ApolloAuto

Inhaltsverzeichnis

1. Einführung und Ziele . . . 3
   1.1. Aufgabenstellung . . . 3
   1.2. Qualitätsziele . . . 4
   1.3. Stakeholder . . . 5
2. Randbedingungen . . . 6
   2.1. Technische Randbedingungen . . . 6
   2.2. Organisatorische Randbedingungen . . . 7
   2.3. Konventionen . . . 7
3. Kontextabgrenzung . . . 8
   3.1. Fachlicher Kontext . . . 8
   3.2. Technischer Kontext . . . 9
4. Lösungsstrategie . . . 10
5. Bausteinsicht . . . 11
   5.1. Whitebox Gesamtsystem . . . 11
6. Laufzeitsicht . . . 13
   6.1. *<Bezeichnung Laufzeitszenario 1>* . . . 13
   6.2. *<Bezeichnung Laufzeitszenario 2>* . . . 14
   6.3. *<Bezeichnung Laufzeitszenario n>* . . . 14
7. Verteilungssicht . . . 15
   7.1. Infrastruktur Ebene 1 . . . 15
   7.2. Infrastruktur Ebene 2 . . . 16
8. Querschnittliche Konzepte . . . 18
   8.1. *<Konzept 1>* . . . 20
   8.2. *<Konzept 2>* . . . 20
   8.3. *<Konzept n>* . . . 20
9. Entwurfsentscheidungen . . . 21
10. Qualitätsanforderungen . . . 22
   10.1. Qualitätsbaum . . . 22
   10.2. Qualitätsszenarien . . . 22
11. Risiken und technische Schulden . . . 24
12. Glossar . . . 25

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

1. Einführung und Ziele

Dieser Abschnitt beschreibt die wesentlichen Anforderungen und treibenden Kräfte, die bei der Umsetzung der Softwarearchitektur und Entwicklung des Systems berücksichtigt werden müssen.

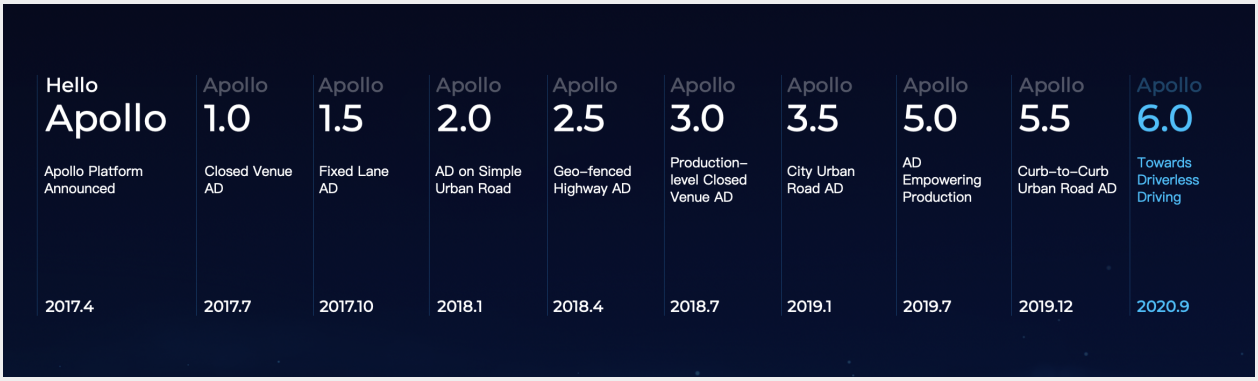
Dazu gehören:

- zugrunde liegende Geschäftsziele,
- wesentliche Aufgabenstellungen und
- essenzielle fachliche Anforderungen an das System sowie
- Qualitätsziele für die Architektur und
- relevante Stakeholder und deren Erwartungshaltung.

1.1. Aufgabenstellung

Was ist ApolloAuto?

Apollo ist eine leistungsstarke und flexible Architektur für autonome Fahrzeuge, die eine schnelle Entwicklung und Tests von diesen ermöglicht. Dessen API ist dabei gänzlich öffentlich einsehbar. Die Ziele der einzelnen Apollo Releases seit dem Start im April können der folgenden Grafik entnommen werden.



Mit diesem Ansatz soll die Grundlage für Implementierungen des autonomen Fahrens geschaffen werden. Langfristig wir Level 5 autonomes Fahren verfolgt.

Die vorliegende Architektur soll verschiedene Anwendungen ermöglichen. Einige mit entscheidenen Kernfunktionen werden in folgender Tabelle aufgelistet.

Use Case	**Erläuterung**
Apollo Go Robotaxi	*Ein autonom fahrender Taxiservice, welcher derzeit Level 4 autonomes Fahren beherrschen soll*
Apollo V2X	*Ein intelligentes Transportsystem für die Erfassung und Verarbeitung von Straßendaten*
Valet Parking	*Das Fahrzeug übernimmt die Parkplatzsuche und den eigentlichen Einparkvorgang in beispielsweise Parkhäusern*

1.2. Qualitätsziele

Qualitätsziel	**Erläuterung**
Schnelle Entwicklung autonomer Fahrzeuge	_ _
Schnelle Tests von autonomen Fahrzeugen	_ _
Flexible Architektur	_ _
Verständliche Darstellung	*Entwickler, die nicht bei Baidu arbeiten, müssen mit dem vorliegenden Code und den gegebenen Beschreibungen des Systems zurechtkommen*

1.3. Stakeholder

Rolle	Erwartungshaltung
Baidu	*Weiterentwicklung des autonomen Fahrens und von V2X-Anwendungen*
Unabhängige Entwickler	*Ziel ist die Entwicklung von Tools, um Produkte für das autonome Fahren zu entwickeln. Die Einarbeitung in das System muss dafür ermöglicht werden.*

2. Randbedingungen

2.1. Technische Randbedingungen

Die technischen Randbedingungen beziehen sich auf die aktuelle Version Apollo 6.0. Es ist sinnvoll, zu unterscheiden, welche Anforderungen an die CPU für die Installation und welche an das Auto gestellt werden.

Tabelle 1. Installationsanforderungen

Randbedingung	**Erläuterung**
Prozessor und RAM	*8-Kern Prozessor mit mindestens 16GB Arbeitsspeicher*
GPU	*Eine NVIDIA Turing GPU wird empfohlen*
Betriebssystem	*Ubuntu 18.04*
GPU-Treiber	*NVIDIA 455.32.00 Treiber sowie alle nachfolgenden Versionen* (Nvidia Treiber)
Docker	*Docker-CE 19.03 sowie alle nachfolgenden Versionen* (Installation von Docker auf Ubuntu)
NVIDIA Container Toolkit	*NVIDIA 455.32.00 Treiber sowie alle nachfolgenden Versionen* (NVIDIA-Docker Github Link)

Tabelle 2. Anforderungen an das korrespondierende Fahrzeug

Randbedingung	**Erläuterung**
Drive-by-Wire	*Fahren und Steuern der Fahrzeuge muss ohne mechanische Kraftübertragung möglich sein. Dazu gehören Lenkung, Bremsen, Gas und Schaltvorgänge.*
Sensorik	*Mehrere LiDAR- und Radar-Sensoren zur besseren Auffassung der Umgebung*
Kameras	*Seitliche und Frontkameras zur Umgebungsaufnahme*
GPS	*GPS-Antenne und Receiver zur Lokalisierung des Fahrzeugs*
IMU	*Inertiale Messeinheit zur Aufnahme verschiedener Sensordaten zum Fahrzeug*

Der Aufbau eines solchen Fahrzeugs kann dem nachfolgenden Bild entnommen werden.

image::FahrzeuganforderungenApolloAuto.png

Die Spezifikationen anderer Versionen können der Github Seite entnommen werden.

2.2. Organisatorische Randbedingungen

Randbedingung	Erläuterung
Team	*Entwicklerteam von Baidu, ggfs. stark auf den chinesischen Markt spezialisiert*
Zeitplan	*Updates mit Abständen von Monaten bis zu einem Jahr*
Vorgehensmodell	*Mit den Updates verbunden sind Erweiterungen der Funktionalität der Software und Integration von zusätzlicher Hardware.*
Veröffentlichung	*Der Installationsordner von ApolloAuto ist frei zugänglich auf einem GitHub Server einsehbar.*

2.3. Konventionen

Konvention	Erläuterung
Dokumentation	*Aktuelle Dokumentation erfolgt über Readme-Dateien in den dazugehörigen Projektordnern.*
<Rolle-2>	*<Kontakt-2>*

3. Kontextabgrenzung

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

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>

4. Lösungsstrategie

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

5.1. Whitebox Gesamtsystem

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

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>

6. Laufzeitsicht

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>

Unabhängig von der Installation, welche implementiert werden soll, muss stets zunächst die Apollo

Version 1.0 installiert werden.

8. Querschnittliche Konzepte

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

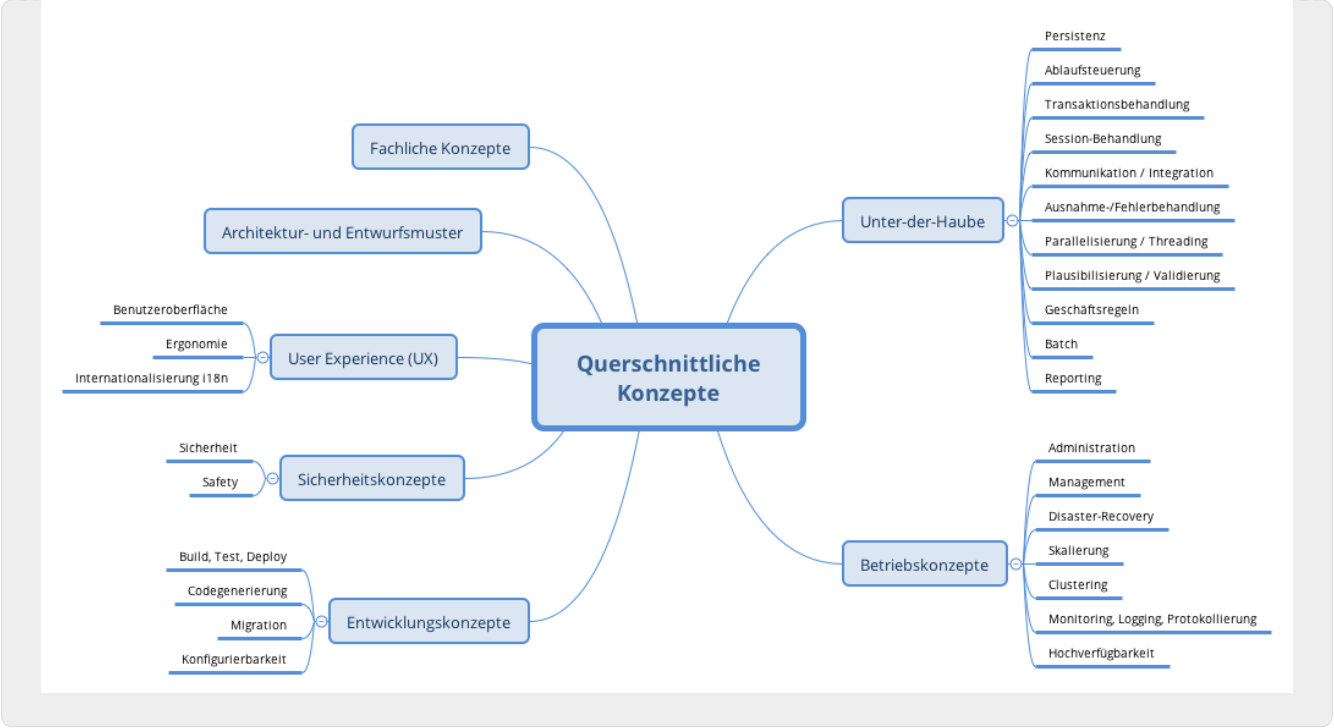
Kann vielfältig sein:

- 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

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.

11. Risiken und technische Schulden

=== Komplizierte Anbindung

Es müssen hardwareseitig alle Voraussetzungen genau erfüllt werden, damit das Setup funktionieren kann. (?-→) Für die Entwicklung des Gesamtsystems ist es daher von großer Bedeutung, sicherzustellen, dass durch Weiterentwicklungen dieses komplexe System nicht funktionsunfähig wird.

12. Glossar

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**
<Begriff-1>	*<Definition-1>*
<Begriff-2	*<Definition-2>*