Universität Bremen
FB 3 – Informatik
Dr. Karsten Hölscher
TutorIn: Euer/Eure TutorIn

Software-Projekt 2 – SoSe 2020

VAK 03-BA-901.02

Architekturbeschreibung

Clara Maria Odinius odinius@uni-bremen.de
Habib Mergan habib1@uni-bremen.de
Kevin Santiago Rey Rodriguez kev_rey@uni-bremen.de
Liam Hurwitz hurwitz@uni-bremen.de
Mehmet Ali Baykara baykara@uni-bremen.de
Miguel Alejandro Caceres Pedraza mcaceres@uni-bremen.de

Abgabe: TT. Monat JJJJ — Version 1.1

Inhaltsverzeichnis

Version und Änderungsgeschichte

Die aktuelle Versionsnummer des Dokumentes sollte eindeutig und gut zu identifizieren sein, hier und optimalerweise auf dem Titelblatt.

Version	Datum	Änderungen
0.1	TT.MM.JJJJ	Dokumentvorlage als initiale Fassung kopiert
0.2	TT.MM.JJJJ	...
...		

1 Einführung

1.1 Zweck

Was ist der Zweck dieser Architekturbeschreibung? Wer sind die LeserInnen?

1.2 Status

1.3 Definitionen, Akronyme und Abkürzungen

Begriff	Definition
Software-Architektur	ist die grundlegende Organisation eines Systems verkörpert
Architektursicht (View)	Repräsentation eines ganzen Systems aus der Perspektive einer kohärenten Menge von Anliegen (IEEE P1471, 2002).
Anwendungsfälle	Spezifiziert eine beliebige Menge von Aktionen, die ein System ausführen muss, damit ein Resultat stattfindet, welches für mindestens einen Akteur von Bedeutung ist.
Framework	Programmiergerüst, welches den Rahmen der Anwendung bildet. Es umfasst Bibliotheken und Komponenten.
UML	Steht für Unified Modeling Language und ist eine grafische Modellierungssprache zur Spezifikation, Visualisierung, Konstruktion und Dokumentation von Modellen für Softwaresysteme.
IDE	integrierte Entwicklungsumgebung - Hilft bei der Bearbeitung von Projekten in der Softwareentwicklung
Versionskontrolle	Hochgeladenen Versionen werden festgehalten und können wieder hergestellt werden
Maven	Java Programme können standardisiert und verwaltet werden
Library	Sammlung von verschiedensten vorgefertigten Methoden oder Klassen
Interface	Schnittstelle
Multiplayer	Spielmodus, mit mehr als einem Spieler

Singleplayer	Spielmodus, mit einem Spieler, der gegen einen computergesteuerten Gegner spielt.
JUnit	Frameworke für Tests für die Programmiersprache Java
Javadoc	Software-Dokumentationswerkzeug für die Programmiersprache Java.
JavaEE	(Java Platform Enterprise Edition) Eine Spezifikation für die transaktionsbasierte Ausführung von in Java programmierten Anwendungen.
GUI	Steht für Graphical User Interface und kennzeichnet eine grafische Schnittstelle, über die ein Mensch mit einer Software interagiert.
Paketdiagramm	Strukturdiagramm der UML, stellt die Verbindung zwischen Paketen, Paketimports bzw. Verschmelzungen und deren Abhängigkeiten dar.
Problemkarte	Beschreiben Probleme im Zusammengang der Einflussfaktoren und stellen Lösungen bzw. entsprechende Strategien dar.
Sequenzdiagramm	Strukturdiagramm der UML, stellt den Austausch von Nachrichten zwischen Objekten mittels einer Lebenslinie dar.
Klassendiagramm	Strukturdiagramm der UML, stellt die statischen Strukturen eines Systems dar.
Komponentendiagramm	Strukturdiagramm der UML, stellt Komponenten und deren Schnittstellen dar.
HTTP	Steht für Hypertext Transfer Protocol, einem zustandslosen Protokoll zum synchronen Versenden von Informationen über Rechnernetze.
HTTPS	Durch Verschlüsselungstechniken gesichertes HTTP.
TCP	Das Transmission Control Protocol ist ein Netzwerkprotokoll, das definiert, auf welche Art und Weise Daten zwischen Netzwerkkomponenten ausgetauscht werden sollen.
UDP	Das User Datagram Protocol, kurz UDP, ist ein minimales, verbindungsloses Netzwerkprotokoll, das zur Transportschicht der Internetprotokollfamilie gehört.

1.4 Referenzen

Vorlesungsfolien

1.5 Übersicht über das Dokument

2 Anwendungsfälle

2.1 Anmeldung

Ein User offen das spielt und dann kommt ein Fenster, wo der User sich mit password und name einloggen kann, wenn er die richtige Daten eintragt. dann das Spielt kann weiter geführt werden. Wenn die Daten nicht richtig sind, dann das User bekommt ein Hinweis, dass er wieder eintragen soll.

2.2 Multi-Spieler

Wenn das Users angemeldet ist, dann er sehen kann, welche der andere User auch verbindet sind und starten ein Spiel mit der anderen User oder gegen Computer.

2.3 Spielt-Starten

Im ersten Fenster des Spiels kann der Benutzer das Schiff, die Waffen, die er verwenden möchte, die Besatzung und den Schwierigkeitsgrad des Spiels auswählen. danach kann er anfangen.

2.4 Spielt-Verlauft

Wenn der Spieler das Spiel beginnt, kann er die Crew in den von ihm gewählten Positionen positionieren. Dann können Sie die Sprungtaste drücken, um die Zielplaneten auszuwählen. Der Benutzer kann dies immer in jeder Runde wiederholen.

Wenn der Benutzer einen Zielplaneten auswählt, kann er auf Gegner treffen und nach einem Dialogfeld mit ihnen kämpfen.

Sobald der Benutzer besiegt hat, kann der Gegner sein Schiff reparieren, seine Crew heilen oder einen Sprung machen.

2.5 Kampf-Regeln

Wenn der Benutzer in einen Kampf verwickelt ist, unterliegt dieser bestimmten Regeln.

- Wenn ein Kampf beginnt, kann der Benutzer das Ziel seiner Waffen auswählen, die die Abschnitte des gegnerischen Schiffes sind.
- Waffen brauchen Zeit zum Laden und Schießen.
- Damit eine Waffe das Ziel treffen kann, muss der Gegner Schilde deaktiviert haben.

- Um die Schilde zu deaktivieren, müssen die Waffen ihn treffen. Jedes Mal, wenn er einen Aufprall findet, verliert der Schild an Stärke. Sobald die Stärke des Schildes Null ist, wird der Schild deaktiviert.
- Waffen treffen nicht immer das Ziel.
- Damit eine Waffe verwendet werden kann, muss der Waffenbereich des Schiffes funktionsfähig sein.
- Jedes Mal, wenn eine Waffe ein Schiff trifft, verliert sie ihren Lebenspunkt.
- Sobald eines der beiden Schiffe keine Lebenspunkte mehr hat, verliert dieses Schiff.

2.6 Pausenmodus

Während eines Kampfes kann der Benutzer das Spiel in den Pausenmodus versetzen. Dies bedeutet, dass beide Schiffe aufhören zu schießen und die Besatzung aufhört, sich zu bewegen. Wenn das Spiel fortgesetzt wird, werden die Waffen wieder aktiviert und auch die Crew.

3 Globale Analyse

3.1 Einflussfaktoren

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
O1 : Organisation				
O1.1 Time-To-Market				
	Die Auslieferung erfolgt am 02.08.2020.	Keine Veränderlichkeit oder Flexibilität, da Vorgaben bestehen.	- -/ - -	Nicht alle Funktionen können realisiert werden bzw. implementiert werden.
O1.2 Architektur-Abgabe				
	Die Auslieferung erfolgt am 31.05.2020.	Keine Veränderlichkeit oder Flexibilität, da Vorgaben bestehen.	- -/ - -	Durch den Zeitdruck könnte die Architektur mangelhaft werden. Wenn wir uns nicht genug Zeit lassen, könnten Aspekte, die relevant für die Architektur sind, vergessen werden.
O1.3 Entwickler				
	Die Projektgruppe besteht aus 6 Entwicklern	Falls ein Entwickler (temporär) ausfällt kann ein anderer Entwickler einspringen. Keine neuen Entwickler können eingestellt werden. Gruppenmitglieder können wegfallen oder austreten	o/ o	Die Architektur kann wegen Zeitmangel (es können nicht mehr als die ursprünglichen sechs Entwickler mitarbeiten) und fehlenden Fähigkeiten Mängel enthalten. Unvollständige Implementierung droht
O1.4 Fähigkeiten Entwickler				
	Nicht alle Entwickler haben die gleiche Programmier-erfahrung	Hohe Veränderlichkeit und Flexibilität durch Ausführen des Projekts und Recherche. TODO!	++/ ++	Die Implementierung kann Mängel enthalten.

O1.5 Teamarbeit in Corona-Zeiten				
	Das persönliche Treffen kann nicht stattfinden, aufgrund von Kontakt-beschränkung	Digitale Treffen über z.B. Discord. Keine Veränderlichkeit aufgrund der Gesetzeslage.	++/ - -	Missverständnisse können öfter auftreten. Teamarbeit könnte erschwert werden und dadurch die Qualität der der Architektur beeinträchtigen

Abge-leitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
T1: Technik				
T1.1: Programmiersprache				
	Java 8 oder höher ist vorgegeben.	Die Programmier-sprache wird vorgegeben. Wir können aber zwischen verschiedenen Versionen auswählen.	- -/ o	Das Architektur muss in Java umgesetzt werden.
T1.2 Betriebssystem				
	Die Anwendung muss auf den gängigen Betriebssystemen (MacOS, Windows, Linux) laufen.	Keine Veränderlichkeit oder Flexibilität, da Mindestanforderungen bestehen. Aber wir können noch weitere Betriebssysteme hinzufügen	- -/ - -	Bei der Implementierung müssen die gängigen Betriebssysteme berücksichtigt werden. Wenn die Anwendung auf mehr Betriebssystemen laufen soll, dann muss mehr Zeit investiert werden.
T1.3 Client-Server-Architektur TODO!				
	Zur Implementierung muss JavaEE 11 benutzt werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Das Projekt muss komplett in Java umgesetzt werden.

T1.4: Framework libgdx TODO!				
	Als Framework zur Erstellung der Oberfläche muss JSF verwendet werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Das Projekt muss in Java umgesetzt werden.
T1.5: Persistenz TODO!				
	Zur sicheren Speicherung der Daten soll die relationalen Datenbank, leichtgewichtige verwendet werden. z.B (H2, SQLite, Derby)	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Bei der Implementierung muss eine leichtgewichtige, relationale Datenbank (ohne Installation verwendbar) verwendet werden.
T1.6: Build System TODO!				
	Maven oder Gradle müssen als Build-System verwendet werden.	Auswahl zwischen Maven und Gradle möglich. Keine Veränderlichkeit möglich, da Mindestanforderungen bestehen.	+/ - -	Das Projekt muss Maven/Gradle-build fähig sein.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
T1.8: Multiple Users TODO!				
	Die Anwendung muss von mehreren Benutzern gleichzeitig verwendbar sein.	Keine Flexibilität oder Veränderlichkeit, da Mindestanforderungen bestehen. Wir können aber es ermöglichen, dass mehr als zwei Spieler gleichzeitig spielen.	- -/ - -	Die Anwendung darf nicht von gleichzeitiger Verwendung von zwei Nutzern überfordert sein. Trotzdem müssen mindestens 2 Spieler gleichzeitig das Spiel spielen können, um die Mindest-anforderungen zu erfüllen.
P1: Produktfaktoren				
P1.1.1: Pause				
	Die Anwendung muss in der Lage sein, ein zuvor von einem bestimmten Benutzer gestartetes Spiel zu speichern und dem Benutzer die Möglichkeit zu geben, es fortzusetzen.	Keine Veränderlichkeit.	- -/ ++	Das Spiel muss über diese Funktionalitä verfügen, da dies eine Mindestanforderung ist.
P1.8: Schwierigkeitsstufen				
	Für Benutzer müssen unter-schiedliche Schwierig-keitsgrade implementiert werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Anwendung darf nicht von gleichzeitiger Verwendung von mehreren Nutzern überfordert sein; Trotzdem müssen mindestens 2 Spieler gleichzeitig das Spielt benutzen zu können, um die Minimal Anforderungen zu erfullen.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.2: Struktur und Eingeschäften des Raumschiff				
P1.2.1: Aufteilung in Sektionen				
	Das Raumschiff muss mindestens die folgenden Abschnitte haben: Antrieb, Waffen, Schilde	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ -	Das Ausfüllen von Abschnitten die für das Spiel nicht erforderlich sind, kann zu möglichen Verzögerungen und Fristen führen.
P1.2.2: Schäden der Sektionen				
	Jeder Abschnitt kann beschädigt werden, wodurch das von ihm gesteuerte System beschädigt wird.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die mögliche Implementierung viele Abschnitte kann zu Problemen bei der Bereitstellung und Entwicklung führen
P1.3: Ressourcen des Spiel				
	Ressourcen müssen implementiert werden, um die Spiellogik auszuführen, zum Beispiel: Geld Energie Hüllenintegrität	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Implementierungslogik dieser Ressourcen und ihre Beziehung zu anderen Entitäten können Implementierungsprobleme und Probleme bei der Bereitstellung verursachen.
P1.3: Raumschiff Eigenschaften				
	Jedes Raumschiff muss Eigenschaften haben, die vom Benutzer geändert werden müssen.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Implementierungslogik dieser Eigenschaften und ihre Beziehung zu anderen Entitäten können Implementierungsprobleme und Probleme bei der Bereitstellung verursachen.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.2: Besatzung				
P1.2.1: Raumschiff hat Besatzung				
	Die Schiffsbesatzung kann vom Benutzer auf die gleiche Weise ausgewählt werden. Die Besatzung hat einen direkten Einfluss auf die Schiffssysteme.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ -	Die Interaktion der Besatzung mit verschiedenen Einheiten der Anwendung kann die Durchführung des Projekts erschweren
P1.2.2: Besatzung Eigenschaften				
	Die Besatzung des Schiffes kann: Kann Systeme/ Räume bewegen kann sterben kann eingestellt werden	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Dieser Faktor kann die Komplexität des Spiels beeinflussen und die Entwicklung der Anwendung verlangsamen.
P1.3: Besatzung Fähigkeiten				
	Die Qualitäten der Besatzung können verbessert werden und beeinflussen die Systeme des Schiffes, auf dem sie arbeiten.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Implementierungslogik dieser Ressourcen und ihre Beziehung zu anderen Entitäten können Implementierungsprobleme und Probleme bei der Bereitstellung verursachen.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.2: Universum				
P1.2.1: Struktur des Universums				
	Das Universum muss Stationen und Planeten haben, die vom Raumschiff des Spielers durchquert werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ -	Die Implementierung dieser Funktionalität kann sich auf die Komplexität des Spiels auswirken und die Bereitstellung verzögern.
P1.2.2: Station/Planet Eigenschaften				
	Jede Station oder jeder Planet muss Aktionen haben, die das Spiel negativ oder positiv beeinflussen müssen.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Dieser Faktor kann die Komplexität des Spiels beeinflussen und die Entwicklung der Anwendung verlangsamen.
P1.3: feindliche Schiffe				
	Im Universum müssen Sie generische feindliche Schiffe haben, deren Verhalten unterschiedliche Abschnitte und Verhaltensweisen aufweist, die von der künstlichen Intelligenz abgeleitet sind.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Implementierungslogik dieser Gegner und ihre Beziehung zu anderen Entitäten können Implementierungsprobleme und Probleme bei der Bereitstellung verursachen.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.2: Struktur des Kampfes				
P1.2.1: taktische Entscheidungen				
	In jeder Runde kann der Benutzer seine Art wählen, das gegnerische Schiff anzugreifen	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ -	Die Implementierung dieser Funktionalität kann sich auf die Komplexität des Spiels auswirken und die Bereitstellung verzögern.
P1.2.2: Waffenverhalten				
	Der Benutzer kann die Abschnitte auswählen, die angegriffen werden sollen, und die Waffen müssen geladen sein, um diese Schüsse auszuführen.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Dieser Faktor kann die Komplexität des Spiels beeinflussen und die Entwicklung der Anwendung verlangsamen.
P1.3: Verteilen von Besatzungsmitgliedern				
	Während eines Kampfes kann die Position der Besatzung auf dem Schiff gewählt werden, die Abschnittsänderungen werden einige Zeit in Anspruch nehmen.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Implementierungslogik dieser Gegner und ihre Beziehung zu anderen Entitäten können Implementierungsprobleme und Probleme bei der Bereitstellung verursachen.

3.2 Probleme und Strategien

Aus einer Menge von Faktoren ergeben sich Probleme, die nun in Form von Problemkarten beschrieben werden. Diese resultieren z. B. aus

- *Grenzen oder Einschränkungen durch Faktoren*
- *der Notwendigkeit, die Auswirkung eines Faktors zu begrenzen*

- *der Schwierigkeit, einen Produktfaktor zu erfüllen, oder*
- *der Notwendigkeit einer allgemeinen Lösung zu globalen Anforderungen.*

Dazu entwickelt Ihr Strategien, um mit den identifizierten Problemen umzugehen.

Achtet auch hier darauf, dass die Probleme und Strategien wirklich die Architektur betreffen und nicht etwa das Projektmanagement. Die Strategien stellen im Prinzip die Designentscheidungen dar. Sie sollten also die Erklärung für den konkreten Aufbau der verschiedenen Sichten liefern.

Beschreibt möglichst mehrere Alternativen und gebt an, für welche Ihr Euch letztlich aus welchem Grunde entschieden habt. Natürlich müssen die genannten Strategien in den folgenden Sichten auch tatsächlich umgesetzt werden!

Ein sehr häufiger Fehler ist es, dass SWP-Gruppen arbeitsteilig vorgehen: die eine Gruppe schreibt das Kapitel zur Analyse von Faktoren und zu den Strategien, die andere Gruppe beschreibt die diversen Sichten, ohne dass diese beiden Gruppen sich abstimmen. Natürlich besteht aber ein Zusammenhang zwischen den Faktoren, Strategien und Sichten. Dieser muss erkennbar sein, indem sich die verschiedenen Kapitel eindeutig aufeinander beziehen.

4 Konzeptionelle Sicht

Diese Sicht beschreibt das System auf einer hohen Abstraktionsebene, d. h. mit sehr starkem Bezug zur Anwendungsdomäne und den geforderten Produktfunktionen und -attributen. Sie legt die Grobstruktur fest, ohne gleich in die Details von spezifischen Technologien abzugleiten. Sie wird in den nachfolgenden Sichten konkretisiert und verfeinert. Die konzeptionelle Sicht wird mit UML-Komponentendiagrammen visualisiert.

5 Modulsicht

Diese Sicht beschreibt den statischen Aufbau des Systems mit Hilfe von Modulen, Subsystemen, Schichten und Schnittstellen. Diese Sicht ist hierarchisch, d. h. Module werden in Teilmodule zerlegt. Die Zerlegung endet bei Modulen, die ein klar umrissenes Arbeitspaket für eine Person darstellen und in einer Kalenderwoche implementiert werden können. Die Modulbeschreibung der Blätter dieser Hierarchie muss genau genug und ausreichend sein, um das Modul implementieren zu können.

Die Modulsicht wird durch UML-Paket- und Klassendiagramme visualisiert.

Die Module werden durch ihre Schnittstellen beschrieben. Die Schnittstelle eines Moduls M ist die Menge aller Annahmen, die andere Module über M machen dürfen, bzw. jene Annahmen, die M über seine verwendeten Module macht (bzw. seine Umgebung, wozu auch Speicher, Laufzeit etc. gehören). Konkrete Implementierungen dieser Schnittstellen

sind das Geheimnis des Moduls und können vom Programmierer festgelegt werden. Sie sollen hier dementsprechend nicht beschrieben werden.

Die Diagramme der Modulsicht sollten die zur Schnittstelle gehörenden Methoden enthalten. Die Beschreibung der einzelnen Methoden (im Sinne der Schnittstellenbeschreibung) geschieht allerdings per Javadoc im zugehörigen Quelltext. Das bedeutet, dass Ihr für alle Eure Module Klassen, Interfaces und Pakete erstellt und sie mit den Methoden der Schnittstellen verseht. Natürlich noch ohne Methodenrümpfe bzw. mit minimalen Rümpfen. Dieses Vorgehen vereinfacht den Schnittstellenentwurf und stellt Konsistenz sicher.

Jeder Schnittstelle liegt ein Protokoll zugrunde. Das Protokoll beschreibt die Vor- und Nachbedingungen der Schnittstellenelemente. Dazu gehören die erlaubten Reihenfolgen, in denen Methoden der Schnittstelle aufgerufen werden dürfen, sowie Annahmen über Eingabeparameter und Zusicherungen über Ausgabeparameter. Das Protokoll von Modulen wird in der Modulsicht beschrieben. Dort, wo es sinnvoll ist, sollte es mit Hilfe von Zustands- oder Sequenz-diagrammen spezifiziert werden. Diese sind dann einzusetzen, wenn der Text allein kein ausreichendes Verständnis vermittelt (insbesondere bei komplexen oder nicht offensichtlichen Zusammenhängen).

Der Bezug zur konzeptionellen Sicht muss klar ersichtlich sein. Im Zweifel sollte er explizit erklärt werden. Auch für diese Sicht muss die Entstehung anhand der Strategien erläutert werden.

6 Datensicht

Hier wird das der Anwendung zugrundeliegende Datenmodell beschrieben. Hierzu werden neben einem erläuternden Text auch ein oder mehrere UML-Klassendiagramme verwendet. Das hier beschriebene Datenmodell wird u. a. jenes der Anforderungsspezifikation enthalten, allerdings mit implementierungsspezifischen Änderungen und Erweiterungen. Siehe die gesonderten Hinweise.

7 Ausführungssicht

Die Ausführungssicht beschreibt das Laufzeitverhalten. Hier werden die Laufzeitelemente aufgeführt und beschrieben, welche Module sie zur Ausführung bringen. Ein Modul kann von mehreren Laufzeitelementen zur Laufzeit verwendet werden. Die Ausführungssicht beschreibt darüber hinaus, welche Laufzeitelemente spezifisch miteinander kommunizieren. Zudem wird bei verteilten Systemen (z. B. Client-Server-Systeme) dargestellt, welche Module von welchen Prozessen auf welchen Rechnern ausgeführt werden.

8 Zusammenhänge zwischen Anwendungsfällen und Architektur

In diesem Abschnitt sollen Sequenzdiagramme mit Beschreibung(!) für zwei bis drei von Euch ausgewählte Anwendungsfälle erstellt werden. Ein Sequenzdiagramm beschreibt den Nachrichtenverkehr zwischen allen Modulen, die an der Realisierung des Anwendungsfalles beteiligt sind. Wählt die Anwendungsfälle so, dass nach Möglichkeit alle Module Eures entworfenen Systems in mindestens einem Sequenzdiagramm vorkommen. Falls Euch das nicht gelingt, versucht möglichst viele und die wichtigsten Module abzudecken.

9 Evolution

Beschreibt in diesem Abschnitt, welche Änderungen Ihr vornehmen müsst, wenn sich Anforderungen oder Rahmenbedingungen ändern. Insbesondere würden hierbei die in der Anforderungsspezifikation unter „Ausblick“ genannten Punkte behandelt werden.

...