**Einführung und Ziele**

Programmierung eines Prototyps für eine Smartphone-Spiele App mit folgenden Eckpunkten:

- Chain Reaction Game

- Touch-Bedienung

- System: Android (+iOS)

- Umsetzung aus der Unreal Engine

Der Prototyp soll voll funktionsfähig sein und als Basis - Konzept für ein finales, ausgereifteres Produkt dienen.

Nach Veröffentlichung sind Tests für verschiedene Monetarisierungsstrategien geplant.

- Micro-Transactions

- diverse Werbebanner und -clips anderer Spiele

- Ingame Währung Verdienstmöglichkeiten

Der Leitsatz der Monetarisierungsstrategien lautet: "Never Pay-to-Win".

**Aufgabenstellung**

- Prototyp Mobile App

- Programmiersprache: C++

- Touch-Screen Technologie kennenlernen und implementieren

Bei einem Chain Reaction Spiel werden auf einem Spielfeld Objekte aktiviert/angeregt, die wiederum die Fähigkeit besitzen weitere Objekte in ihrer Reichweite aktivieren/anregen zu können (z.B. durch Explosion, Streuung, Ausbreitung, etc.). Nach Aktivierung verschwinden die Objekte in der Regel. Ziel ist ein freies Spielfeld. Schwierigkeitsgrade können z.B. durch Zeitlimits, Punktelimits, verschiedene Objekt-Eigenschaften erzeugt werden.

**Qualitätsziele**

- vollständige und einwandfreie Funktionalität

- Design, Grafiken, Animationen ist zwar wichtig, wird allerdings zunächst vernachlässigt.

- Zunächst wenig Content, dieser wird zu einem späteren Zeitpunkt sukzessiv aufgebaut.

**Stakeholder**

**Aktuell**

Programmierer:

- Maximilian Weller

- Philipp Haag

- Tom-Oliver Marek

**Späterer Zeitpunkt**

Design / Art:

-

Marketing / Vertrieb:

-

Investoren:

-

Musik / Sound:

-

**Randbedingungen**

**Technische Randbedingungen:**

\* Der Zugang erfolgt mittels Spiele-App (Google Playstore)

\* Implementierung in C++

\* Quelltexterstellung Visual Studio 2017 Community Version 15.7.4

\* Umsetzung über Unreal Engine Version 4.20.00

\* Git wird für die weitere Zusammenarbeit genutzt

**Organisatorische Randbedingungen:**

\* Team: Maximilian Weller, Philipp Haag, Tom-Oliver Marek

\* Beginn der Entwicklung: 30.08.2018

\* Erste grafische Vorführung: <offen> (kein Zeitlimit)

\* Erster vorzeigbarer Prototyp: <offen> (kein Zeitlimit)

**Finanzielle Randbedingungen:**

**Kontextabgrenzung**

**Inhalt.**

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

Differenzieren Sie fachlichen Kontext (fachliche Ein- und Ausgaben) und technischen Kontext (Kanäle, Protokolle, Hardware), falls nötig.

**Motivation.**

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

**Form.**

Verschiedene Optionen:

* Diverse Kontextdiagramme
* Listen von Kommunikationspartnern mit deren Schnittstellen

**Fachlicher Kontext**

**Inhalt.**

Festlegung **aller** Kommunikationspartner (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 Black Box darstellen und die fachlichen Schnittstellen zu den Nachbarn beschreiben.

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

**<Diagramm und/oder Tabelle>**

**<optional: Erläuterung der externen fachlichen Schnittstellen>**

**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 Infrastruktur- oder Hardwareentwickler entscheiden auch über diese technischen Schnittstellen.

**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>**

**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
* relevante organisatorische Entscheidungen, beispielsweise für bestimmte Entwicklungsprozesse oder Delegation bestimmter Aufgaben an andere Stakeholder.

**Motivation.**

Diese allerwichtigsten Entscheidungen bilden wesentliche „Eckpfeiler“ der Architektur. Von ihnen hängen meistens 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. Verweisen Sie eher auf weitere Ausführungen in Folgeabschnitten.

**Bausteinsicht**

**Inhalt.**

Diese Sicht zeigt die statische Zerlegung des Systems in Bausteine (Module, Komponenten, Subsysteme, Klassen, Interfaces, Pakete, Bibliotheken, Frameworks, Schichten, Partitionen, Tiers, Funktionen, Makros, Operationen, Datenstrukturen…) sowie deren Beziehungen.

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-Templates 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 enhaltenen 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>**

An dieser Stelle beschreiben Sie die <Blackbox 1> anhand des folgenden Blackbox-Templates:

* Zweck/Verantwortung
* Schnittstelle(n), sofern sie nicht als eigenständige Beschreibungen herausgezogen sind. Hierzu gehören eventuell auch Qualitäts- und Leistungsmerkmale dieser Schnittstelle.
* (Optional) Qualitäts-/Leistungsmerkmale der Blackbox, beispielsweise Verfügbarkeit, Laufzeitverhalten…
* (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**

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

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**

An dieser Stelle können Sie den inneren Aufbau (einiger) Bausteine aus Ebene 2 als Whitebox beschreiben.

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 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: 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 solche Szenarien in der Dokumentation hauptsächlich zur besseren Kommunikation mit Stakeholdern, die statische 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 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:

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

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

Nutzen Sie die Verteilungssicht insbesondere, 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. Hardwarearchitekten können bei Bedarf die Infrastruktur mit beliebigen Details beschreiben.

**Motivation.**

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

Das oberste Verteilungsdiagramm könnte bereits in Ihrem technischen Kontext enthalten sein, mit Ihrer Infrastruktur als EINE Black-Box. 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 Prozessoren und Kanäle zeigen, sind die 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 oä. sowie die physischen Verbindungskanäle zwischen diesen,
* wichtige Begründungen für dieser Verteilungsstruktur,
* Qualitäts- und/oder Leistungsmerkmale dieser Infrastruktur,
* Zuordnung von Softwareartefakten zu Bestandteilen der Infrastruktur

Für mehrere Umgebungen oder alternatives Deployment kopieren Sie diesen Teil von arc42 für alle wichtigen Umgebungen.

***<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 (=*querschittlich*) 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 für insbesondere technische Konzepte,
* Verweise auf "übliche" Nutzung von Standardframeworks (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 ( a 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 Q-Oberbegriffen als Hauptzweige

In jedem Fall sollten Sie hier Verweise auf die Szenarien 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 für die meisten Softwarearchitekten sind zwei Arten von Szenarien:

* Nutzungsszenarien (auch genannt 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 unmittelbarer 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.

**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 Risiko 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>* |

- Entwicklungsumgebung: Unreal Engine Version 4.20.2

- Git Bash / Git Hub

<https://git-scm.com/book/en/v2>