

Pragmatisch Dokumentieren

Entwicklungsergebnisse effizient festhalten



Matthias Hölzer-Klüpfel

[illegible]

Die richtigen Werkzeuge



Projektalltag



Ich bin
Qualitätsmanager;
ich schreibe nur mit
Word!

Eine Risikoanalyse
ist eine Excel-
Tabelle!

Detailliertes
Design? Mache ich
schon immer in
Word!

Wer auch nur noch
ein einziges UML-
Diagramm in Visio
malt, fliegt raus!

Medienbrüche

SW-Anforderungen

SW-Systemprüfung

SW-Architektur

SW-Integrationstest

Detailliertes Design

Verifikation d. Impl.

Implementierung

Medienbrüche

DOORS

A black rectangular button with the word "DOORS" in white capital letters.

Squish

A lime green rectangular button with the word "Squish" in white capital letters.

Enterprise Architect

An orange rectangular button with the text "Enterprise Architect" in white capital letters.

googletest

A lime green rectangular button with the text "googletest" in white lowercase letters.

Doxygen

A lime green rectangular button with the word "Doxygen" in white capital letters.

Code Collaborator

A blue rectangular button with the text "Code Collaborator" in white capital letters.

Visual Studio

A lime green rectangular button with the text "Visual Studio" in white capital letters.

Vergleich Code ↔ Dokumentation

Code

- ▶ leicht zu ändern
- ▶ kurze Iterationen
- ▶ erläuternde Kommentare
- ▶ automatisierter Test
- ▶ zentrale IDE
- ▶ eindeutige Quelle der Wahrheit

Dokumentation

- ▶ leicht zu ändern
- ▶ lange Revisionen
- ▶ steht für sich selbst
- ▶ manuelles Review
- ▶ vielfältige Tools
- ▶ 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

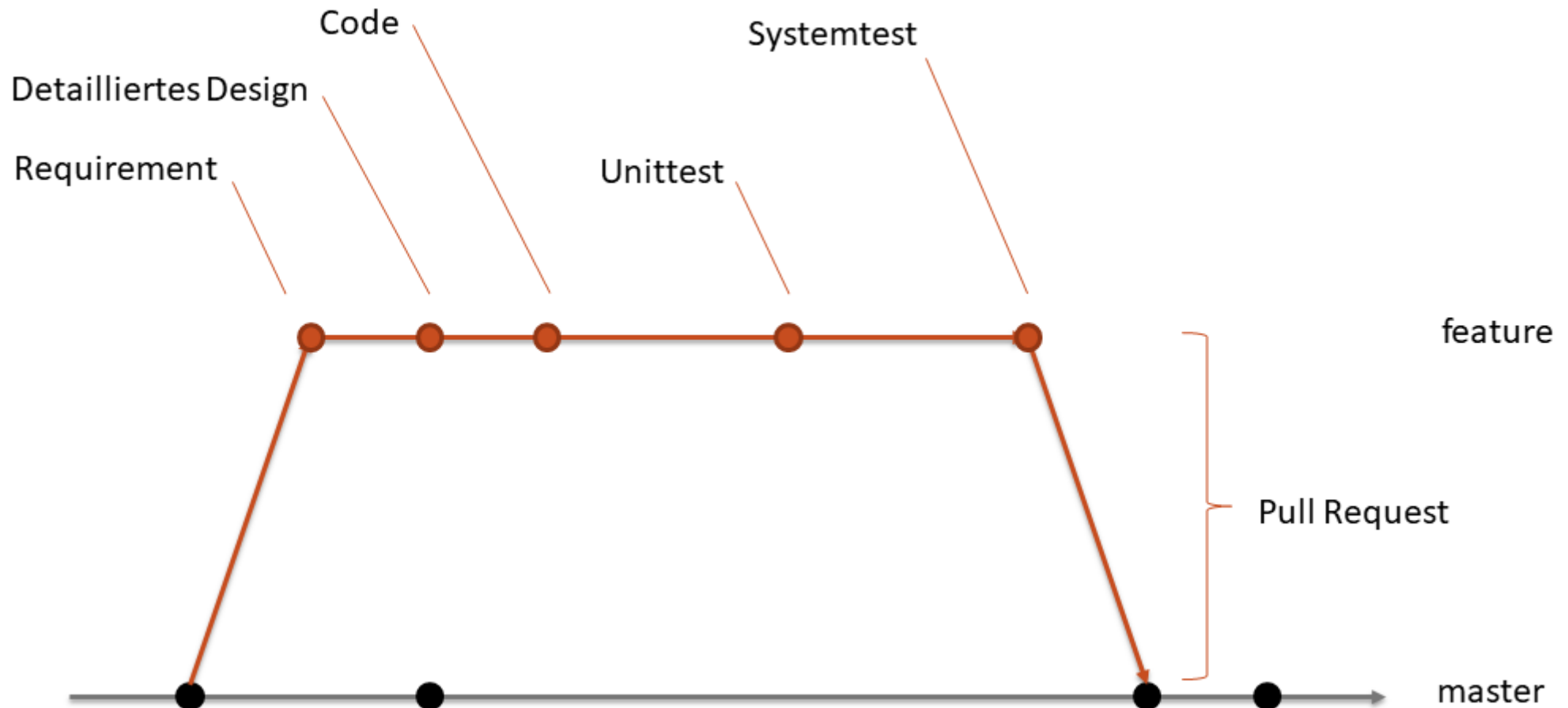
Dokumente

- ▶ 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

```
<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

File Edit Selection View Go Run Terminal Help

vdI-5702-1 [Dev Container: dossier]

EXPLORER

- VDI-5702-1 [DEV CONTAINER: DOSSIER]
- .devcontainer
- .vscode
- content
 - base-practices
 - Software Configuration Management Process Gro...
 - Software Development Process Group
 - SD_Software Development.kt
 - SD1_Software Development Lifecycle Definition.kt
 - SD2_Software Development Planning.kt
 - SD3_Software Requirements Analysis.kt
 - SD4_Software Architectural Design.kt
 - SD5_Software Detailed Design.kt**
 - SD6_Software Risk Management.kt
 - SD7_Software Implementation.kt
 - SD8_Software Integration and Integration Test.kt
 - SD9_Software Verification.kt
 - SD11_Software Release.kt
 - Software Maintenance Process Group
 - Software Problem Resolution Process Group
 - Standalone Software Development Process Group
 - descriptions
 - work-products
 - med-core
 - review
 - style
- .gitattributes
- .gitignore
- azure-pipelines.yml
- pdf-export.dm
- README.md
- test.html
- test.xml
- unit-test.dm
- OUTLINE
- TIMELINE

SD5_Software Detailed Design.kt

```
content > base-practices > Software Development Process Group > SD5_Software Detailed Design.kt
1 process: #SD5 tag="SD.5" name="Software Detailed Design"
2
3
4 purpose:
5     The objective of the /r.q/SD5/ {process} is to refine the {software items} and interfaces defined in the software architecture to
6     create {software units} and their interfaces.
7
8     Detailed design specifies algorithms, data representations, interfaces among different {software units}, and interfaces between
9     {software units} and data structures.
10
11     It is necessary to document the design of each {software unit} and its interface so that the {software unit} can be implemented
12     correctly.
13
14     The software detailed design fills in the details necessary to construct the software. It should be complete enough that the
15     programmer is not required to make ad hoc design decisions.
16
17 outcomes:
18     outcome: #SD5_OC1 The software architecture is refined until it is represented by {software units} that can be implemented and
19     tested separately.
20     outcome: #SD5_OC2 Each {software unit} is designed.
21     outcome: #SD5_OC6 A secure software design is developed.
22     outcome: #SD5_OC4 The interfaces between {software units} and external components as well as in between {software units} are
23     defined.
24     outcome: #SD5_OC7 Security of the interfaces is considered.
25     outcome: #SD5_OC5 The software detailed design is verified.
26
27 base-practices:
28     base-practice: #SD5_BP1 tag="SD.5 BP.1" name="Subdivide Software into Software Units"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE SPELL CHECKER

bash - vdi-5702-1

vscode@756807d30722:/workspaces/vdi-5702-1\$

Dev Container: dossier mhk/draft_review* 0 0

Matthias Hölzer-Klöpfl (3 years ago) Ln 1, Col 24 (4 selected) Spaces: 4 UTF-8 LF {} klartext

Beispiel: Medical SPICE

medical-spice.pdf

Seite: 33 / 134

Suchen:

Lesezeichen

- 6.1 Software Development (SD)
 - 6.1.1 Software Development Life Cycle Definition (SD.1)
 - 6.1.2 Software Development Planning (SD.2)
 - 6.1.3 Software Requirements Analysis (SD.3)
 - 6.1.4 Software Architectural Design (SD.4)
 - 6.1.5 Software Detailed Design (SD.5)
 - 6.1.5.1 Purpose
 - 6.1.5.2 Outcomes
 - 6.1.5.3 Base practices
 - 6.1.5.3.1 SD.5_BP.1 - Subdivide Software into Software Units
 - 6.1.5.3.2 SD.5_BP.2 - Design each Software Unit
 - 6.1.5.3.3 SD.5_BP.3 - Design Interfaces
 - 6.1.5.3.4 SD.5_BP.4 - Verify Detailed Design
 - 6.1.5.4 Work products
 - 6.1.6 Software Risk Management (SD.6)
 - 6.1.7 Software Implementation (SD.7)
 - 6.1.7.1 Purpose
 - 6.1.7.2 Outcomes
 - 6.1.7.3 Base practices
 - 6.1.7.3.1 SD.7_BP.1 - Implement Software Units
 - 6.1.7.3.2 SD.7_BP.2 - Ensure Acceptance Criteria
 - 6.1.7.3.3 SD.7_BP.3 - Verify Software Units
 - 6.1.7.4 Work products
 - 6.1.8 Software Integration and Integration Test (SD.8)
 - 6.1.8.1 Purpose
 - 6.1.8.2 Outcomes
 - 6.1.8.3 Base practices
 - 6.1.8.3.1 SD.8_BP.1 - Integrate Software Units
 - 6.1.8.3.2 SD.8_BP.2 - Verify Software Integration
 - 6.1.8.3.3 SD.8_BP.3 - Perform Software Integration Test
 - 6.1.8.3.4 SD.8_BP.4 - Perform Regression Tests
 - 6.1.8.3.5 SD.8_BP.5 - Record Integration Test Results
 - 6.1.8.3.6 SD.8_BP.6 - Use the Software Problem Resolution Process
 - 6.1.8.3.7 SD.8_BP.7 - Evaluate Integration Test Process
 - 6.1.8.4 Work products
 - 6.1.9 Software Verification (SD.9)
 - 6.1.9.1 Purpose
 - 6.1.9.2 Outcomes
 - 6.1.9.3 Base practices
 - 6.1.9.3.1 SD.9_BP.1 - Perform Software Verification
 - 6.1.9.3.2 SD.9_BP.2 - Perform Regression Testing
 - 6.1.9.3.3 SD.9_BP.3 - Record Software Verification Results
 - 6.1.9.3.4 SD.9_BP.4 - Use the Software Problem Resolution Process
 - 6.1.9.3.5 SD.9_BP.5 - Evaluate Software System Test Results
 - 6.1.9.4 Work products
 - 6.1.10 Software Release (SD.11)
 - 6.1.10.1 Purpose
 - 6.1.10.2 Outcomes
 - 6.1.10.3 Base practices
 - 6.1.10.3.1 SD.11_BP.1 - Complete Software Verification
 - 6.1.10.3.2 SD.11_BP.2 - Document Residual Anomalies

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)
SOP 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

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



Ich bin
Qualitätsmanager;
ich schreibe nur mit
Word!

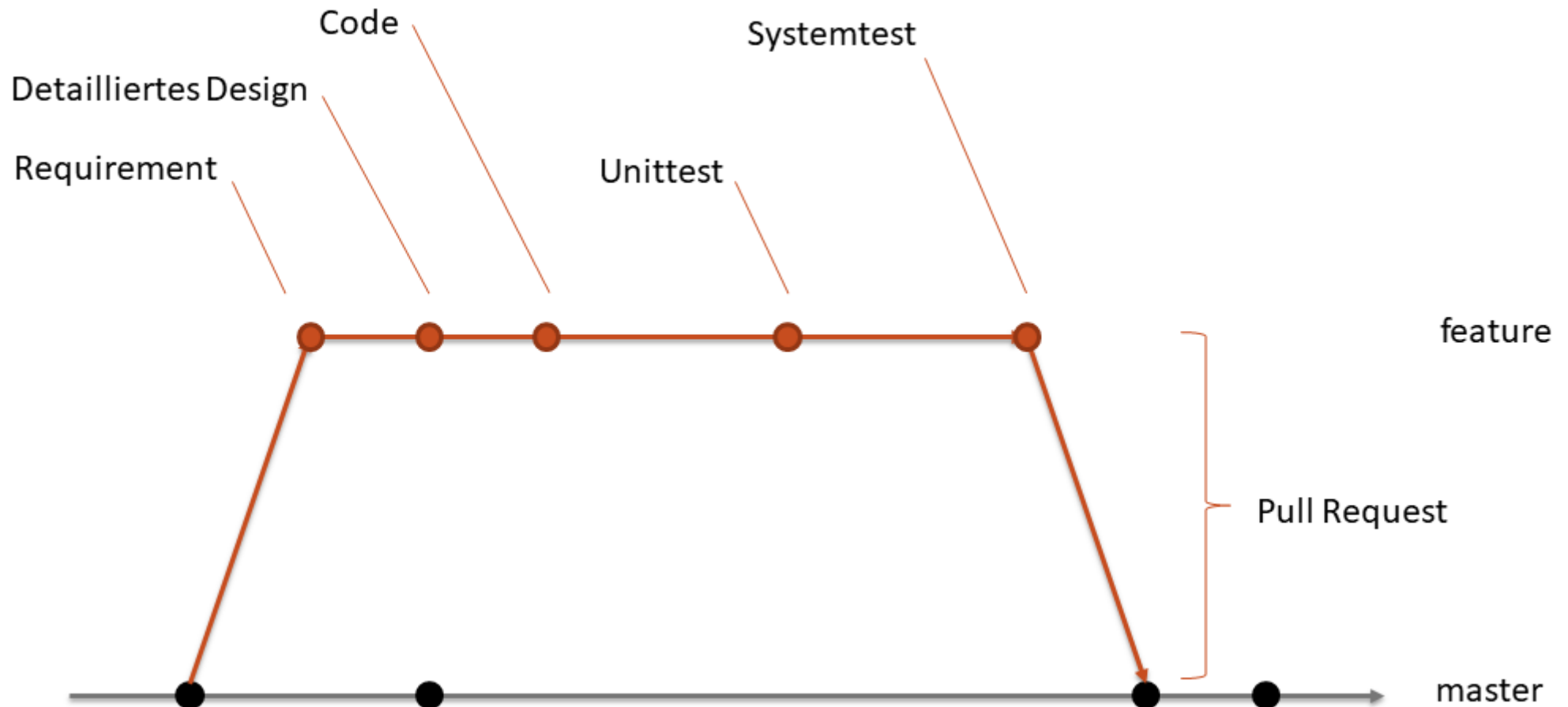
Meine Entwickler
würden sich am
liebsten auch noch
die Zähne mit git
putzen

Eine Risikoanalyse
ist eine Excel-
Tabelle!

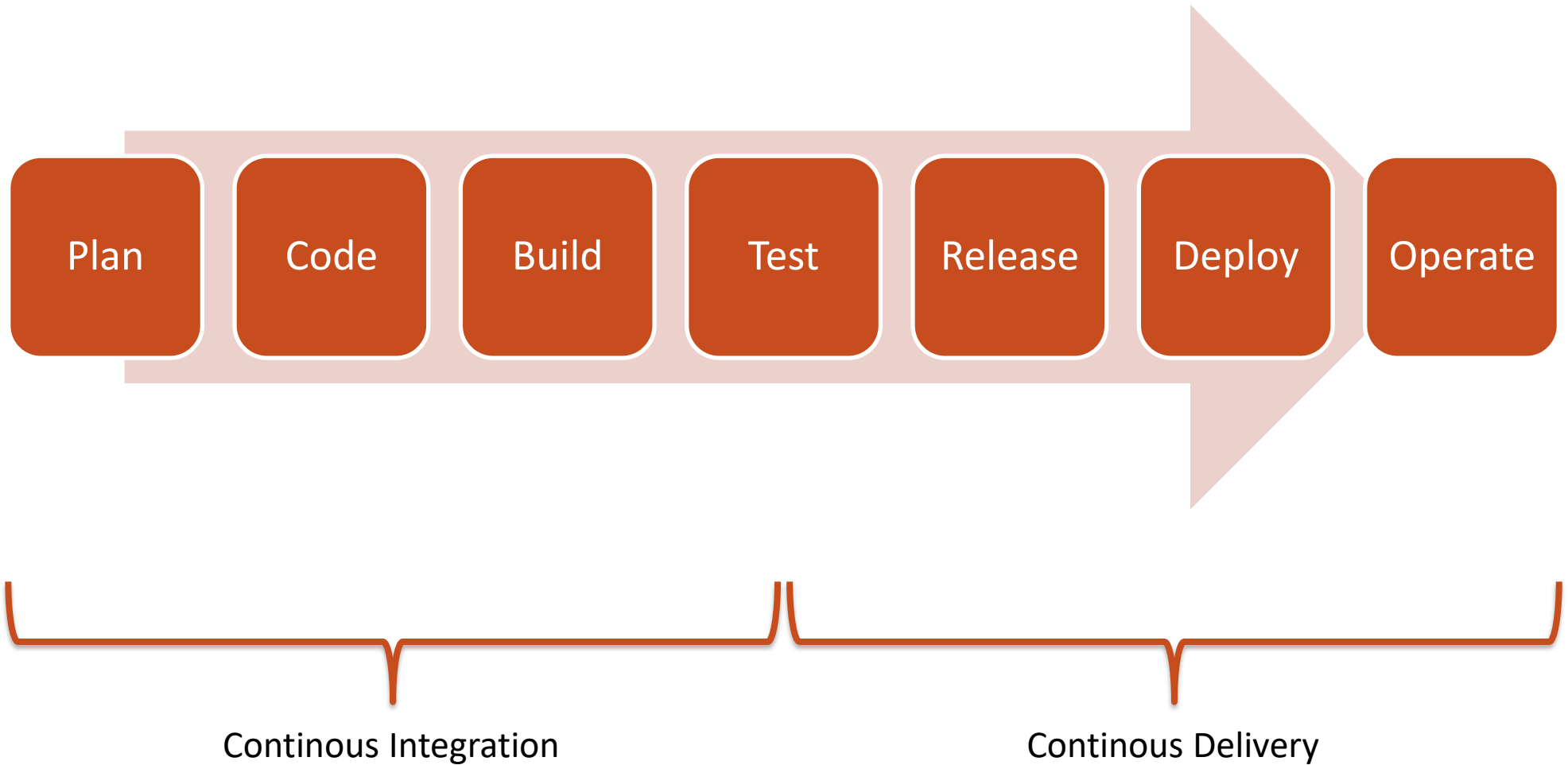
Detailliertes
Design? Mache ich
schon immer in
Word!

Wer auch nur noch
ein einziges UML-
Diagramm in Visio
malt, fliegt raus!

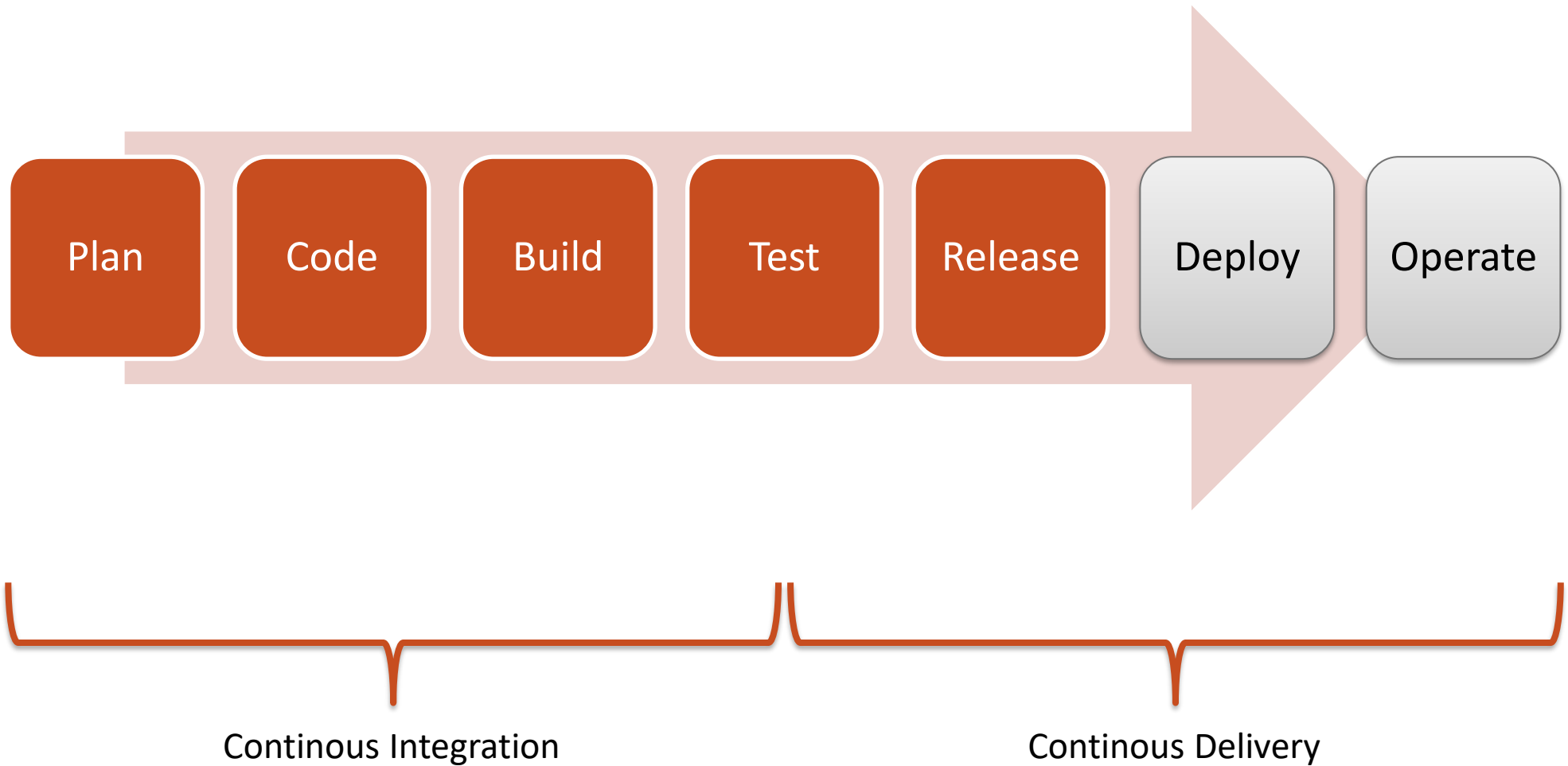
Pull Request



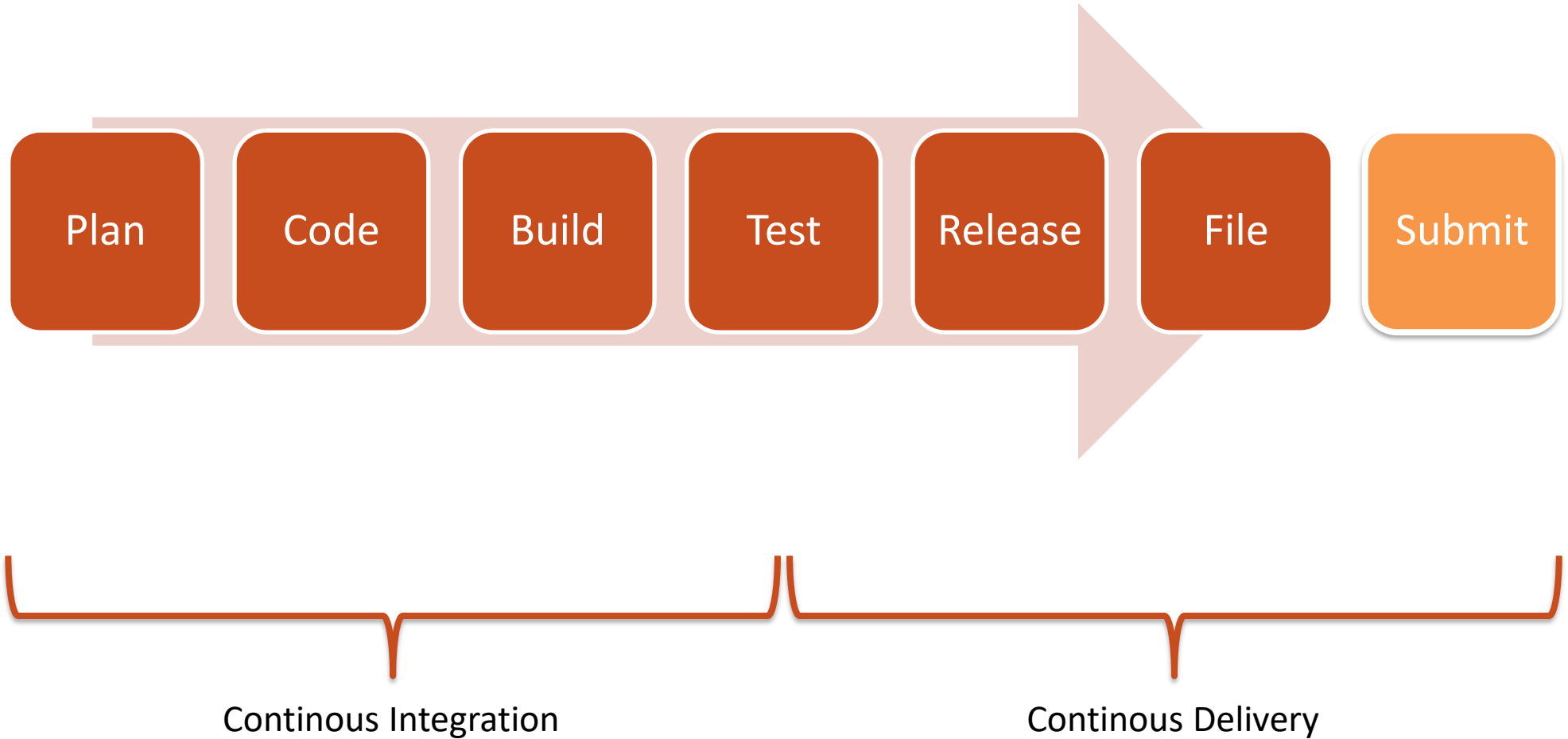
DevOps-Pipeline



DevOps-Pipeline



DevOps-Pipeline



Dokumentation in Workflow integrieren!



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

Dokumentengenerierung



Beispiel: Testskript

Beispiel.xml X

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <?xml-stylesheet type="text/xsl" href="test-script.xsl"?>
3
4  <testscript revision="$Revision: 123 $" version="1.0.0" xmlns="http://www.hoelzerkluepfel.de/testscript">
5
6      <sequence id="TEST1" name="Alarmfunktion">
7
8          <description>
9              Dieser Test überprüft die Funktion des visuellen Alarms.
10          </description>
11
12          <repeat count="5">
13              <wait start="490ms" timeout="510ms">
14                  <check-transition name="Alarm LED" from="Off" to="On"/>
15              </wait>
16              <wait start="490ms" timeout="510ms">
17                  <check-transition name="Alarm LED" from="On" to="Off"/>
18              </wait>
19          </repeat>
20
21          <verify requirement="REQ1234"/>
22
23      </sequence>
24
25  </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

Step	Action	Expected Result									
#1	Repeat the following steps for 5 times:	All repeated commands pass									
	<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>		Step	Action	Expected Result	#1	Wait for the following condition/s to become true: The 'Alarm LED' transitions from 'Off' to 'On'	Condition true in less than 510ms but not before 490ms	#2	Wait for the following condition/s to become true: The 'Alarm LED' transitions from 'On' to 'Off'	Condition true in less than 510ms but not before 490ms
	Step		Action	Expected Result							
	#1		Wait for the following condition/s to become true: The 'Alarm LED' transitions from 'Off' to 'On'	Condition true in less than 510ms but not before 490ms							
#2	Wait for the following condition/s to become true: The 'Alarm LED' transitions from 'On' to 'Off'	Condition true in less than 510ms but not before 490ms									

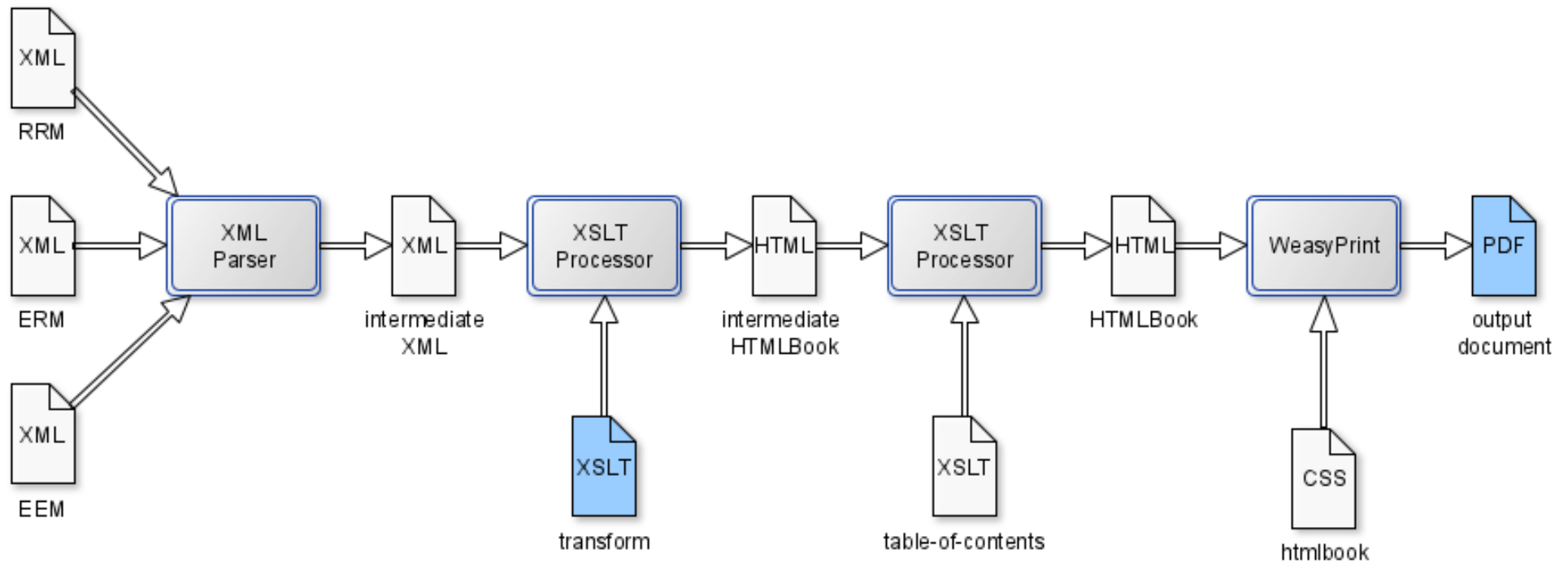
Finished verification for requirement REQ1234

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

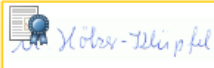
- ▶ Freigabe von Dokumentation ist ein manueller Prozess
- ▶ Reviews sind aufwändig und nie fertig
- ▶ Unterschriften erscheinen oft „endgültig“
→ die Unterschrift wird gerne hinausgezögert
- ▶ eine große Anzahl an Unterschriften löst das Problem nicht...

Bedeutung von Unterschriften definieren

Generic Dialysis System

(Software Architecture Specification)

Sign-off

Author	Approver
 matthias@hoelzer-kluepfe l.de 2016-10-18 10:02+02:00 Signs to certify that the document has been prepared in accordance with all applicable procedures of the quality management system.	Signs to certify that the document has been reviewed and approved by all relevant stakeholders.

Revision History

Revision	Date	Author	Change
01	2016-10-18	Matthias Hölzer-Klüpfel	First issue

Signatureeigenschaften

Gültigkeitsstatus



Gültigkeit der Signatur:Signatur ist gültig

Dokumentveränderungen:Das Dokument wurde seit der Signierung nicht verändert oder enthält nur zugelassene Änderungen.

Dokumentsignierer:Dieses Dokument wurde durch den aktuellen Nutzer signiert

Zeitgültigkeit:Signatur-Datum/Uhrzeit stammen von der Uhr des

Zusätzliche Informationen

Signiert von:matthias@hoelzer-kluepfel.de

Datum:18.10.2016 10:02:07

Grund:I am the author of this document

Ort:Nicht verfügbar

Kontaktdaten:Nicht verfügbar

Dokumentversion

Dokumentrevision 1 von 1

Signatur validieren

Zertifikat ansehen...

Zu vertrauenswürdigem

Signierte Version ansehen

Schließen

Zertifikat ansehen



Zertifikatdetails

Dieses Zertifikat enthält Informationen zur Identität einer Person, sowie zum Herausgeber und dem Ablaufdatum des Zertifikates.

Zertifikatdetails

Name:matthias@hoelzer-kluepfel.de

Gültig ab:13.06.2016 08:50:53

Gültig bis:13.09.2019 08:50:53

Herausgeber:StartCom Class 1 Client CA

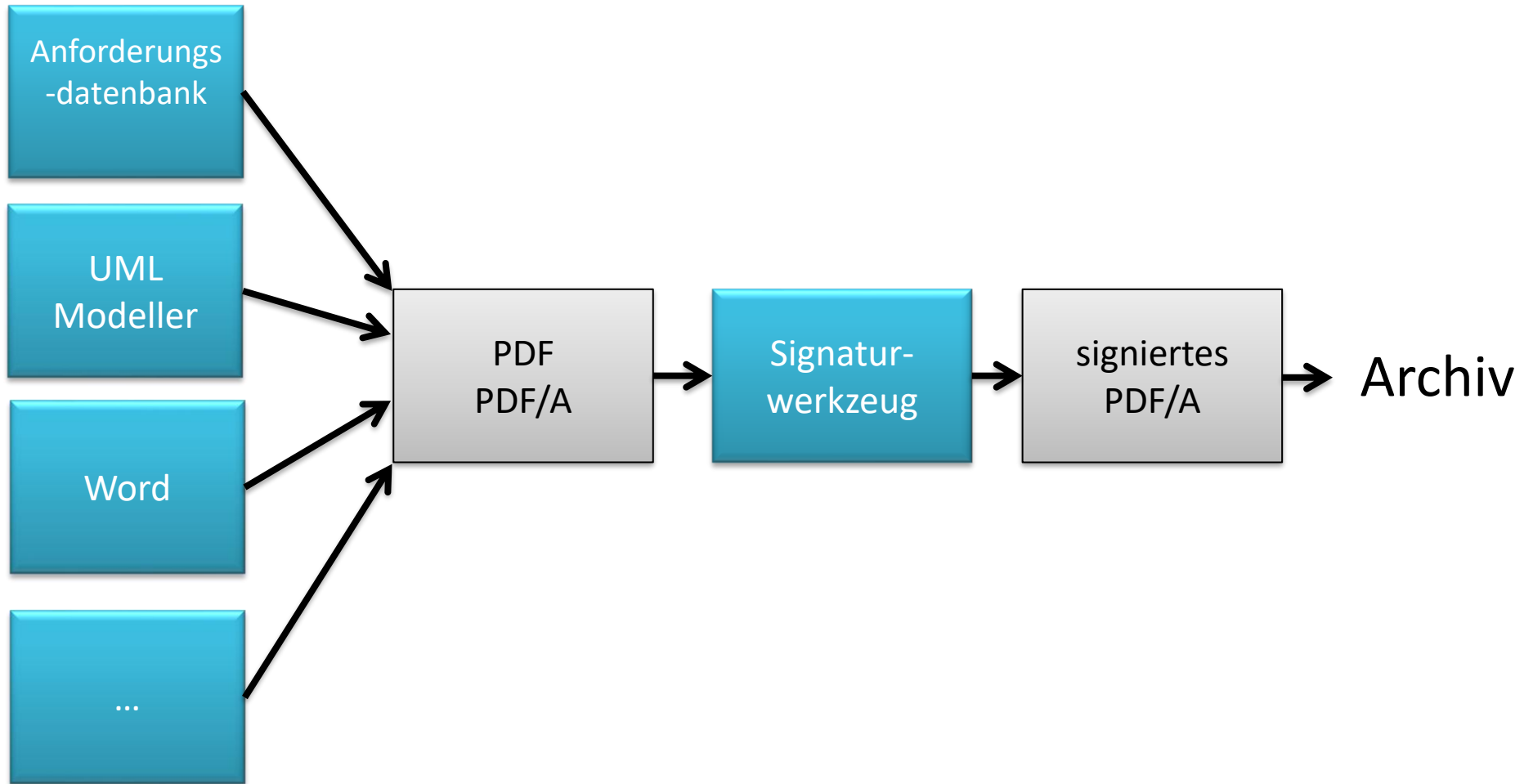
Algorithmus:2048-Bit RSA

Beabs. Nutzung:Digitale Signaturen & Daten werden

Speicherformat:Systemspeicher

Seriennummer:54 04 e8 9d 0d f0 48 c7 d7 37 99 30 81 a5

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