# Einführung und Ziele

* ***zugrunde liegende Geschäftsziele:***
* Kostengünstige Ride-Sharing App
* Benutzer mit nahegelegenen Fahrern verbinden, und dadurch
* Anzahl der Autos auf der Straße verringern
* Alternative Transportationsmöglichkeit für Benutzer die keinen Zugang zu einem Auto haben
* ***wesentliche Aufgabenstellungen:***
* Entwicklung einer iOS / Android App
* Zahlungssystem
* Echtzeit-Updates (Verfügbarkeit von Fahrten, Ankunftszeiten…)
* ***essenzielle fachliche Anforderungen an das System:***
* Benutzerregistrierung & Profilerstellung
* Fahrtensuche / Buchung
* Echtzeit-Tracking (von Fahrten & jeglichen Updates)
* Verifizierungsprozess (Fahrer/Mitfahrer)
* Bewertungs-/Rezensionssystem
* ***Qualitätsziele für die Architektur:***
* Sicherheit & Datenschutz (Zahlungsdetails, persönliche Informationen)
* Skalierbarkeit (wachsende Nutzeranzahl)
* Performance (keine Verzögerungen)
* Kompatibilität (iOS/Android)
* Zuverlässigkeit (bei Ausfall 🡪 Wiederherstellungsfähigkeit)
* ***relevante Stakeholder und deren Erwartungshaltung:***
* Maryam Patel (Investor): Entwicklungsplan, gute Rendite
* Raj Gupta (Geschäftsentwicklung): App soll lokalen Transportbedürfnissen entsprechen
* Megan Chen (UX-Designerin): Benutzerfreundliches Design
* Amirah Rahman (Fahrervertreter): App soll Bedürfnissen & Anforderungen der Fahrer gerecht werden
* Javier Gomez (Umweltaktivist): Reduzierung von Emissionen/Verkehr

## Aufgabenstellung

**Inhalt:**

Die RideShare-App soll eine mobile Applikation sein, die Benutzer mit nahegelegenen Fahrern für Mitfahrgelegenheiten verbindet. Sie soll nicht nur kostengünstiger sein, sondern gleichzeitig auch Emissionen und Verkehr reduzieren.

**Motivation:**

Durch immer mehr Verkehr und den damit verbundenen Umweltauswirkungen besteht immer mehr Bedarf an Alternativen für den Personenverkehr. RideShare soll nicht nur die Verkehrsbelastung reduzieren, sondern auch Personen die keinen Zugang zu einem Auto haben die Möglichkeit geben, für persönliche Zwecke eines benutzen zu können (ohne eines kaufen zu müssen).

***Use-Cases:***

## Use Case 1: Benutzerregistrierung

Ein neuer Benutzer kann sich registrieren und ein Profil erstellen

## Use Case 2: Fahrtensuche und Buchung

Ein Benutzer kann nach verfügbaren Fahrten suchen und eine Fahrt buchen

## Use Case 3: Echtzeit-Tracking

Benutzer können den Standort des Fahrzeugs in Echtzeit verfolgen

## Use Case 4: Zahlungsabwicklung

Nach Abschluss der Fahrt kann der Benutzer über die App bezahlen

## Qualitätsziele

|  |  |
| --- | --- |
| Priorität 1: Sicherheit | Szenario: Ein Benutzer will auf private Daten eines anderen Nutzers zugreifen aber wird durch das System daran gehindert. |
| Priorität 2: Performance | Szenario: Ein Benutzer fordert eine Fahrt an – die Antwortzeit unserer App soll unter 3 Sekunden betragen. |
| Priorität 3: Benutzerfreundlichkeit | Ein neuer Benutzer sollte es schaffen, innerhalb von 10 Minuten nach dem runterladen problemlos durch die App navigieren zu können. |
| Priorität 4: Skalierbarkeit | Szenario: Nach einem guten Marketing, sollte das System problemlos tausende neue Benutzer innerhalb einer Stunde aushalten können. |
| Priorität 5: Zuverlässigkeit | Szenario: Bei einem Netzwerkausfall sollte das System innerhalb von 2 Minuten wieder problemlos funktionieren. |

## Stakeholder

|  |  |  |
| --- | --- | --- |
| Rolle | Kontakt | Erwartungshaltung |
| *Investor* | *maryam.patel@venturecapital.xyz* | *Skalierbare Architektur* |
| *Business-Development*  *UX-Designer*  *Fahrervertreter*  Umweltaktivist | *raj.gupta@lta.com*  [*megan.chen@creativeagency.abc*](mailto:megan.chen@creativeagency.abc)  [*amirah.rahman@rideshare.drivers*](mailto:amirah.rahman@rideshare.drivers)  *javier.gomez@sustainable.xyz* | *Benutzerfreundliche Architektur*  *Flexible Architektur*  *Zuverlässige Architektur*  *Umweltfreundliche Architektur* |

# Randbedingungen

|  |  |
| --- | --- |
| Technische Randbedingungen | Kompatibilität mit iOS und Android |
| Organisatorische Randbedingungen | Einhaltung von Datenschutzrechten |
| Politische Randbedingungen | Einhaltung der Verkehrsregeln |

# Kontextabgrenzung

Passagier:

* Eingabe: Anfrage für eine Fahrt (Ort, Ziel), Buchungsanfrage, Zahlungsart
* Ausgabe: Verfügbare Fahrer/Autos, Bestätigung (der Fahrt & Zahlung), Fahrtdetails

Fahrer:

* Eingabe: Fahrtenangebot, Annahme/Ablehnung von Fahrt
* Ausgabe: Auszahlung, Buchungsanfrage, Buchungsbestätigung

*Schnittstellen zu Nachbarsystemen:*

Zahlungssystem:

* Eingabe: Zahlungsanfrage, Zahlungsart (PayPal, Bankomatkarte, Guthaben)
* Ausgabe: Zahlungsstatus, Fehlermeldungen, Guthaben

Technische Schnittstellen:

* Internetverbindung: Datenübertragung
* GPS: Standortbestimmung
* Protokolle: HTTPS/Sockets für Kommunikation & Datenübertragung
* Smartphones: Für Benutzung der App erforderlich
* Server: Datenbank & Backend-Logik

## Fachlicher Kontext

|  |  |  |
| --- | --- | --- |
| ***Kommunikationsbeziehung*** | ***Eingabe*** | ***Ausgabe*** |
| Mitfahrer / Passagier | Suchanfrage für Fahrten, Buchungsanfrage, Zahlungsdetails | Liste von verfügbaren Fahrern in der Nähe, Buchungsbestätigung, Zahlungsbestätigung, Fahrtdetails |
| Fahrer | Fahranfragen, Annahme/Ablehnung | Buchungsanfragen, Auszahlung, Buchungsbestätigung |
| Zahlungssystem | Zahlungsanfrage, Details, Zahlungsart | Zahlungsstatus, Guthaben, Fehlermeldungen |

## Technischer Kontext

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Technische Schnittstelle | Eingabe | Ausgabe | Übertragungsmedium | Protokoll |
| Internet (4G/5G/Wi-Fi) | Suchanfrage für Fahrten, Buchungsanfrage, Annahme/Ablehnung, Zahlung | Verfügbare Fahrten, Bestätigung der Buchung und Zahlung, Zahlungsstatus | Drahtlos | HTTPS |
| GPS | Standort des Passagiers/Fahrers | Standort des Fahrers für den Passagier, Standort des Passagiers für den Fahrer | Satellit | NMEA 0183 |
| Benachrichtigungs-system | Buchungsanfrage, Zahlungsbestätigung | Buchungsbestätigung, Annahme/Ablehnung | Drahtlos | FCM / APNs |

***Mapping für die technischen Schnittstellen:***

* Internet:
* Suchanfrage für eine Fahrt: App sendet Anfrage an Server und findet eine verfügbare Fahrt
* Buchungsanfrage: Auswahl einer Fahrt 🡪 Buchungsanfrage an Server
* Zahlungsanfrage: Nach einer Fahrt 🡪 Zahlungsanfrage an Zahlungsprozessor
* GPS:
* Standort Passagier: App erfasst Standort 🡪 Server (verfügbare Fahrten finden)
* Standort Fahrer: App erfasst Standort 🡪 informiert Passagier über Ankunftszeit
* Benachrichtigungssystem:
* Buchungsanfrage: Fahrer erhält Benachrichtigung
* Buchungsbestätigung: Fahrer akzeptiert eine Anfrage 🡪 Passagier erhält Bestätigung

# Lösungsstrategie

Technologieentscheidungen:

* Frontend: React (plattformübergreifende Mobilität)
* Backend: Node.js (Realtime Verarbeitung der Daten)
* Zahlungssystem: Stripe (Sicherheit, Benutzerfreundlichkeit)
* Datenbank: MongoDB (Skalierbarkeit)

Top-Level Zerlegung des Systems:

* Model-View-Controller (MVC) für Trennung von UI, Logik und Daten
* Client-Server Architektur

Qualitätsanforderungen:

* Sicherheit: 2-Schritt-Authentifizierung (z.B. OAuth) und JWT für Datenübertragung
* Skalierbarkeit: Cloud-basierte Datenbank
* Benutzerfreundlichkeit: Intuitives UX-Design

Organisatorisches:

* Agile Entwicklungsmethode: iteratives Feedback nach jedem Schritt
* Aufgabeneinteilung: Spezialisierte Teams arbeiten an verschiedenen Tasks

# 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.

## 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.

### <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>*

### <Name Blackbox 2>

*<Blackbox-Template>*

### <Name Blackbox n>

*<Blackbox-Template>*

### <Name Schnittstelle 1>

…

### <Name Schnittstelle m>

## 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.

### Whitebox *<Baustein 1>*

…zeigt das Innenleben von *Baustein 1*.

*<Whitebox-Template>*

### Whitebox *<Baustein 2>*

*<Whitebox-Template>*

…

### Whitebox *<Baustein m>*

*<Whitebox-Template>*

## 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.

### Whitebox <\_Baustein x.1\_>

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

*<Whitebox-Template>*

### Whitebox <\_Baustein x.2\_>

*<Whitebox-Template>*

### Whitebox <\_Baustein y.1\_>

*<Whitebox-Template>*

# 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
* …

## *<Bezeichnung Laufzeitszenario 1>*

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

## *<Bezeichnung Laufzeitszenario 2>*

…

## *<Bezeichnung Laufzeitszenario n>*

…

# 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.

## 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>*

## 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.

### *<Infrastrukturelement 1>*

*<Diagramm + Erläuterungen>*

### *<Infrastrukturelement 2>*

*<Diagramm + Erläuterungen>*

…

### *<Infrastrukturelement n>*

*<Diagramm + Erläuterungen>*

# 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



## *<Konzept 1>*

*<Erklärung>*

## *<Konzept 2>*

*<Erklärung>*

…

## *<Konzept n>*

*<Erklärung>*

# 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](http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions)) für jede wichtige Entscheidung

# 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.

## 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.

* 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.

## 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.

# Risiken und technische Schulden

**Inhalt.**

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

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.

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

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