Template

Inhaltsverzeichnis

Über arc42

arc42, das Template zur Dokumentation von Software- und Systemarchitekturen.

Erstellt von Dr. Gernot Starke, Dr. Peter Hruschka und Mitwirkenden.

Template Revision: 7.0 DE (asciidoc-based), January 2017

Diese Version des Templates enthält Hilfen und Erläuterungen. Sie dient der Einarbeitung in arc42 sowie dem Verständnis der Konzepte. Für die Dokumentation eigener System verwenden Sie besser die *plain* Version.

1. Einführung und Ziele

Dieser Abschnitt führt in die Aufgabenstellung ein und skizziert die Ziele, die Apollo Auto verfolgt.

1.1. Aufgabenstellung

1.1.1. Was ist Apollo Auto?

Apollo ist eine hochleistungsfähige, flexible Architektur, die die Entwicklung, das Testen und den Einsatz von autonomen Fahrzeugen beschleunigt. Apollo Auto bietet unter Andrem Lösungen für Valet Parking, V2X-Kommunikation und intelligente Lichtsignalanlagen.

1.1.2. Wesentliche Features:

- Valet Parking
  - Software- und Hardware-Integrationslösung. Multifusionslösung bestand aus Fahrzeug, Cloud, HD-Karte und Parkplätzen
  - Bietet hochwertige Dienstleistungen, wie automatische Parkplatzerkennung und autonomes Parken, für Kunden.
- V2X-Communication
  - Interaktionslösung für intelligente Fahrzeuginfrastruktur
  - Apollo V2X umfasst ein intelligentes Transportsystem für Fahrzeug Straßendatenerfassung und intelligente Verarbeitungsanalyse, Verkehrssicherheit und -effizienz
  - Wahrnehmung aller Verkehrsteilnehmer im Sichtfeld und die bereitgestellten straßenseitigen Sensorinformationen können für die Entscheidungsfindung beim autonomen Fahren auf hohem Niveau verwendet werden
  - Wahrnehmung von Verkehrteilnehmern ausserhalb des Sichtfeldes

- Bietet einen vollständigen, kontinuierlichen, multimodalen Datendienst mit niedriger Latenz für L4-Autopilot-Fahrzeuge, die in mehreren Szenarien getestet wurden
- Durch die permanente dynamische Erfassung von Verkehrsinformationen und die Cloud-Integration, wird eine weltweite optimale kollaborative Steuerungsfunktionen für Verkehrsteilnehmer und Verkehrsmanagement erreicht
    - Smart Traffic Signals
- Holographisches Wahrnehmen und Verstehen, basierend auf dem holografischen Wahrnehmungs- und Erkennungssystem
- Status von Fußgängern und Fahrzeugen auf jeder Fahrspur genau erkennen und die Leistung des aktuellen Verkehrsflusses wie Volumen, Warteschlangenlänge, Verspätungen usw.
- Vollständige raum-zeitliche Ableitung und Entscheidungsfindung
- Echtzeitsteuerung der gesamten Szene
- Reduzierung der durchschnittlichen Wartezeit um 20-30% während der Rush Hour
    - Robotaxi
- Die Robotaxis, die aus Chinas erstem werkseitig installierten L4-Passagier-Fahrzeug sind zur Zeit auf öffentlichen Straßen im Testbetrieb
- Sie werden in Kooperation von Baido und FAW an einer gemeinsamen Produktionsline hergstellt
    - Minibus
- Die Minibusse ermöglichen ebenfalls autonomes Fahren der Stufe 4
- Funktionen sind unter Anderem Hinderniserkennung und -vermeidung, zu einem Zielort Fahren und Kreuzungen überqueren

1.2. Qualitätsziele

Die folgende Tabelle beschreibt die zentralen Qualitätsziele von DokChess, wobei die Reihenfolge eine grobe Orientierung bezüglich der Wichtigkeit vorgibt.

Qualitätsziel	**Motivation und Erläuterung**
Zugängliches Beispiel (Analysierbarkeit)	• Apollo Auto ist eine offene Plattform • Daher ist es wichtig, dass sich neue Entwickler möglichst schnell in die Architektur, Entwurf und Implementierung einarbeiten können
Echzeitsteuerung von einzelnen Fahrzeugen und Verkehrströmen	• Apollo Auto übernimmt zuverlässig und sicher die autonome Steuerung von Fashrzeugen auf Level 4
Echtzeit Umfelderkennung	• Für die Steuerung von Fahrzeugen wird ein Modell des Umfelds benötigt • Aus den Sensoprdaten wird ein digitales Abbild des Fahrweges, von beweglichen und unbeweglichen Hindernissen und von Signalen geschaffen
Prediction	• Um die Fahrsicherheit weiter zu erhöhen, wird auf die Sensor- und Zustandsdaten von anderen Verkehrsteilnehmern in Echtzeit zugegriffen

1.3. Stakeholder

Die folgende Tabelle stellt die Stakeholder von Apollo Auto und ihre jeweilige Intention dar.

Rolle	**Interesse, Bezug**
Softwarearchitekten	• Wollen ein Gefühl bekommen, wie Architekturdokumentation für ein konkretes System aussehen kann • Möchten sich Dinge (z.B. Form, Notation) für Ihre tägliche Arbeit abgucken • Gewinnen Sicherheit für Ihre eigenen • Haben in der Regel keine tiefen Schachkenntnisse
Entwickler	• Nehmen Architekturaufgaben im Team wahr • Brauchen ein generelles Verständnis für die Architektur
OEM & Lieferanten	• Entwickeln neue Produkte auf Grundlage von Apollo Auto • Wollen Anregungen für eigene Produkte finden
Gesetzgeber & Genehmigungsbehörden	• Entwickeln einen gesetzlichen Rahmen zur Zulassung von fahrerlosen Fahrzeugen im öffentlichen Straßenverkehr • Etablieren Prüfvorschriften und Tests für Genehmigungsverfahren

Rolle	Interesse, Bezug
Universitäten	• Entwickeln eigene Forschungsprojekte auf Grundlage von Apollo Auto • Wollen Anregungen für weitere Forschungsprojekte und studentische Arbeiten finden
Studenten	• Interessieren sich aufgrund ihres Studiums für die verschiedenen • Setzen eigene Projekte (z.B. Masterarbeit) zum Thema autonomes Fahren mit Apollo Auto um • Schreiben eine Architektur Dokumentation zu Apollo Auto

2. Randbedingungen

Beim Einsatz von Apollo sind verschiedene Randbedingungen zu beachten. Dieser Abschnitt stellt sie dar und erklärt auch – wo nötig – deren Motivation.

2.1. Technische Randbedingungen

- Für den Einsatz von Apollo Auto wird eine anspruchsvolle Hardwareausstattung benötigt, eine Lösung mit einem marktüblichen Standard-Notebook allein ist nicht möglich
- Ein Fahrzeug, das mit By-Wire-Systemen ausgestattet ist, zum Beispiel Brake-by-Wire, Steering-by-Wire, Throttle-by-Wire oder Shift-by-Wire (Apollo wird derzeit auf Lincoln MKZ getestet).
- Ein Rechner mit einem 4-Kern-Prozessor und mindestens 8 GB Speicher (16 GB für Apollo 3.5 und höher)
- Ubuntu 18.04
- Zusätzlich wird eine umfangreiche Sensorik benötigt die Bild- und Abstandsinformationen aus dem Umfeld aufnehmen
- Arbeitskenntnisse über Docker

2.2. Organisatorische Randbedingungen

hier irgendwas das github verwendet wird .... sonst kein plan wie die arbeiten, denke mal auch verteilt

Randbedingung	Erläuterungen, Hintergrund
github	Quellcode ist über github verfügbar.
Bereitstellung von Daten	Alle Daten müssen in einem Format hochgeladen werden, das den Apollo-Datenspezifikationen entspricht.

Speicherung von Daten	Daten, die in China gesammelt wurden, dürfen nur auf Servern in China gespeichert werden. Daten, die in anderen Ländern und Regionen erhoben werden, unterliegen den Beschränkungen der Datenspeicherung, die durch die Gesetze der jeweiligen Länder festgelegt sind.
Bereitstellung von Daten	Als Initiator dieser Plattform stellt Baidu die Ausgangsdaten für diese Plattform bereit. Die Daten stehen allen Partnern dieser Plattform offen. Das Prinzip der fairen Daten stellt sicher, dass Partner mit größeren eigenen Beiträgen mehr Daten und Dienste von dieser Plattform erhalten.
Datenschutz	Jeder Partner kann seine eigenen Daten anzeigen und die Datenschutzeigenschaften der Daten als privat oder öffentlich festlegen. Die von Partnern hochgeladenen Daten gelten standardmäßig als privat.
Endscheidungsführung	Grundsätzlich handelt es sich bei Apollo Auto um eine offene Plattform. Allerdings wirde Baidu um die architektonische Integrität, die Systemzuverlässigkeit und die schnelle Entwicklung von Apollo zu gewährleisten, bei Bedarf wichtige Entscheidungen treffen, während die aktive Beteiligung der breiteren Gemeinschaft erhalten bleibt.

2.3. Konventionen

Konvention	Erläuterungen, Hintergrund
Dokumentation	*Terminologie und Gliederung nach dem deutschen arc42-Template in der Version 6.0*
Kodierrichtlinien für C++	*C++ Coding Conventions von Sun/Oracle, geprüft mit Hilfe von CheckStyle*
Kodierrichtlinien für Python	*Python Coding Conventions von Sun/Oracle, geprüft mit Hilfe von CheckStyle*
Spezifische Datenformate und Frameworks für autonomes Fahren	*Verwendung etablierter Standards für autonomes Fahren. zum Beispiel sind alle Softwaremodule als ROS(Robot Operating System)-Knoten zu behandeln.*

3. Kontextabgrenzung

Dieser Abschnitt beschreibt das Umfeld von Baidu Apollo. Für welche Benutzer ist es da, und mit welchen Fremdsystemen interagiert es?

Inhalt

Die Kontextabgrenzung grenzt das System von allen Kommunikationsbeziehungen (Nachbarsystemen und Benutzerrollen) ab. Sie legt damit die externen Schnittstellen fest.

Differenzieren Sie fachliche (fachliche Ein- und Ausgaben) und technische Kontexte (Kanäle, Protokolle, Hardware), falls nötig.

Motivation

Die fachlichen und technischen Schnittstellen zur Kommunikation gehören zu den kritischsten Aspekten eines Systems. Stellen Sie sicher, dass Sie diese komplett verstanden haben.

Form

Verschiedene Optionen:

- Diverse Kontextdiagramme
- Listen von Kommunikationsbeziehungen mit deren Schnittstellen

3.1. Fachlicher Kontext

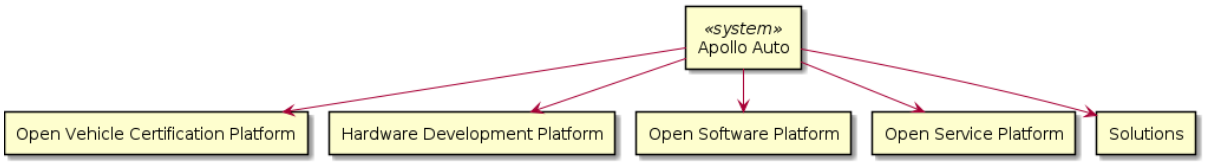


Abbildung 1. Benutzer und Benutzergruppen von VENOM

Inhalt

Festlegung **aller** Kommunikationsbeziehungen (Nutzer, IT-Systeme, ...) mit Erklärung der fachlichen Ein- und Ausgabedaten oder Schnittstellen. Zusätzlich (bei Bedarf) fachliche Datenformate oder Protokolle der Kommunikation mit den Nachbarsystemen.

Motivation

Alle Beteiligten müssen verstehen, welche fachlichen Informationen mit der Umwelt ausgetauscht werden.

Form

Alle Diagrammarten, die das System als Blackbox darstellen und die fachlichen Schnittstellen zu den Nachbarsystemen beschreiben.

Alternativ oder ergänzend können Sie eine Tabelle verwenden. Der Titel gibt den Namen Ihres Systems wieder; die drei Spalten sind: Kommunikationsbeziehung, Eingabe, Ausgabe.

<Diagramm und/oder Tabelle>

<optional: Erläuterung der externen fachlichen Schnittstellen>

3.2. Technischer Kontext

Inhalt

Technische Schnittstellen (Kanäle, Übertragungsmedien) zwischen dem System und seiner Umwelt. Zusätzlich eine Erklärung (*mapping*), welche fachlichen Ein- und Ausgaben über welche technischen Kanäle fließen.

Motivation

Viele Stakeholder treffen Architekturentscheidungen auf Basis der technischen Schnittstellen des Systems zu seinem Kontext.

Insbesondere bei der Entwicklung von Infrastruktur oder Hardware sind diese technischen Schnittstellen durchaus entscheidend.

Form

Beispielsweise UML Deployment-Diagramme mit den Kanälen zu Nachbarsystemen, begleitet von einer Tabelle, die Kanäle auf Ein-/Ausgaben abbildet.

<Diagramm oder Tabelle>

<optional: Erläuterung der externen technischen Schnittstellen>

<Mapping fachliche auf technische Schnittstellen>

ausführlich

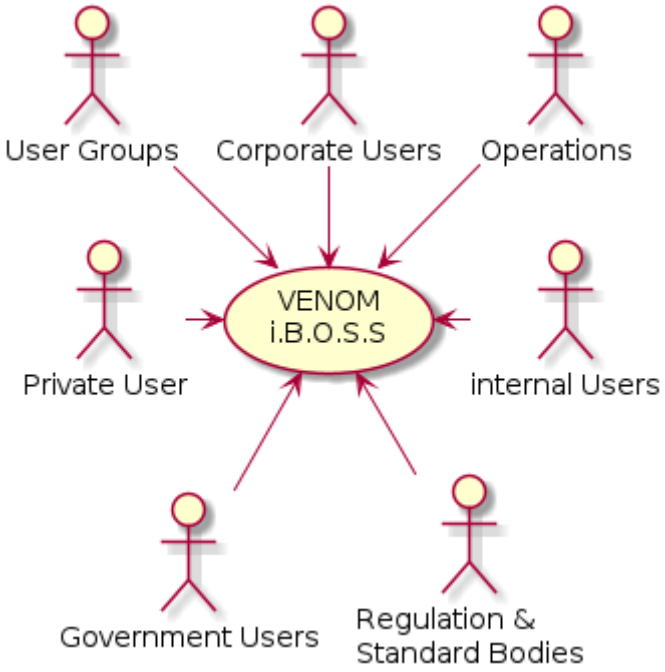


Abbildung 2. Benutzer und Benutzergruppen von VENOM

4. Lösungsstrategie

Dieser Abschnitt enthält einen stark verdichteten Architekturüberblick. Eine Gegenüberstellung der wichtigsten Ziele und Lösungsansätze.

Inhalt

Kurzer Überblick über die grundlegenden Entscheidungen und Lösungsansätze, die Entwurf und Implementierung des Systems prägen. Hierzu gehören:

- Technologieentscheidungen
- Entscheidungen über die Top-Level-Zerlegung des Systems, beispielsweise die Verwendung gesamthaft prägender Entwurfs- oder Architekturmuster,
- Entscheidungen zur Erreichung der wichtigsten Qualitätsanforderungen sowie
- relevante organisatorische Entscheidungen, beispielsweise für bestimmte Entwicklungsprozesse oder Delegation bestimmter Aufgaben an andere Stakeholder.

Motivation

Diese wichtigen Entscheidungen bilden wesentliche „Eckpfeiler“ der Architektur. Von ihnen hängen viele weitere Entscheidungen oder Implementierungsregeln ab.

Form

Fassen Sie die zentralen Entwurfsentscheidungen **kurz** zusammen. Motivieren Sie, ausgehend von Aufgabenstellung, Qualitätszielen und Randbedingungen, was Sie entschieden haben und warum Sie so entschieden haben. Vermeiden Sie redundante Beschreibungen und verweisen Sie eher auf weitere Ausführungen in Folgeabschnitten.

5. Bausteinsicht

Inhalt

Diese Sicht zeigt die statische Zerlegung des Systems in Bausteine sowie deren Beziehungen. Beispiele für Bausteine sind unter anderem:

- Module
- Komponenten
- Subsysteme
- Klassen
- Interfaces
- Pakete
- Bibliotheken
- Frameworks
- Schichten
- Partitionen
- Tiers
- Funktionen
- Makros
- Operationen
- Datenstrukturen
- …

Diese Sicht sollte in jeder Architekturdokumentation vorhanden sein. In der Analogie zum Hausbau bildet die Bausteinsicht den *Grundrissplan*.

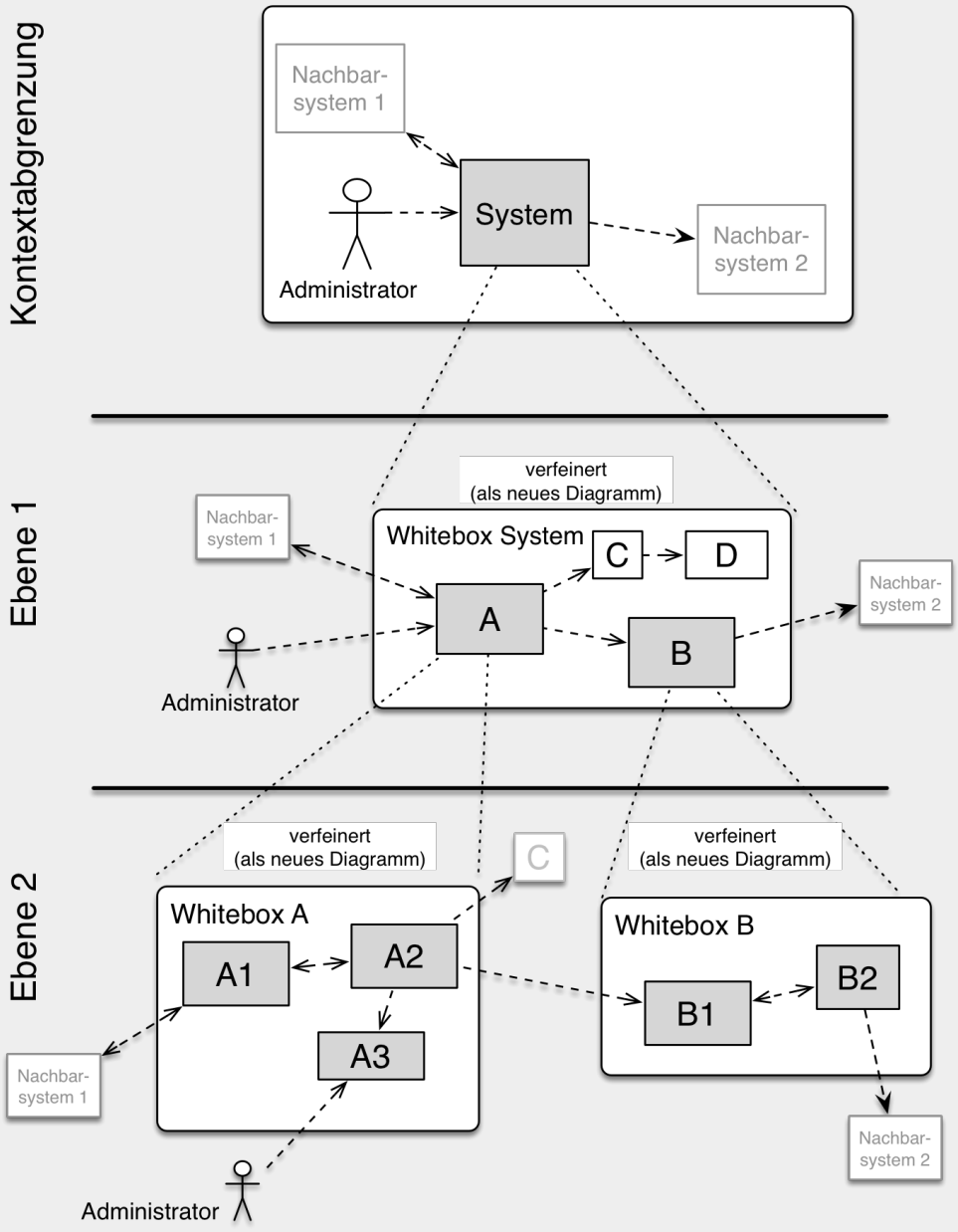
Motivation

Behalten Sie den Überblick über den Quellcode, indem Sie die statische Struktur des Systems durch Abstraktion verständlich machen.

Damit ermöglichen Sie Kommunikation auf abstrakterer Ebene, ohne zu viele Implementierungsdetails offenlegen zu müssen.

Form

Die Bausteinsicht ist eine hierarchische Sammlung von Blackboxen und Whiteboxen (siehe Abbildung unten) und deren Beschreibungen.



Ebene 1 ist die Whitebox-Beschreibung des Gesamtsystems, zusammen mit Blackbox-Beschreibungen der darin enthaltenen Bausteine.

Ebene 2 zoomt in einige Bausteine der Ebene 1 hinein. Sie enthält somit die Whitebox-Beschreibungen ausgewählter Bausteine der Ebene 1, jeweils zusammen mit Blackbox-Beschreibungen darin enthaltener Bausteine.

Ebene 3 zoomt in einige Bausteine der Ebene 2 hinein, usw.

5.1. Whitebox Gesamtsystem

An dieser Stelle beschreiben Sie die Zerlegung des Gesamtsystems anhand des nachfolgenden Whitebox-Templates. Dieses enthält:

- Ein Übersichtsdiagramm
- die Begründung dieser Zerlegung
- Blackbox-Beschreibungen der hier enthaltenen Bausteine. Dafür haben Sie verschiedene Optionen:
  - in *einer* Tabelle, gibt einen kurzen und pragmatischen Überblick über die enthaltenen Bausteine sowie deren Schnittstellen.
  - als Liste von Blackbox-Beschreibungen der Bausteine, gemäß dem Blackbox-Template (siehe unten). Diese Liste können Sie, je nach Werkzeug, etwa in Form von Unterkapiteln (Text), Unter-Seiten (Wiki) oder geschachtelten Elementen (Modellierungswerkzeug) darstellen.
- (optional:) wichtige Schnittstellen, die nicht bereits im Blackbox-Template eines der Bausteine erläutert werden, aber für das Verständnis der Whitebox von zentraler Bedeutung sind. Aufgrund der vielfältigen Möglichkeiten oder Ausprägungen von Schnittstellen geben wir hierzu kein weiteres Template vor. Im schlimmsten Fall müssen Sie Syntax, Semantik, Protokolle, Fehlerverhalten, Restriktionen, Versionen, Qualitätseigenschaften, notwendige Kompatibilitäten und vieles mehr spezifizieren oder beschreiben. Im besten Fall kommen Sie mit Beispielen oder einfachen Signaturen zurecht.

<Übersichtsdiagramm>

Begründung

<Erläuternder Text>

Enthaltene Bausteine

<Beschreibung der enthaltenen Bausteine (Blackboxen)>

Wichtige Schnittstellen

<Beschreibung wichtiger Schnittstellen>

Hier folgen jetzt Erläuterungen zu Blackboxen der Ebene 1.

Falls Sie die tabellarische Beschreibung wählen, so werden Blackboxen darin nur mit Name und Verantwortung nach folgendem Muster beschrieben:

Name	Verantwortung
<Blackbox 1>	*<Text>*
<Blackbox 2>	*<Text>*

Falls Sie die ausführliche Liste von Blackbox-Beschreibungen wählen, beschreiben Sie jede wichtige Blackbox in einem eigenen Blackbox-Template. Dessen Überschrift ist jeweils der Namen dieser Blackbox.

5.1.1. <Name Blackbox 1>

Beschreiben Sie die <Blackbox 1> anhand des folgenden Blackbox-Templates:

- Zweck/Verantwortung
- Schnittstelle(n), sofern diese nicht als eigenständige Beschreibungen herausgezogen sind. Hierzu gehören eventuell auch Qualitäts- und Leistungsmerkmale dieser Schnittstelle.
- (Optional) Qualitäts-/Leistungsmerkmale der Blackbox, beispielsweise Verfügbarkeit, Laufzeitverhalten o. Ä.
- (Optional) Ablageort/Datei(en)
- (Optional) Erfüllte Anforderungen, falls Sie Traceability zu Anforderungen benötigen.
- (Optional) Offene Punkte/Probleme/Risiken

<Zweck/Verantwortung>

<Schnittstelle(n)>

<(Optional) Qualitäts-/Leistungsmerkmale>

<(Optional) Ablageort/Datei(en)>

<(Optional) Erfüllte Anforderungen>

<(optional) Offene Punkte/Probleme/Risiken>

5.1.2. <Name Blackbox 2>

<Blackbox-Template>

5.1.3. <Name Blackbox n>

<Blackbox-Template>

5.1.4. <Name Schnittstelle 1>

...

5.1.5. <Name Schnittstelle m>

5.2. Ebene 2

> Beschreiben Sie den inneren Aufbau (einiger) Bausteine aus Ebene 1 als Whitebox.
>
> Welche Bausteine Ihres Systems Sie hier beschreiben, müssen Sie selbst entscheiden. Bitte stellen Sie dabei Relevanz vor Vollständigkeit. Skizzieren Sie wichtige, überraschende, riskante, komplexe oder besonders volatile Bausteine. Normale, einfache oder standardisierte Teile sollten Sie weglassen.

5.2.1. Whitebox *<Baustein 1>*

> ...zeigt das Innenleben von *Baustein 1*.

<Whitebox-Template>

5.2.2. Whitebox *<Baustein 2>*

<Whitebox-Template>

...

5.2.3. Whitebox *<Baustein m>*

<Whitebox-Template>

5.3. Ebene 3

Beschreiben Sie den inneren Aufbau (einiger) Bausteine aus Ebene 2 als Whitebox.

Bei tieferen Gliederungen der Architektur kopieren Sie diesen Teil von arc42 für die weiteren Ebenen.

5.3.1. Whitebox <_Baustein x.1_>

...zeigt das Innenleben von *Baustein x.1*.

<Whitebox-Template>

5.3.2. Whitebox <_Baustein x.2_>

<Whitebox-Template>

5.3.3. Whitebox <_Baustein y.1_>

<Whitebox-Template>

6. Laufzeitsicht

Diese Sicht visualisiert im Gegensatz zur statischen Bausteinsicht dynamische Aspekte. Wie spielen die Teile zusammen?

Inhalt

Diese Sicht erklärt konkrete Abläufe und Beziehungen zwischen Bausteinen in Form von Szenarien aus den folgenden Bereichen:

- Wichtige Abläufe oder *Features*: Wie führen die Bausteine der Architektur die wichtigsten Abläufe durch?
- Interaktionen an kritischen externen Schnittstellen: Wie arbeiten Bausteine mit Nutzern und Nachbarsystemen zusammen?
- Betrieb und Administration: Inbetriebnahme, Start, Stop.
- Fehler- und Ausnahmeszenarien

Anmerkung: Das Kriterium für die Auswahl der möglichen Szenarien (d.h. Abläufe) des Systems ist deren Architekturrelevanz. Es geht nicht darum, möglichst viele Abläufe darzustellen, sondern eine angemessene Auswahl zu dokumentieren.

Motivation

Sie sollten verstehen, wie (Instanzen von) Bausteine(n) Ihres Systems ihre jeweiligen Aufgaben erfüllen und zur Laufzeit miteinander kommunizieren.

Nutzen Sie diese Szenarien in der Dokumentation hauptsächlich für eine verständlichere Kommunikation mit denjenigen Stakeholdern, die die statischen Modelle (z.B. Bausteinsicht, Verteilungssicht) weniger verständlich finden.

Form

Für die Beschreibung von Szenarien gibt es zahlreiche Ausdrucksmöglichkeiten. Nutzen Sie beispielsweise:

- Nummerierte Schrittfolgen oder Aufzählungen in Umgangssprache
- Aktivitäts- oder Flussdiagramme
- Sequenzdiagramme
- BPMN (Geschäftsprozessmodell und -notation) oder EPKs (Ereignis-

Prozessketten)

- Zustandsautomaten
- …

6.1. *<Bezeichnung Laufzeitszenario 1>*

- <hier Laufzeitdiagramm oder Ablaufbeschreibung einfügen>
- <hier Besonderheiten bei dem Zusammenspiel der Bausteine in diesem Szenario erläutern>

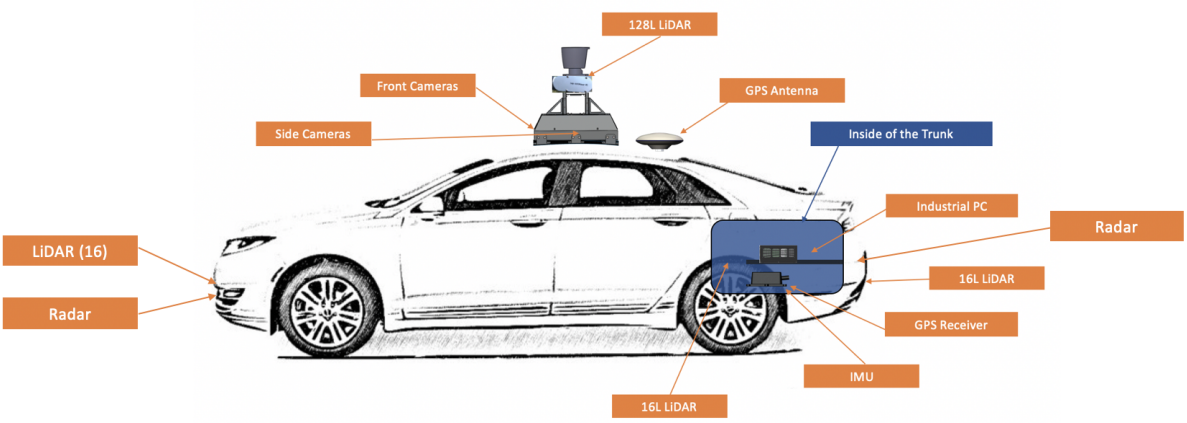
6.2. *<Bezeichnung Laufzeitszenario 2>*

…

6.3. *<Bezeichnung Laufzeitszenario n>*

…

7. Verteilungssicht



Inhalt

Die Verteilungssicht beschreibt:

1. die technische Infrastruktur, auf der Ihr System ausgeführt wird, mit Infrastrukturelementen wie Standorten, Umgebungen, Rechnern, Prozessoren, Kanälen und Netztopologien sowie sonstigen Bestandteilen, und
2. die Abbildung von (Software-)Bausteinen auf diese Infrastruktur.

Häufig laufen Systeme in unterschiedlichen Umgebungen, beispielsweise Entwicklung-/Test- oder Produktionsumgebungen. In solchen Fällen sollten Sie alle relevanten Umgebungen aufzeigen.

Nutzen Sie die Verteilungssicht insbesondere dann, wenn Ihre Software auf mehr als einem Rechner, Prozessor, Server oder Container abläuft oder Sie Ihre Hardware sogar selbst konstruieren.

Aus Softwaresicht genügt es, auf die Aspekte zu achten, die für die Softwareverteilung relevant sind. Insbesondere bei der Hardwareentwicklung kann es notwendig sein, die Infrastruktur mit beliebigen Details zu beschreiben.

Motivation

Software läuft nicht ohne Infrastruktur. Diese zugrundeliegende Infrastruktur beeinflusst Ihr System und/oder querschnittliche Lösungskonzepte, daher müssen Sie diese Infrastruktur kennen.

Form

Das oberste Verteilungsdiagramm könnte bereits in Ihrem technischen Kontext enthalten sein, mit Ihrer Infrastruktur als EINE Blackbox. Jetzt zoomen Sie in diese Infrastruktur mit weiteren Verteilungsdiagrammen hinein:

- Die UML stellt mit Verteilungsdiagrammen (Deployment diagrams) eine Diagrammart zur Verfügung, um diese Sicht auszudrücken. Nutzen Sie diese, evtl. auch geschachtelt, wenn Ihre Verteilungsstruktur es

verlangt.

- Falls Ihre Infrastruktur-Stakeholder andere Diagrammarten bevorzugen, die beispielsweise Prozessoren und Kanäle zeigen, sind diese hier ebenfalls einsetzbar.

7.1. Infrastruktur Ebene 1

An dieser Stelle beschreiben Sie (als Kombination von Diagrammen mit Tabellen oder Texten):

- die Verteilung des Gesamtsystems auf mehrere Standorte, Umgebungen, Rechner, Prozessoren o. Ä., sowie die physischen Verbindungskanäle zwischen diesen,
- wichtige Begründungen für diese Verteilungsstruktur,
- Qualitäts- und/oder Leistungsmerkmale dieser Infrastruktur,
- Zuordnung von Softwareartefakten zu Bestandteilen der Infrastruktur

Für mehrere Umgebungen oder alternative Deployments kopieren Sie diesen Teil von arc42 für alle wichtigen Umgebungen/Varianten.

*<**Übersichtsdiagramm**>*

Begründung

<Erläuternder Text>

Qualitäts- und/oder Leistungsmerkmale

<Erläuternder Text>

Zuordnung von Bausteinen zu Infrastruktur

<Beschreibung der Zuordnung>

7.2. Infrastruktur Ebene 2

An dieser Stelle können Sie den inneren Aufbau (einiger) Infrastrukturelemente aus Ebene 1 beschreiben.

Für jedes Infrastrukturelement kopieren Sie die Struktur aus Ebene 1.

7.2.1. *<Infrastrukturelement 1>*

<Diagramm + Erläuterungen>

7.2.2. *<Infrastrukturelement 2>*

<Diagramm + Erläuterungen>

...

7.2.3. *<Infrastrukturelement n>*

<Diagramm + Erläuterungen>

8. Querschnittliche Konzepte

Dieser Abschnitt beschreibt allgemeine Strukturen und Aspekte, die systemweit gelten. Darüber hinaus stellt er verschiedene technische Lösungskonzepte vor.

Inhalt

Dieser Abschnitt beschreibt übergreifende, prinzipielle Regelungen und Lösungsansätze, die an mehreren Stellen (=*querschnittlich*) relevant sind.

Solche Konzepte betreffen oft mehrere Bausteine. Dazu können vielerlei Themen gehören, beispielsweise:

- fachliche Modelle,
- eingesetzte Architektur- oder Entwurfsmuster,
- Regeln für den konkreten Einsatz von Technologien,
- prinzipielle — meist technische — Festlegungen übergreifender Art,
- Implementierungsregeln

Motivation

Konzepte bilden die Grundlage für *konzeptionelle Integrität* (Konsistenz, Homogenität) der Architektur und damit eine wesentliche Grundlage für die innere Qualität Ihrer Systeme.

Manche dieser Themen lassen sich nur schwer als Baustein in der Architektur unterbringen (z.B. das Thema „Sicherheit“). Hier ist der Platz im Template, wo Sie derartige Themen geschlossen behandeln können.

Form

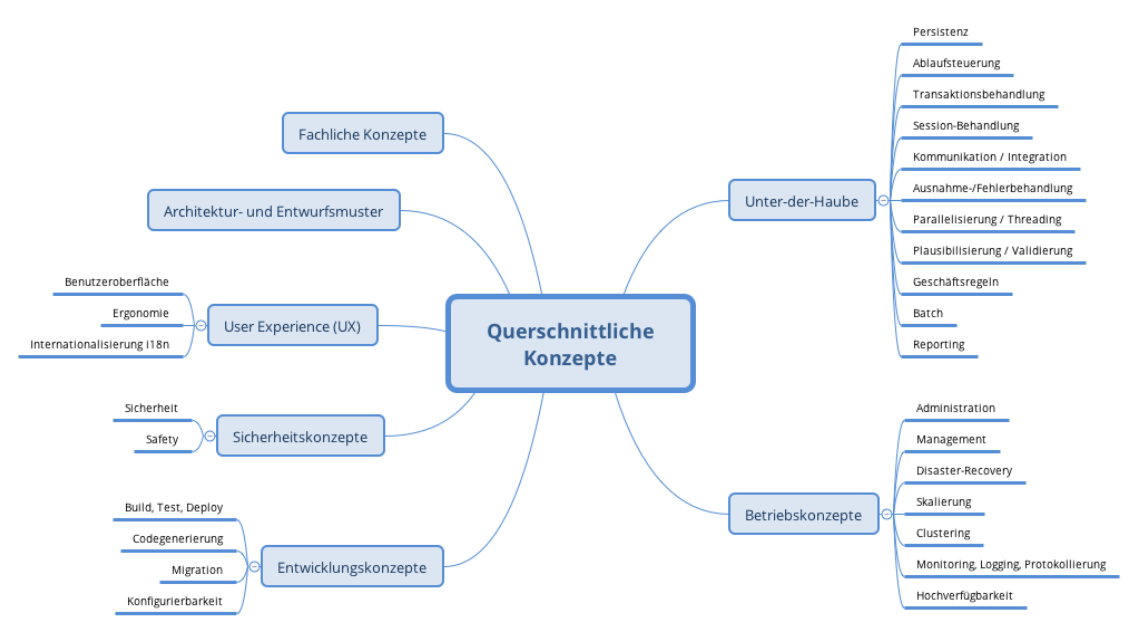
Kann vielfältig sein:

- Konzeptpapiere mit beliebiger Gliederung,
- übergreifende Modelle/Szenarien mit Notationen, die Sie auch in den Architektursichten nutzen,
- beispielhafte Implementierung speziell für technische Konzepte,
- Verweise auf „übliche“ Nutzung von Standard-Frameworks (beispielsweise die Nutzung von Hibernate als Object/Relational Mapper).

Struktur

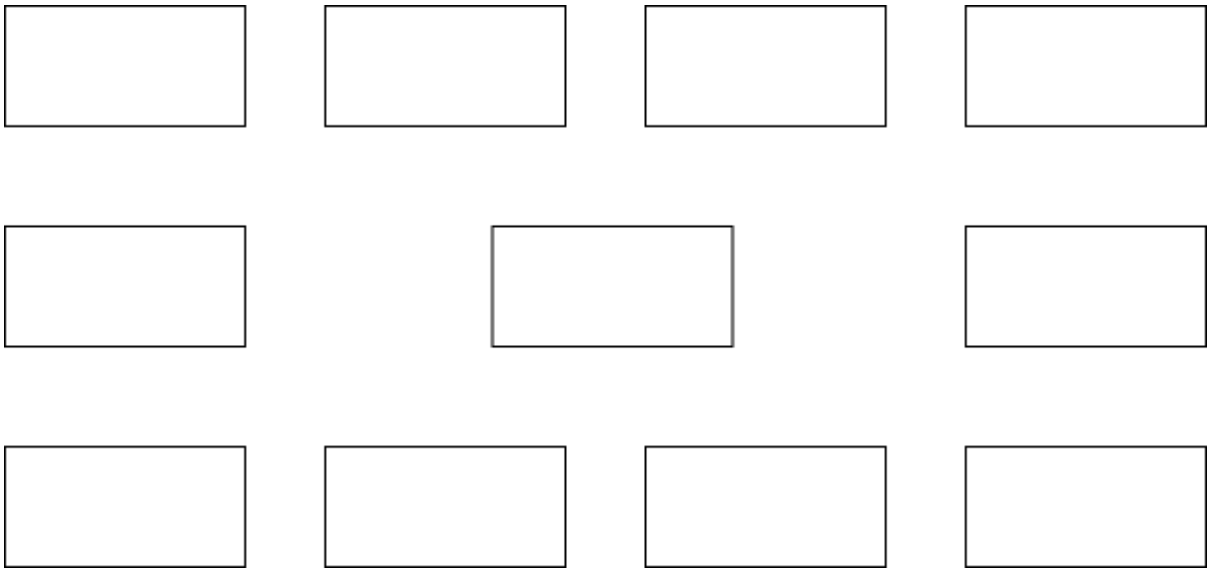
Eine mögliche (nicht aber notwendige!) Untergliederung dieses Abschnittes könnte wie folgt aussehen (wobei die Zuordnung von Themen zu den Gruppen nicht immer eindeutig ist)

- Fachliche Konzepte
- User Experience (UX)
- Sicherheitskonzepte (Safety und Security)
- Architektur- und Entwurfsmuster
- Unter-der-Haube
- Entwicklungskonzepte
- Betriebskonzepte



8.1. *<Konzept 1>*

<Erklärung>



8.2. *<Konzept 2>*

<Erklärung>

...

8.3. *<Konzept n>*

<Erklärung>

9. Entwurfsentscheidungen

Inhalt

Wichtige, teure, große oder riskante Architektur- oder Entwurfsentscheidungen inklusive der jeweiligen Begründungen. Mit "Entscheidungen" meinen wir hier die Auswahl einer von mehreren Alternativen unter vorgegebenen Kriterien.

Wägen Sie ab, inwiefern Sie Entscheidungen hier zentral beschreiben, oder wo eine lokale Beschreibung (z.B. in der Whitebox-Sicht von Bausteinen) sinnvoller ist. Vermeiden Sie Redundanz. Verweisen Sie evtl. auf Abschnitt 4, wo schon grundlegende strategische Entscheidungen beschrieben wurden.

Motivation

Stakeholder des Systems sollten wichtige Entscheidungen verstehen und nachvollziehen können.

Form

Verschiedene Möglichkeiten:

- Liste oder Tabelle, nach Wichtigkeit und Tragweite der Entscheidungen geordnet
- ausführlicher in Form einzelner Unterkapitel je Entscheidung
- ADR (Architecture Decision Record) für jede wichtige Entscheidung

10. Qualitätsanforderungen

Dieser Abschnitt beinhaltet konkrete Qualitätsszenarien, welche die zentralen Qualitätsziele, aber auch andere geforderte Qualitätseigenschaften besser fassen. Sie ermöglichen es, Entscheidungsoptionen zu bewerten.

Inhalt

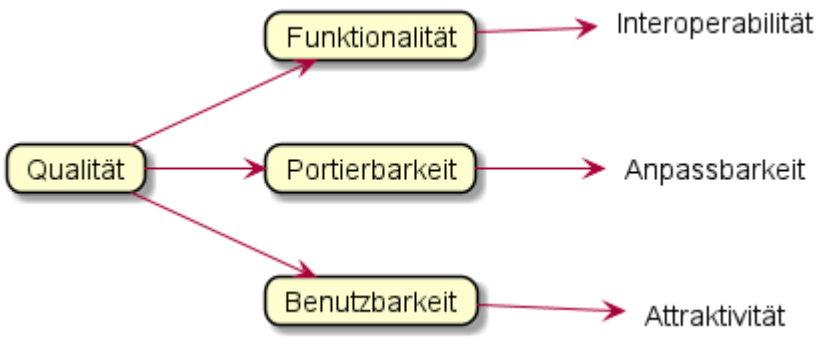
Dieser Abschnitt enthält möglichst alle Qualitätsanforderungen als Qualitätsbaum mit Szenarien. Die wichtigsten davon haben Sie bereits in Abschnitt 1.2 (Qualitätsziele) hervorgehoben.

Nehmen Sie hier auch Qualitätsanforderungen geringerer Priorität auf, deren Nichteinhaltung oder -erreichung geringe Risiken birgt.

Motivation

Weil Qualitätsanforderungen die Architekturentscheidungen oft maßgeblich beeinflussen, sollten Sie die für Ihre Stakeholder relevanten Qualitätsanforderungen kennen, möglichst konkret und operationalisiert.

10.1. Qualitätsbaum



Inhalt

Der Qualitätsbaum (à la ATAM) mit Qualitätsszenarien an den Blättern.

Motivation

Die mit Prioritäten versehene Baumstruktur gibt Überblick über die — oftmals zahlreichen — Qualitätsanforderungen.

Form

- Baumartige Verfeinerung des Begriffes „Qualität“, mit „Qualität“ oder „Nützlichkeit“ als Wurzel.
- Mindmap mit Qualitätsoberbegriffen als Hauptzweige

In jedem Fall sollten Sie hier Verweise auf die Qualitätsszenarien des folgenden Abschnittes aufnehmen.

10.2. Qualitätsszenarien

Inhalt

Konkretisierung der (in der Praxis oftmals vagen oder impliziten) Qualitätsanforderungen durch (Qualitäts-)Szenarien.

Diese Szenarien beschreiben, was beim Eintreffen eines Stimulus auf ein System in bestimmten Situationen geschieht.

Wesentlich sind zwei Arten von Szenarien:

- Nutzungsszenarien (auch bekannt als Anwendungs- oder Anwendungsfallszenarien) beschreiben, wie das System zur Laufzeit auf einen bestimmten Auslöser reagieren soll. Hierunter fallen auch Szenarien zur Beschreibung von Effizienz oder Performance. Beispiel: Das System beantwortet eine Benutzeranfrage innerhalb einer Sekunde.
- Änderungsszenarien beschreiben eine Modifikation des Systems oder seiner unmittelbaren Umgebung. Beispiel: Eine zusätzliche Funktionalität wird implementiert oder die Anforderung an ein Qualitätsmerkmal ändert sich.

Motivation

Szenarien operationalisieren Qualitätsanforderungen und machen deren Erfüllung mess- oder entscheidbar.

Insbesondere wenn Sie die Qualität Ihrer Architektur mit Methoden wie ATAM überprüfen wollen, bedürfen die in Abschnitt 1.2 genannten Qualitätsziele einer weiteren Präzisierung bis auf die Ebene von diskutierbaren und nachprüfbaren Szenarien.

Form

Entweder tabellarisch oder als Freitext.

Das folgende Bild gibt einen Überblick über die relevanten Qualitätsmerkmale und den ihnen jeweils zugeordneten Szenarien.

11. Risiken und technische Schulden

Die folgenden Risiken wurden zu Beginn des Vorhabens identifiziert. Sie beeinflussten die Planung der ersten drei Iterationen maßgeblich. Seit Abschluss der dritten Iteration werden sie beherrscht. Dieser Architekturüberblick zeigt die Risiken inklusive der damaligen Eventualfallplanung weiterhin, wegen ihres großen Einflusses auf die Lösung.

Inhalt

Eine nach Prioritäten geordnete Liste der erkannten Architekturrisiken und/oder technischen Schulden.

Motivation

> Risikomanagement ist Projektmanagement für Erwachsene.
>
> — Tim Lister, Atlantic Systems Guild

Unter diesem Motto sollten Sie Architekturrisiken und/oder technische Schulden gezielt ermitteln, bewerten und Ihren Management-Stakeholdern (z.B. Projektleitung, Product-Owner) transparent machen.

Form

Liste oder Tabelle von Risiken und/oder technischen Schulden, eventuell mit vorgeschlagenen Maßnahmen zur Risikovermeidung, Risikominimierung oder dem Abbau der technischen Schulden.

12. Glossar

Das folgende Glossar erklärt Begriffe aus dem Bereich Autonomes Fahren.

Inhalt

Die wesentlichen fachlichen und technischen Begriffe, die Stakeholder im Zusammenhang mit dem System verwenden.

Nutzen Sie das Glossar ebenfalls als Übersetzungsreferenz, falls Sie in mehrsprachigen Teams arbeiten.

Motivation

Sie sollten relevante Begriffe klar definieren, so dass alle Beteiligten

- diese Begriffe identisch verstehen, und
- vermeiden, mehrere Begriffe für die gleiche Sache zu haben.

Form

- Zweispaltige Tabelle mit <Begriff> und <Definition>
- Eventuell weitere Spalten mit Übersetzungen, falls notwendig.

Begriff	**Definition**
By-Wire	*Bezeichnung für (zumindest partielles) Fahren oder Steuern von Fahrzeugen ohne mechanische Kraftübertragung der Bedienelemente zu den entsprechenden Stellelementen wie etwa Drosselklappen. Das By-Wire-Konzept umfasst dabei zumindest zwei oder mehr der „X-by-Wire“- Systeme wie etwa Brake-by-Wire (Bremssteuerung) und Steer-by-Wire (Lenkung)..*
LIDAR	*ist eine dem Radar verwandte Methode zur optischen Abstands- und Geschwindigkeitsmessung sowie zur Fernmessung. Statt der Radiowellen wie beim Radar werden Laserstrahlen verwendet.*

Begriff	Definition
RADAR	*ist die Bezeichnung für verschiedene Erkennungs- und Ortungsverfahren und -geräte auf der Basis elektromagnetischer Wellen im Radiofrequenzbereich (Funkwellen).*

13. Anhang

13.1. Apollo

13.2. Arc42

- die Dokumentation muss bereits zu Beginn der Arbeiten an der Architektur/am Quellcode erfolgen um stets den aktuellen Stand abbilden zu können.
- die Dokumentation soll mit dem Projekt wachsen!
- die nachträgliche Dokumentation ist eine Sysiphusarbeit
    - Beispiel:
        - Diagramme in der Laufzeitsicht sind effizienter zu gestalten während ein Modul im Entstehen ist. Änderungen können mit jeder Entwicklungsstufe eingepflegt werden.
        - Ein fertiges Modul in ein Diagramm zu überführen erfordert viel Zeit, da alle beteiligten Module und deren Funktionsweise erst identifiziert werden müssen.

13.3. docToolChain

- plantUML ist zwar bei sehr einfachen Diagrammen eine Erleichterung, kommt aber an beispielsweise draw.io oder tikz (in Latex) nicht heran wenn es um komplexe und detailierte Diagramme geht
    - teilweise werden bei der inline-Erstellung bei plantUML Diagrammeinstellungen nicht übernommen
- Schriftgrößen sind nicht einstellbar, was an manchen Stellen - z.B Tabellen nötig gewesen wäre
- Ansonsten ist docToolChain eine interessante Alternative, besonders in Bezug auf den CI-Task in github.
    - damit nicht bei jedem Commit die Dokumentation neu erstellt wird,

bietet github die Möglichkeit über Commitnachrichten, z.B "[no ci]" oder "[skip ci]" den CI-Task auszusetzen. Hier wäre es sinnvoller über Commitnachrichten explizit die Generierung zu starten.