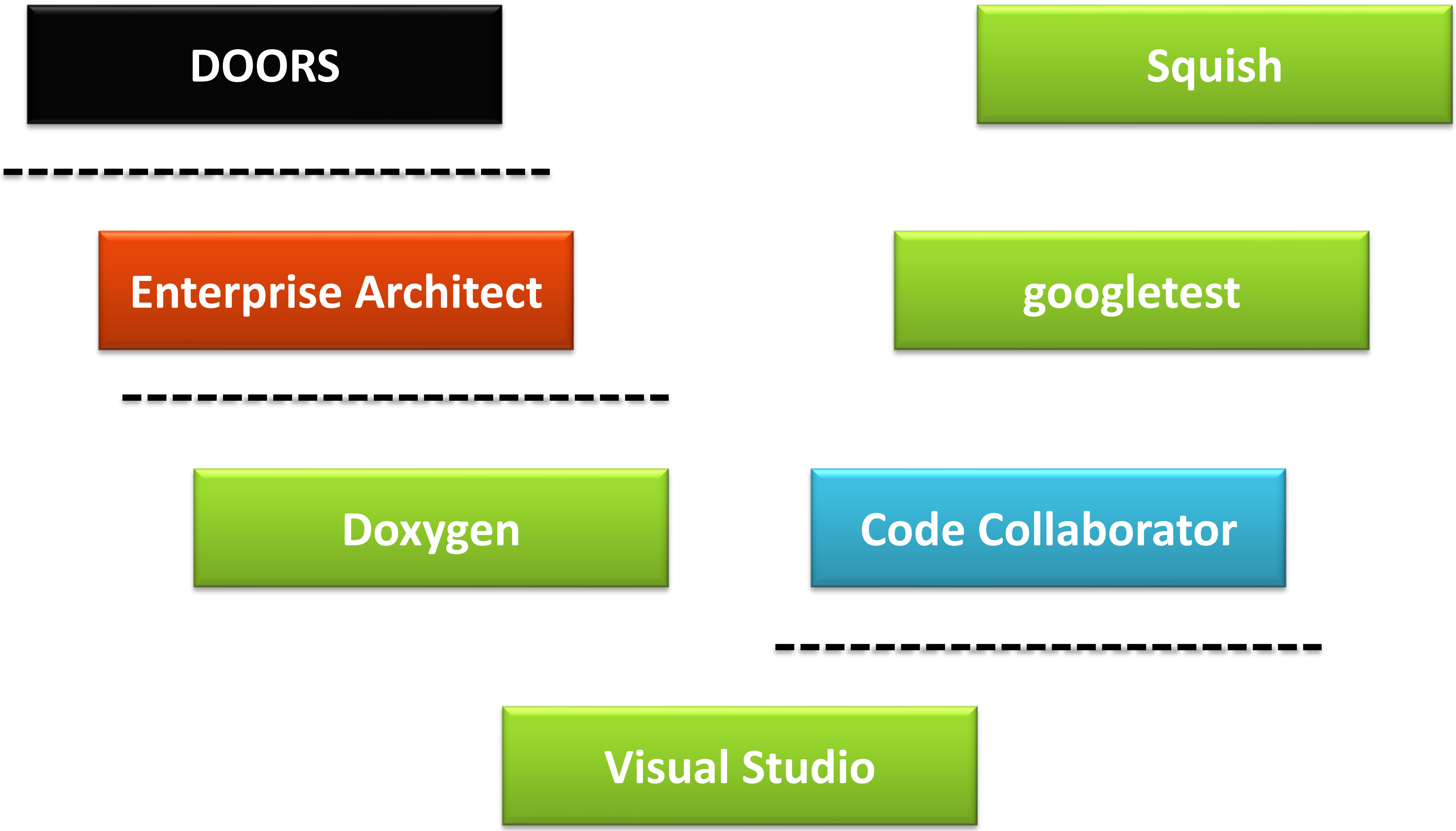
SW-Integrationstest

Detailliertes Design

Verifikation d. Impl.

Implementierung

Medienbrüche



Vergleich Code ↔ Dokumentation

Code	**Dokumentation**
leicht zu ändern	leicht zu ändern
kurze Iterationen	lange Revisionen
erläuternde Kommentare	steht für sich selbst
automatisierter Test	manuelles Review
zentrale IDE	vielfältige Tools
eindeutige Quelle der Wahrheit	entspricht nur mit viel Mühe der Realität

Warum nicht so?

- IDE
- Squish
- PlantUML
- behave
- Doxygen
- googletest
- Visual Studio

Entwicklungswerkzeuge nutzen!

- Möglichst auf Office-Dokumente verzichten
- Offene Formate verwenden
- Dokumentation direkt in der IDE erstellen
- Ideal: „ausführbare“ Dokumentation

Dokumentation als Code

Textbasierte Dokumentation

Artefakt	Beispiele
Requirements	Markdown, LaTeX, ASN
Architektur	PlantUML, ADSL
Design	Doxygen
Code	☺
Buildsystem	Dockerfile, Jenkinsfile, azure-pipelines.xml, bitbucket-pipelines.xml
Unittests	googletest et. al.
Systemtests	behave, gherkin

Meist das
größte
Problem…

Arten von Dokumentation

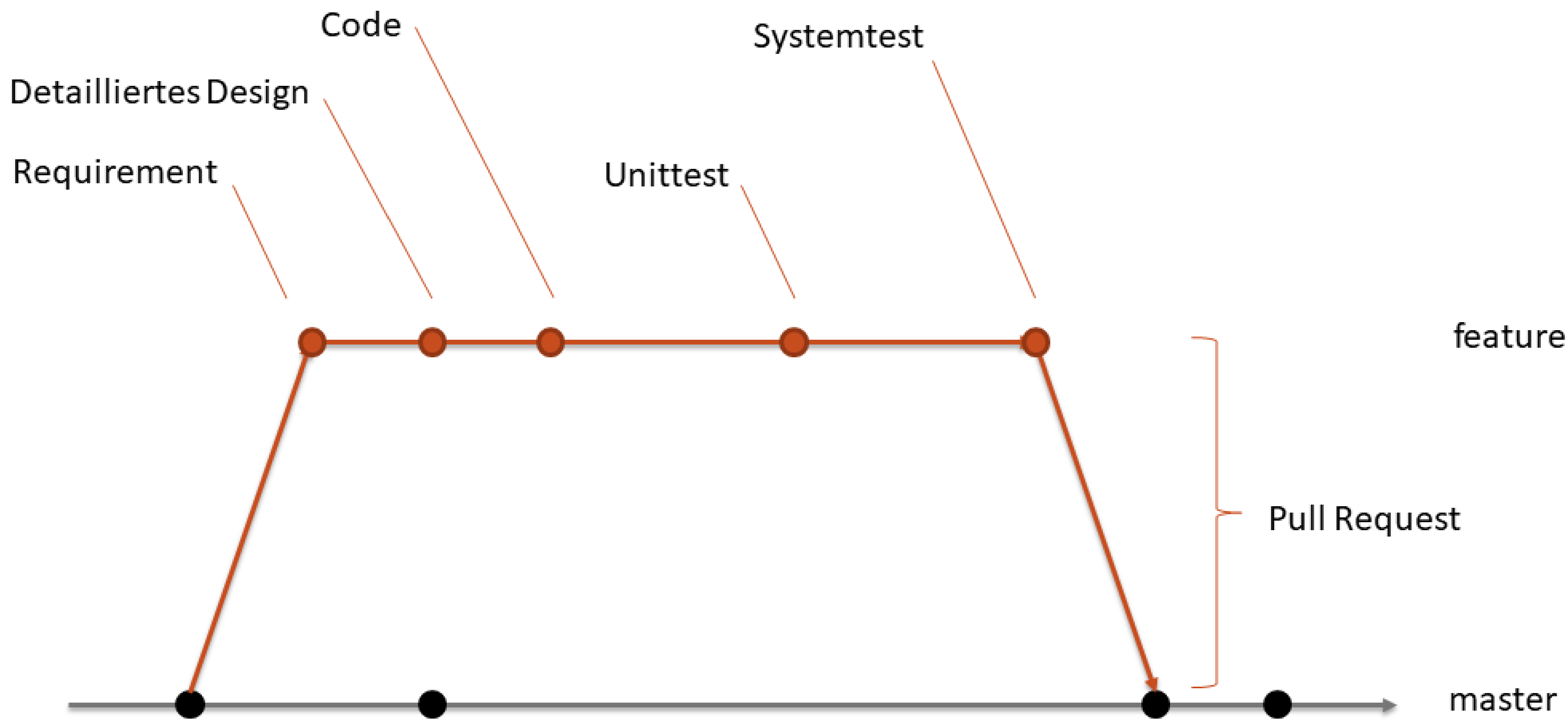
Dokumente

- unterliegen einem Lebenszyklus (Prüfung, Freigabe, Aktualisierung, ...)
- verfügen über eine Version(snummer)

Aufzeichnungen

- belegen die Konformität mit dem Qualitätsmanagement-System
- werden erstellt und aufbewahrt

Pull Request



Pull Request Prüfungen

Vor dem mergen des Pull Requests wird geprüft:

- Kompilierung
- Statische Code-Analyse
- Unit-Tests
- (automatisierte) Software-Systemtests
- Code-Review

- Dokumentengenerierung
- Traceability

Prüfung der Dokumentationsinhalte im Rahmen des Approvals der Pull Requests.

Dokumentation wie Code behandeln!

- Dokumentation und Aufzeichnungen trennen
- Klartext-Formate für Dokumentation verwenden
- Unter gemeinsame Versionskontrolle stellen
- Statische Analyse automatisiert durchführen
- Reviews im Rahmen von Pullrequests

Exkurs: klartext

Requirements in Markdown? Ernsthaft?

Markup-Sprachen sind:

- gut für (nahezu) unstrukturierte Inhalte wie Freitext
- problematisch für definierte Schemata

Datenbanken sind:

- gut für fix strukturierte Daten
- problematisch für Freitext

Technische Dokumentation ist:

- zum Teil strukturiert mit unterschiedlichen Schemata
- zum Teil Freitext

Wunschliste

Ideal wäre ein System, das

- flexible Möglichkeiten zur Strukturierung anbietet
- einfache Integration von Freitext erlaubt
- einfach zu erstellen und zu verarbeiten ist
- Inhalte von der Darstellung trennt
- unterschiedliche Ausgabeformate zulässt

Also irgendwie eine Mischung aus

- XML
- Markdown
- Python

Ausgangspunkt

```xml
<task id="task-define-sdlc" name="Define Software Development Life Cycle" scope="project-start">
    <description>
        <xhtml:p>
            Define the <g ref="glossary-sdlc">Software Development Life Cycle</g> for
            the development project.
        </xhtml:p>
        <xhtml:p>
            This includes:
        </xhtml:p>
        <xhtml:ul>
            <xhtml:li>
                referencing <g ref="glossary-SOP">standard operating procedures</g>
                relevant to the development project
            </xhtml:li>
            <xhtml:li>
                defining <g ref="glossary-activity">activities</g> and <g ref="task">tasks</g>
                to be performed
            </xhtml:li>
            <xhtml:li>
                documenting and justifying any deviations from the
                <g ref="glossary-SOP">standard operating procedures</g> (i.e. process tailoring)
            </xhtml:li>
        </xhtml:ul>
    </description>
    <responsible ref="project-manager"/>
    <output ref="software-development-plan"/>
</task>
```

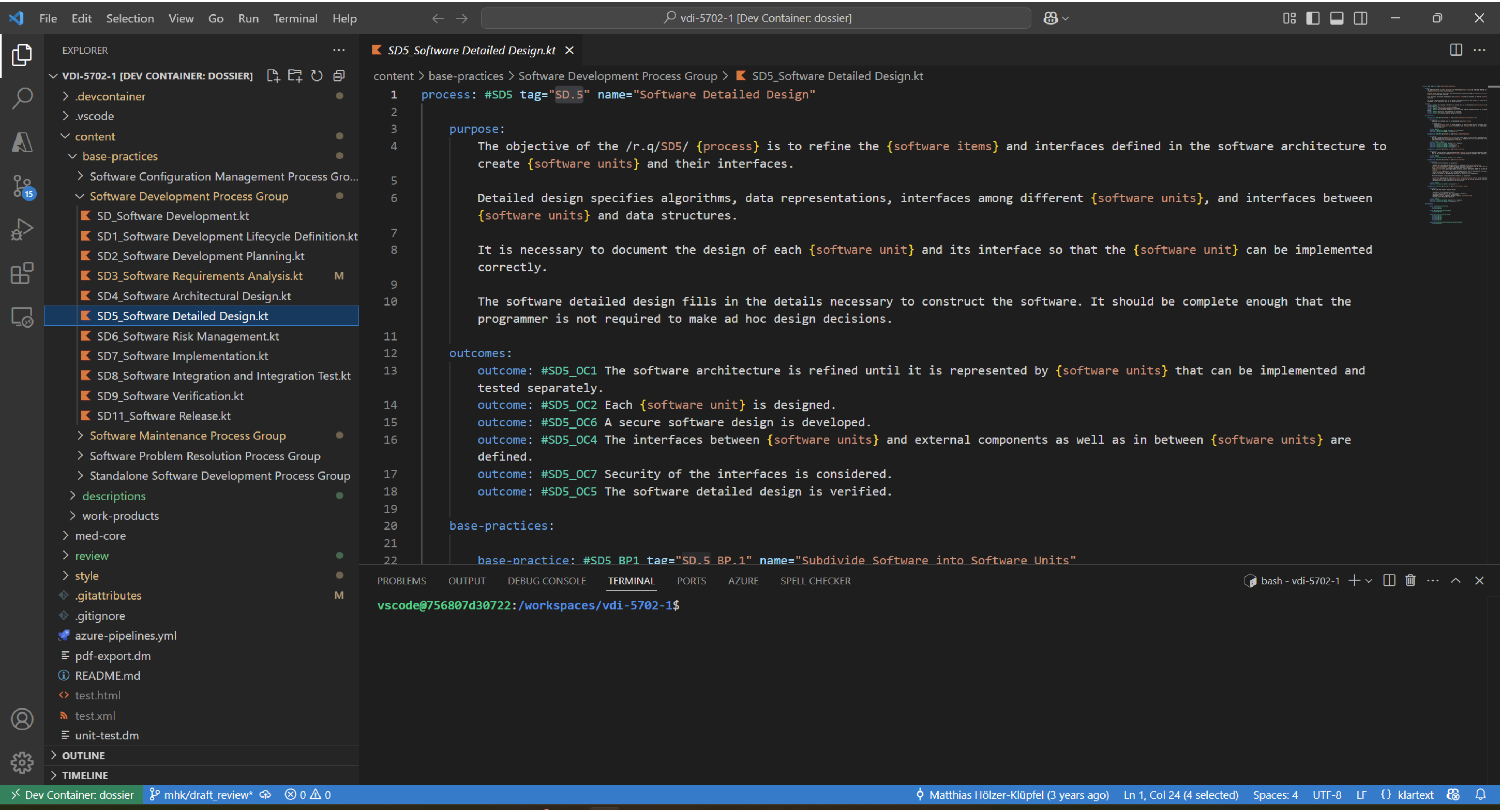
klartext

```
// This is a task definition

task: #task-define-sdlc name="Define Software Development Life Cycle" scope="project-start"

    description:
        Define the {software development life cycle} for the development project.

        This includes:

        - **referencing** {standard operating procedures} relevant to the development project
        - **defining** {activities} and {tasks} to be performed
        - **documenting** and justifying any deviations from the {standard operating procedures}
          (i.e. process tailoring)

    responsible> project-manager
    output> software-development-plan
```

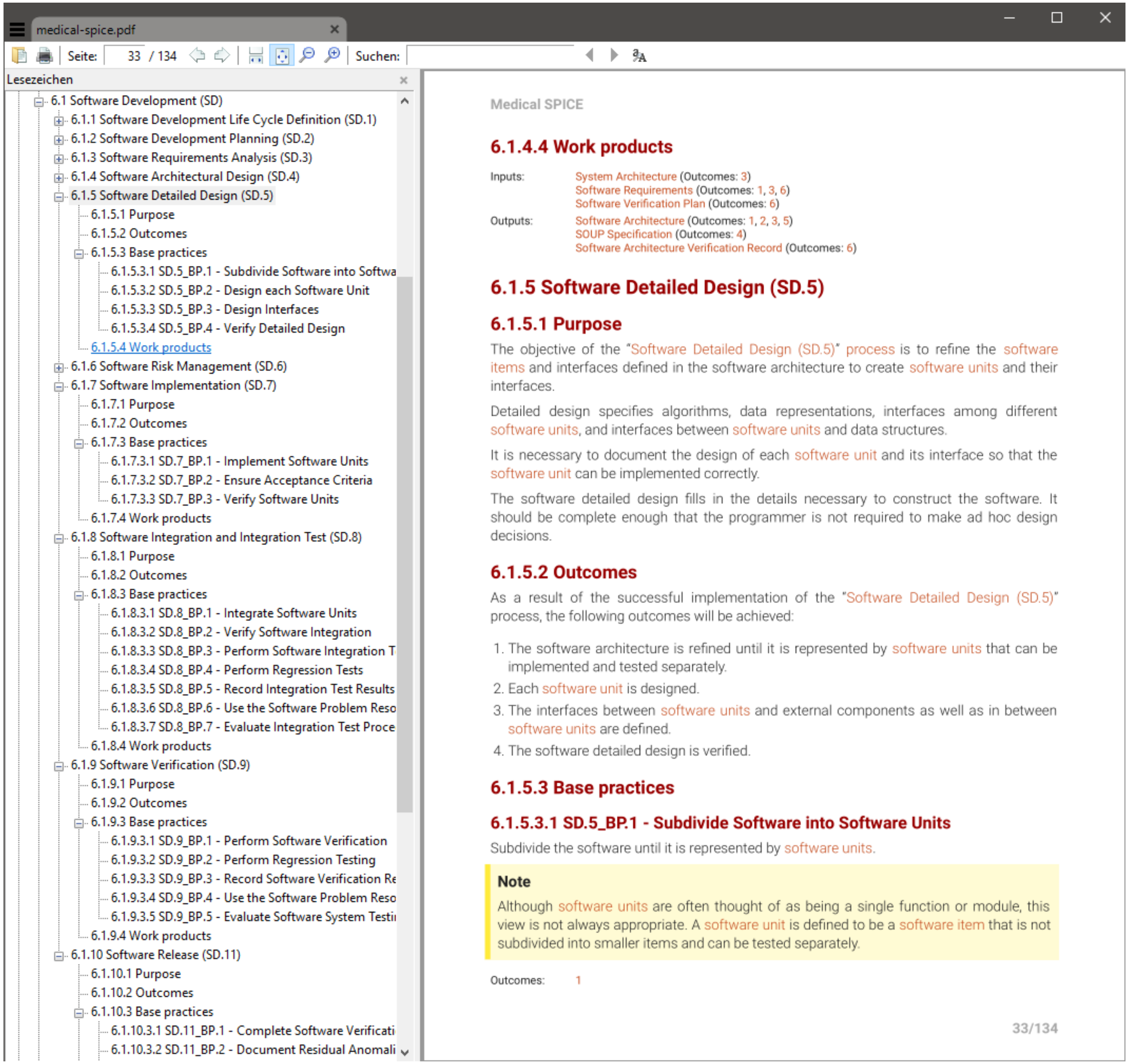
Beispiel: Medical SPICE

```
process: #SD5 tag="SD.5" name="Software Detailed Design"

    purpose:
        The objective of the /r.q/SD5/ {process} is to refine the {software items} and interfaces defined in
        the software architecture to create {software units} and their interfaces.

        Detailed design specifies algorithms, data representations, interfaces among different
        {software units}, and interfaces between {software units} and data structures.

        It is necessary to document the design of each {software unit} and its interface so that the
        {software unit} can be implemented correctly.

    outcomes:
        outcome: #SD5_OC1 The software architecture is refined until it is represented by {software units}
                 that can be implemented and tested separately.
        outcome: #SD5_OC2 Each {software unit} is designed.
        outcome: #SD5_OC4 The interfaces between {software units} and external components as well as in
                 between {software units} are defined.
        outcome: #SD5_OC5 The software detailed design is verified.

    base-practices:

        base-practice: #SD5_BP1 tag="SD.5_BP.1" name="Subdivide Software into Software Units"

            description:
                Subdivide the software until it is represented by {software units}.

                !!! note "Note"
                    Although {software units} are often thought of as being a single function or module, this
                    view is not always appropriate. A {software unit} is defined to be a {software item} that
                    is not subdivided into smaller items and can be tested separately.

            outcome> SD5_OC1
```

Beispiel: Medical SPICE



Beispiel: Medical SPICE



Medical SPICE

6.1.4.4 Work products

Inputs: System Architecture (Outcomes: 3)
Software Requirements (Outcomes: 1, 3, 6)
Software Verification Plan (Outcomes: 6)

Outputs: Software Architecture (Outcomes: 1, 2, 3, 5)
SOUP Specification (Outcomes: 4)
Software Architecture Verification Record (Outcomes: 6)

6.1.5 Software Detailed Design (SD.5)

6.1.5.1 Purpose

The objective of the "Software Detailed Design (SD.5)" process is to refine the software items and interfaces defined in the software architecture to create software units and their interfaces.

Detailed design specifies algorithms, data representations, interfaces among different software units, and interfaces between software units and data structures.

It is necessary to document the design of each software unit and its interface so that the software unit can be implemented correctly.

The software detailed design fills in the details necessary to construct the software. It should be complete enough that the programmer is not required to make ad hoc design decisions.

6.1.5.2 Outcomes

As a result of the successful implementation of the "Software Detailed Design (SD.5)" process, the following outcomes will be achieved:

1. The software architecture is refined until it is represented by software units that can be implemented and tested separately.
2. Each software unit is designed.
3. The interfaces between software units and external components as well as in between software units are defined.
4. The software detailed design is verified.

6.1.5.3 Base practices

6.1.5.3.1 SD.5_BP.1 - Subdivide Software into Software Units

Subdivide the software until it is represented by software units.

Note

Although software units are often thought of as being a single function or module, this view is not always appropriate. A software unit is defined to be a software item that is not subdivided into smaller items and can be tested separately.

Outcomes: 1

33/134

Im klartext dokumentieren!

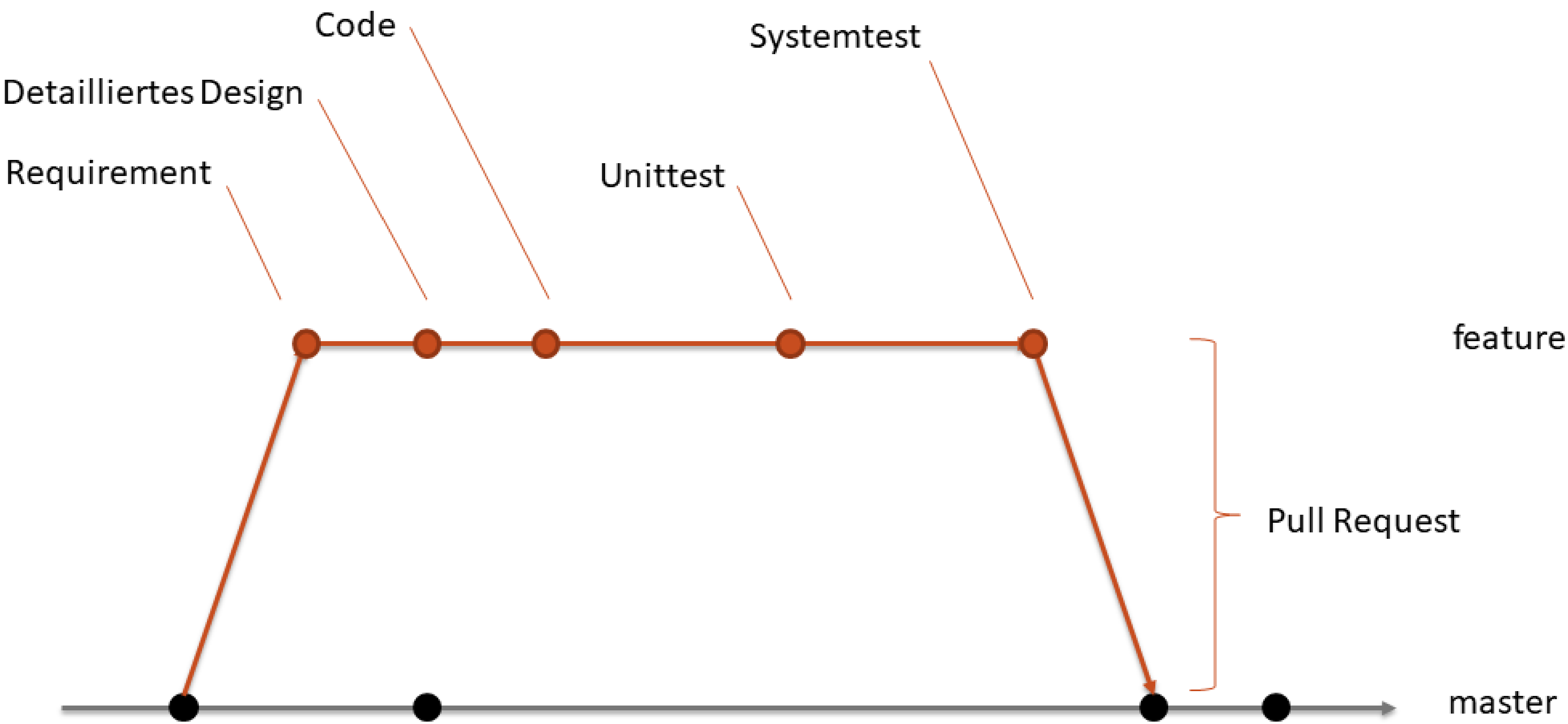
- klartext: eine Markup-Sprache für semi-strukturierte Dokumentation
- open-source & work-in-progress
- Wer mithelfen möchte:

  www.klartext-dossier.org

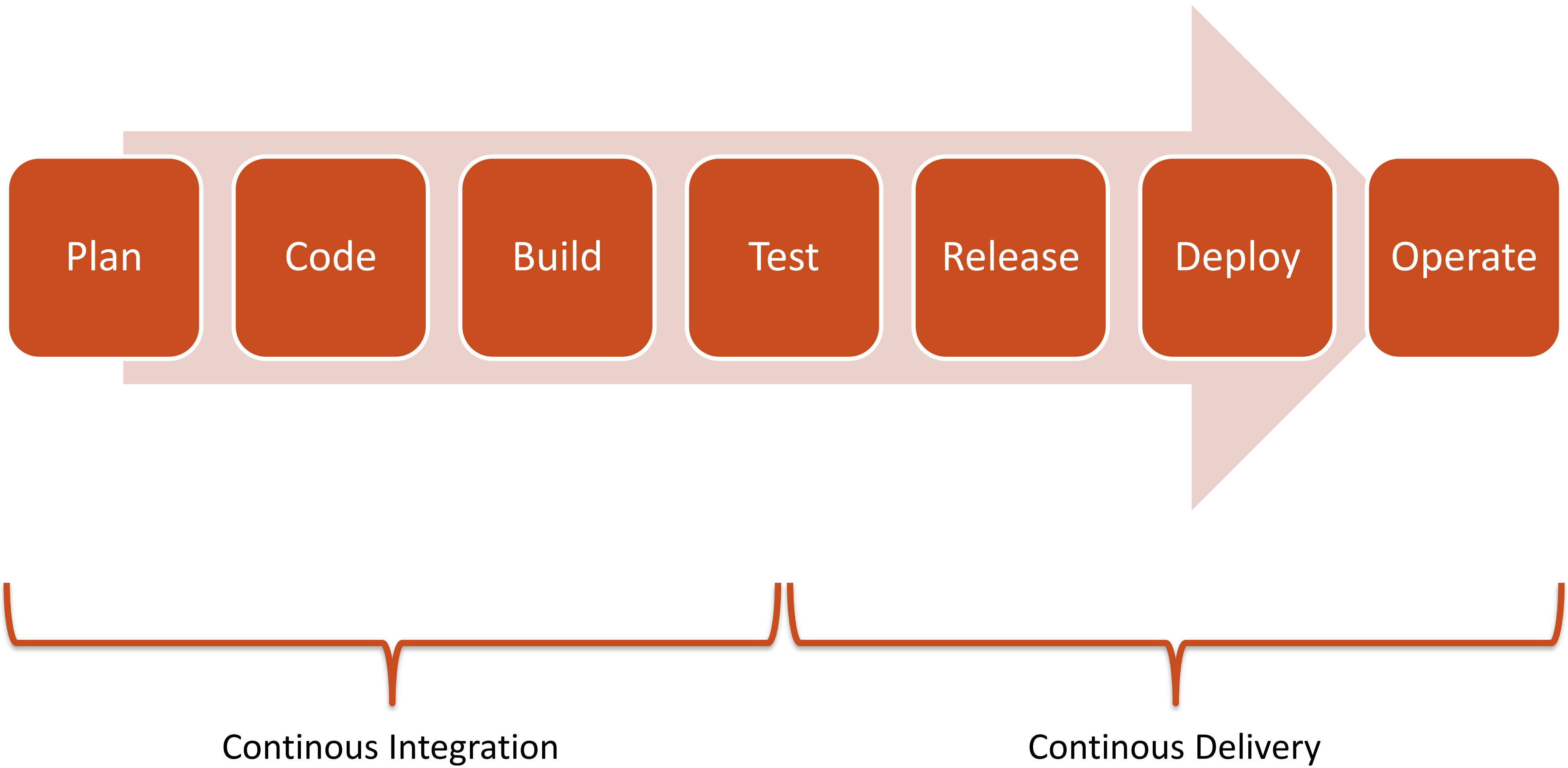
Workflow-Integration

Projektalltag

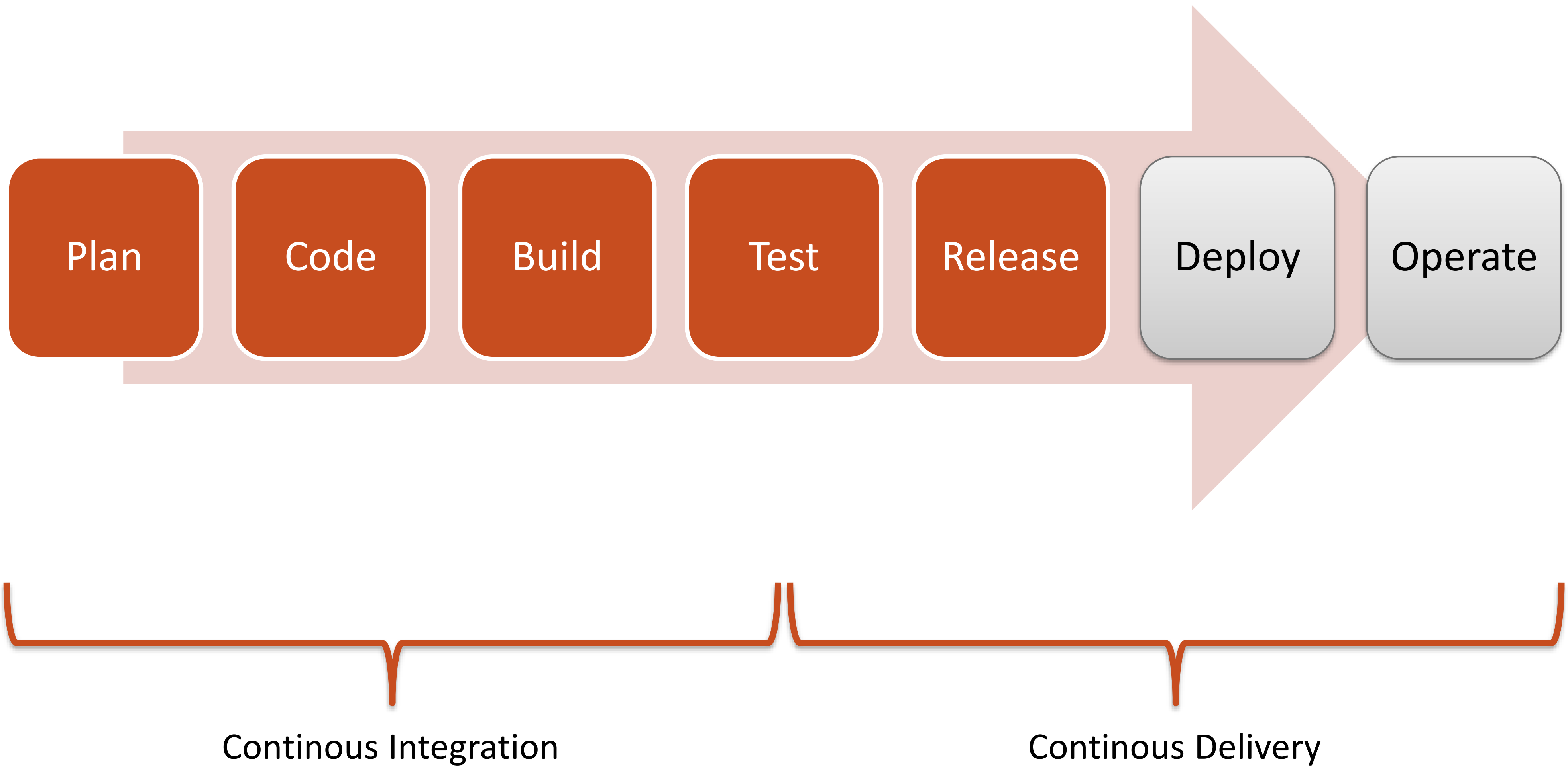
Pull Request



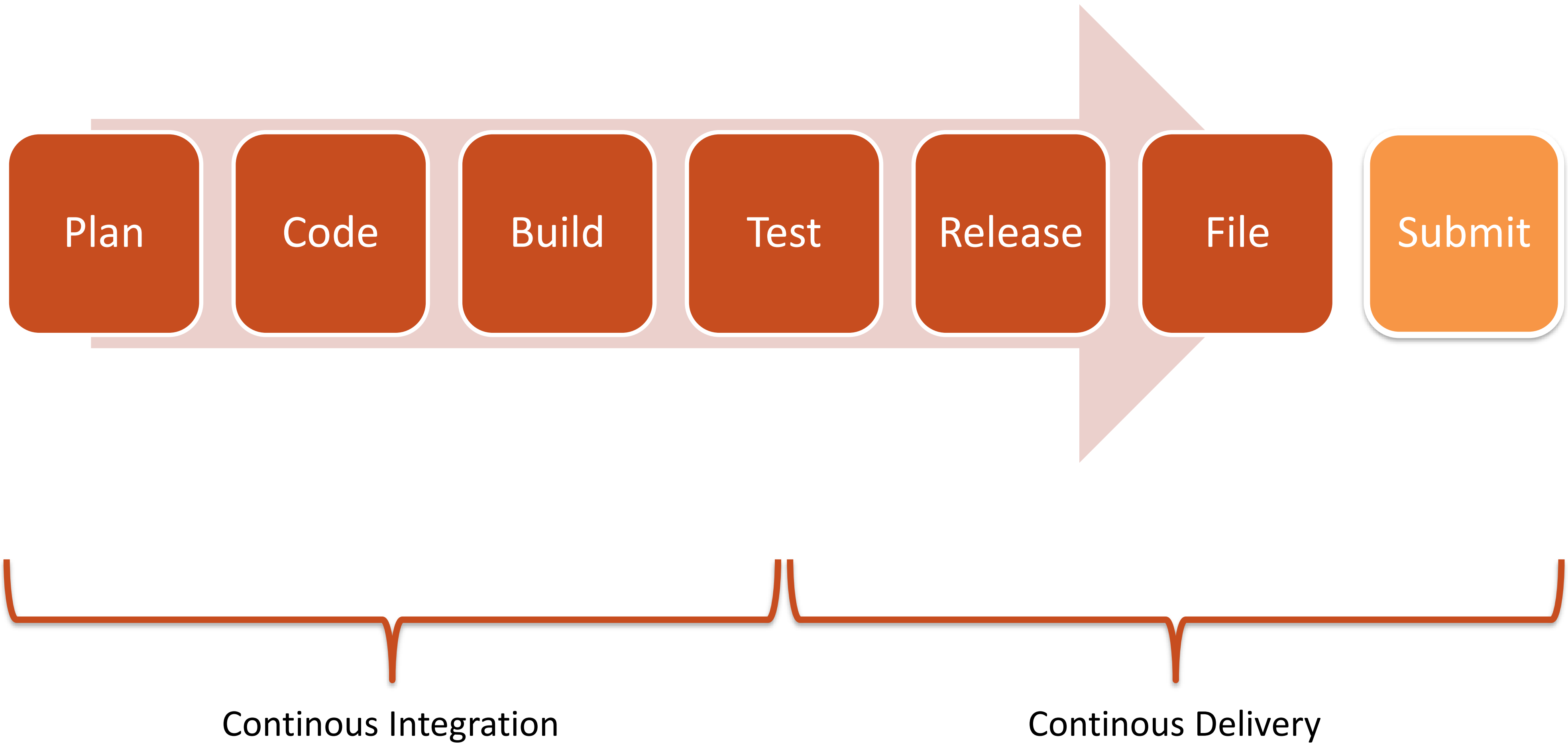
DevOps-Pipeline



DevOps-Pipeline



DevOps-Pipeline



Dokumentation in Workflow integrieren!

- Dokumentation in Pull Requests einschließen
- Automatisierte Prüfungen durchführen
- Nur approven, wenn Code und Dokumentation zusammenpassen
- Dokumentationsschnipsel bei Integration auf Integrationsbranch prüfen
- Dokumente bei Integration auf Releasebranch prüfen

Dokumentengenerierung

Beispiel: Testskript

Beispiel.xml

```xml
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="test-script.xsl"?>

<testscript revision="$Revision: 123 $" version="1.0.0" xmlns="http://www.hoelzerkluepfel.de/testscript">

  <sequence id="TEST1" name="Alarmfunktion">

    <description>
      Dieser Test überprüft die Funktion des visuellen Alarms.
    </description>

    <repeat count="5">
      <wait start="490ms" timeout="510ms">
        <check-transition name="Alarm LED" from="Off" to="On"/>
      </wait>
      <wait start="490ms" timeout="510ms">
        <check-transition name="Alarm LED" from="On" to="Off"/>
      </wait>
    </repeat>

    <verify requirement="REQ1234"/>

  </sequence>

</testscript>
```

REQ1234: Test der Alarm LED

Nach dem POST blinkt die Alarm LED
5 mal mit 1Hz und 50% Einschaltdauer.

Testspezifikation

1 TEST1 - Alarmfunktion

1.1 Introduction

Dieser Test überprüft die Funktion des visuellen Alarms.

1.2 Test Steps

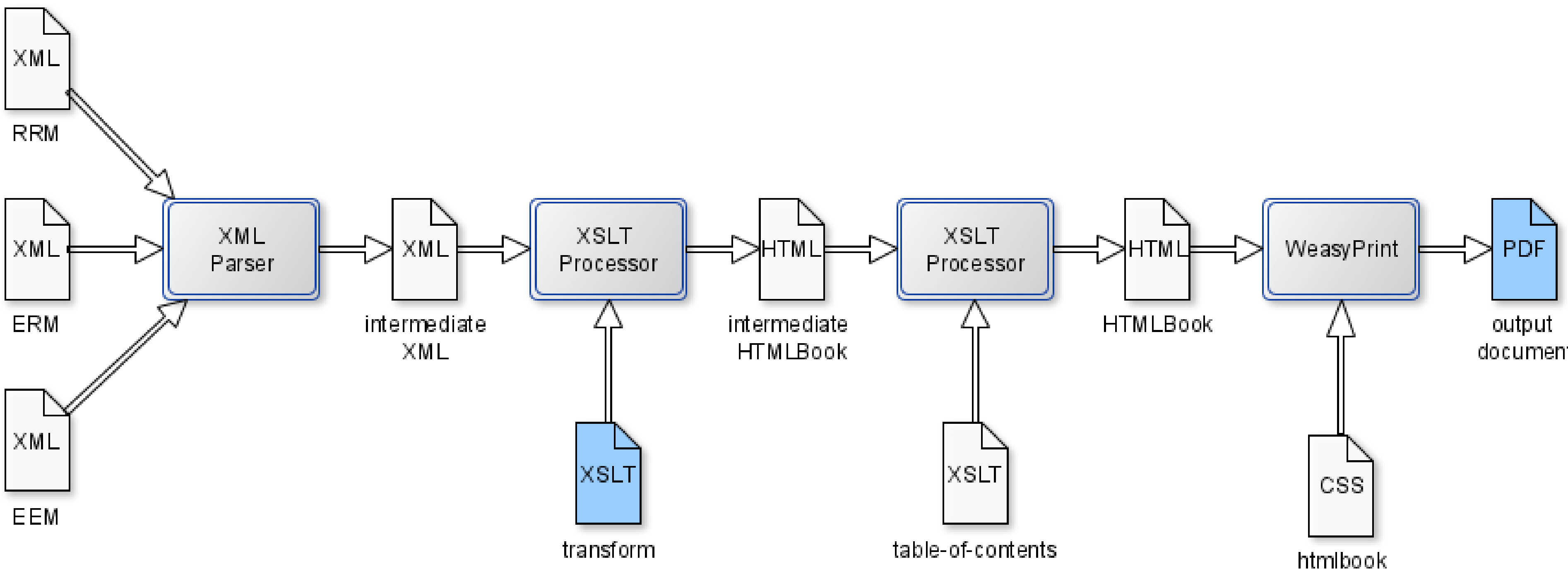
<table>
<tr><th>Step</th><th>Action</th><th>Expected Result</th></tr>
<tr><td>#1</td><td>Repeat the following steps for 5 times:
<table>
<tr><th>Step</th><th>Action</th><th>Expected Result</th></tr>
<tr><td>#1</td><td>Wait for the following condition/s to become true:
The 'Alarm LED' transitions from 'Off' to 'On'</td><td>Condition true in less than 510ms but not before 490ms</td></tr>
<tr><td>#2</td><td>Wait for the following condition/s to become true:
The 'Alarm LED' transitions from 'On' to 'Off'</td><td>Condition true in less than 510ms but not before 490ms</td></tr>
</table></td><td>All repeated commands pass</td></tr>
<tr><td colspan="3">Finished verification for requirement REQ1234</td></tr>
</table>

1.3 Traces to Requirements

This test script will verify the following requirements:

- REQ1234

Dokumentengenerierung



Verarbeitungspipelines

```
pipeline:

    include:
        input: "Content.kt"
        input: "med-core/glossary_en.kt"
        input: "med-core/bibliography_en.kt"

    xml-transform:
        stylesheet: "transform.xslt"
        stylesheet: "med-core/glossary.xslt"
        stylesheet: "med-core/bibliography.xslt"
        stylesheet: "unique-ids.xslt"
        stylesheet: "table-of-contents.xslt"

    xhtml-to-pdf:
        stylesheet: "htmlbook.css"
        output: "Document.pdf"
```

Prozessverankerung

- Notwendige Inhalte in der SOP festlegen
- Templates als Arbeitserleichterung, nicht als Vorgabe
- Verschiedene Templates bereitstellen
- Beim Review nicht an der Form festbeißen

Dokumente automatisch generieren!

- Nur die reine, redundanzfreie, minimal notwendige Information dokumentieren
- Vollständige Dokumente automatisch generieren
- Generierung eigentlich nur für ein Release notwendige
- aber: Mechanismus vorab, dauernd testen

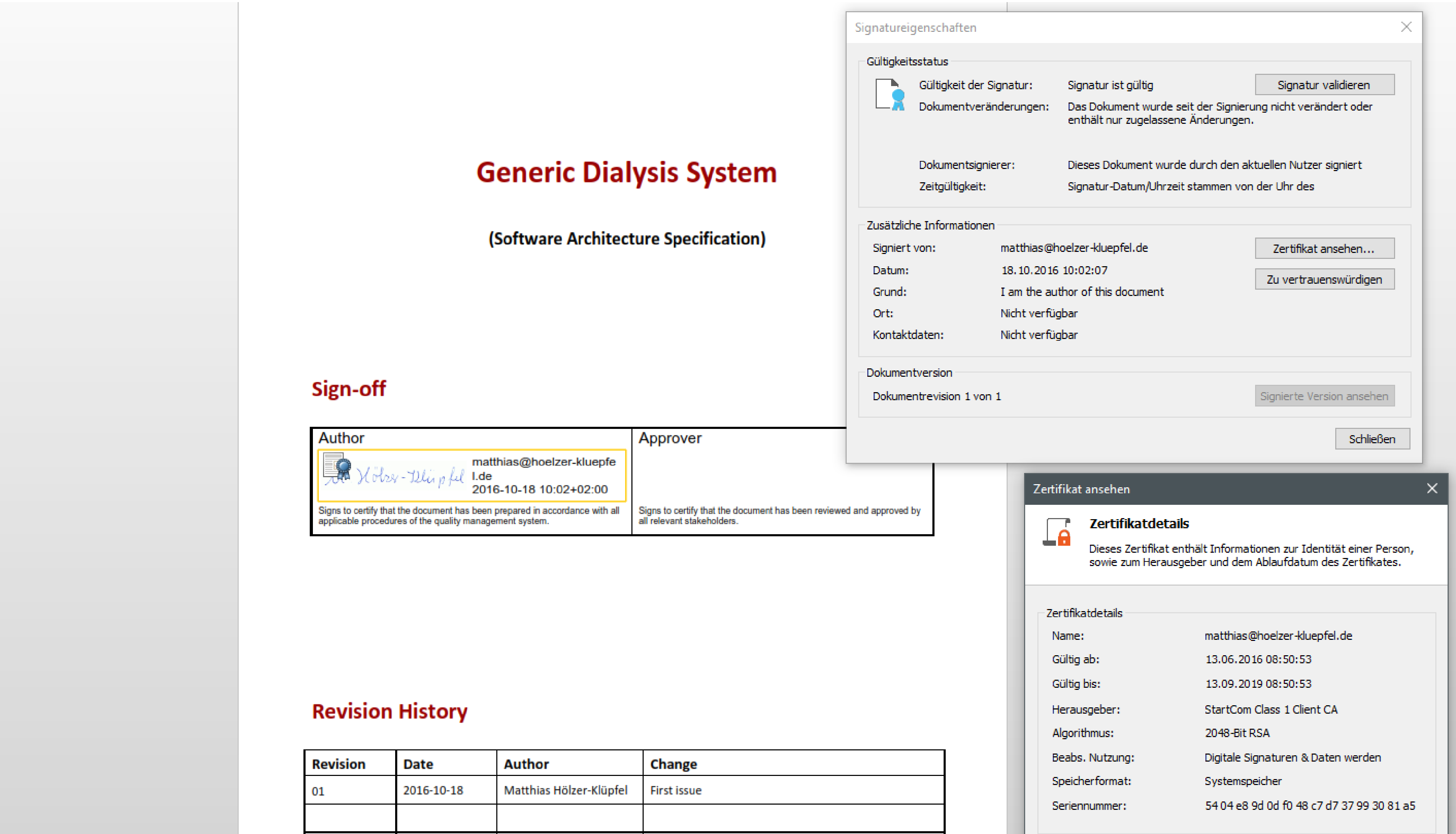
Freigabe vereinfachen

Problemstellung

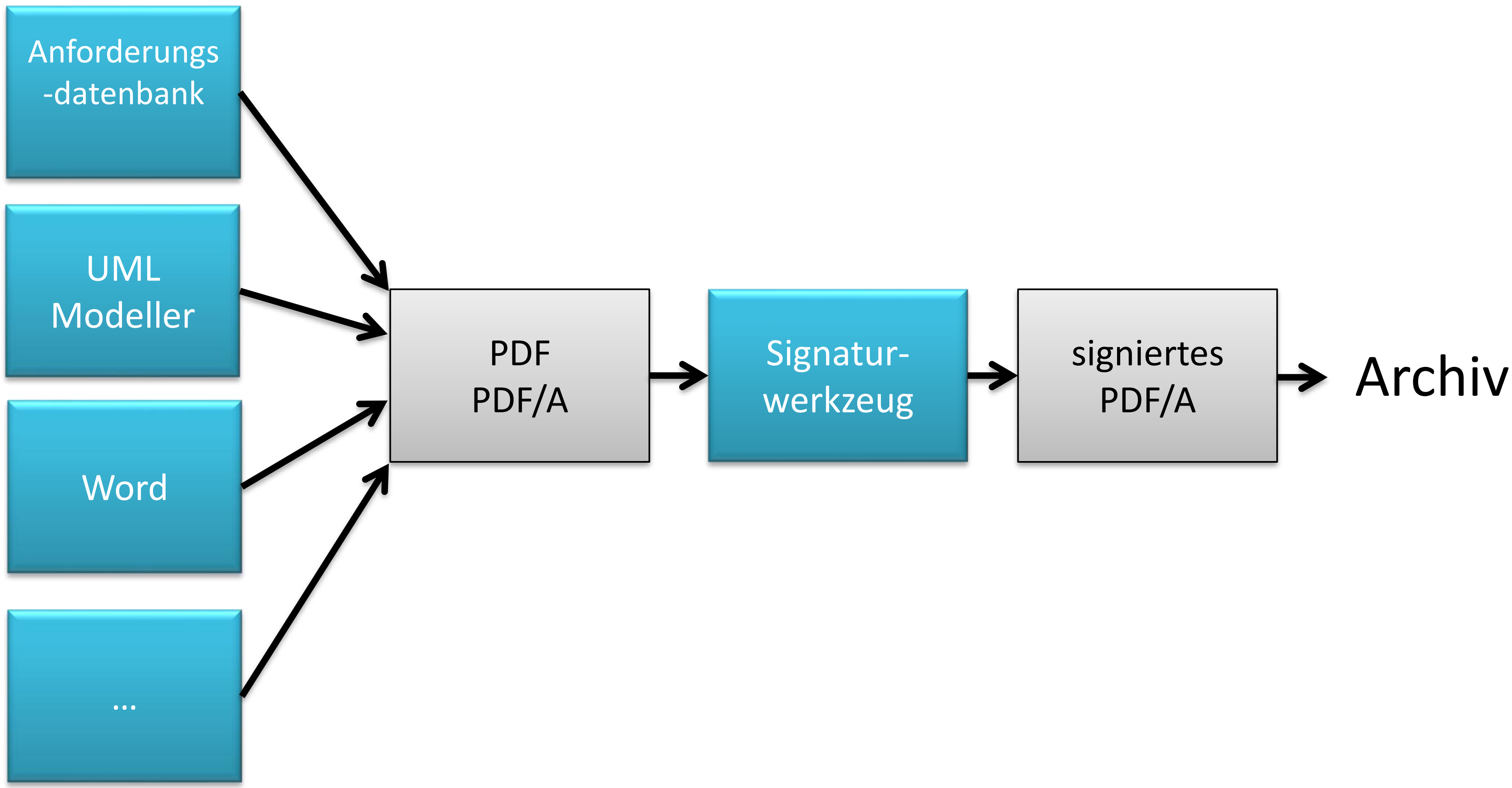
- Freigabe von Dokumentation ist ein manueller Prozess
- Reviews sind aufwändig und nie fertig
- Unterschriften erscheinen oft „endgültig“
  → die Unterschrift wird gerne hinausgezögert

- eine große Anzahl an Unterschriften löst das Problem nicht…

Bedeutung von Unterschriften definieren



Elektronische Signatur



Elektronische Signatur

Idealer Workflow:

- Integration in einen Pull Request
- Elektronische Signatur zur Bestätigung eines Approvals
- Automatische PDF-Signatur der entstandenen Dokumente

Freigabeprozesse leichtgewichtig gestalten!

- Die Bedeutung von Unterschriften definieren
- Elektronische Unterschriften verwenden
- Anzahl der notwendigen Unterschriften reduzieren

Fazit

- Entwicklungswerkzeuge nutzen!
- Dokumentation wie Code behandeln!
- Im klartext dokumentieren!
- Dokumentation in Workflow integrieren!
- Dokumente automatisch generieren!
- Freigabeprozesse leichtgewichtig gestalten!

Kontakt

MATTHIAS HÖLZER-KLÜPFEL
DIPLOM-PHYSIKER, M.SC.

- post Günterslebener Str. 15
  97291 Thüngersheim
- mobil +49 176 6085 7994
- mail matthias@hoelzer-kluepfel.de
  web www.hoelzer-kluepfel.de