

# Eine Untersuchung der Zusammenarbeit zwischen Requirements Engineer und Software Architekt als Seminararbeit Wintersemester 2016/2017

Franz-Dominik Dahmann  
Master Informatik

Hochschule Bonn-Rhein-Sieg  
<https://www.h-brs.de/de>

Grantham-Allee 20, 53757 Sankt Augustin  
Email: franz.dahmann@smail.inf.h-brs.de

Jan Eric Müller  
Master Informatik

Hochschule Bonn-Rhein-Sieg  
<https://www.h-brs.de/de>

Grantham-Allee 20, 53757 Sankt Augustin  
Email: jan-eric.mueller@smail.inf.h-brs.de

## Abstract—...

Um eine Optimierung zu ermöglichen bieten sich verschiedene Vorgehensweisen an. Im folgenden wird eine Kategorisierung von Anforderungen vorgenommen, die Relevant für die Architektur einer Software sein können. Diese Kategorisierung bietet Requirements Engineers die Möglichkeit speziellere Fragen zu der gewünschten Architektur der Software zu stellen. Neben einer Kategorisierung wird ferner ein iteratives Vorgehen zum Design der Software Architektur vorgestellt. Mithilfe des Attribute driven Designs (ADD) soll es möglich sein eine feste Struktur zu haben, anhand derer eine korrekte Architektur entworfen werden kann. Desweiteren wird ein Ansatz vorgestellt und bewertet, welcher die Prozesse der Architektur-Entscheidung und des Requirements-Engineering erfahrungsgetrieben in einen gemeinsamen Prozess integriert. Dieser ermöglicht eine engere Kommunikation zwischen Software-Architekt und Requirements Engineer. Anschließend wird dieser Ansatz mit dem Twin Peaks Modell verglichen und beurteilt welches der beiden Verfahren den größeren Mehrwert bringt oder ob diese sinnvoll kombiniert werden können.

Mit der Untersuchung der Verfahren soll erreicht werden, dass sowohl dem Requirements Engineer als auch dem Software Architekten, Potenziale aufgezeigt werden auf deren Basis die Zusammenarbeit verbessert werden kann. Durch die Beschreibung mehrerer Verfahren soll zudem die Möglichkeit gegeben sein, abzuwägen welche in der individuellen Situation am besten geeignet sind.

...

Das zentrale Problem der Zusammenarbeit scheint eine mangelnde Verständigung zwischen den Requirements Engineer und dem Software Architekten zu sein. Software Architekten sehen sich gezwungen, entweder eine Architektur anzunehmen, oder weiterführende Gespräche mit dem Kunden zu führen. Requirements Engineers sehen sich mit dem Problem konfrontiert, dass sie nicht die Fachkenntnis haben, die architekturrelevanten Informationen mit dem Kunden zu erarbeiten. Um jedoch eine Optimierung zu ermöglichen ist es zunächst notwendig zu identifizieren, warum die Anforderungsdokumente, die der Requirements Engineer generiert, nicht ausreichen um einen vollständigen Architekturentwurf herzustellen und wie Erkenntnisse des Software-Architekten wieder in die Anforderungs-Spezifikationen einfließen können.

Außerdem wird recherchiert ob Zusammenarbeit und Kommunikation durch eine passende Tool-Unterstützung verbessert werden kann.

## I. EINFÜHRUNG

Bei der Realisierung eines Softwareentwicklungsprojektes besteht häufig eine Kluft zwischen den Software-Architekten auf der einen Seite und den Requirements Engineers auf der anderen Seite. Software Architekten sind zum Beispiel häufig mit dem Problem konfrontiert, dass Anforderungsdokumente nicht ausreichend sind um weitreichende Architekturentscheidungen in Bezug auf den Softwareentwurf zu treffen. Daher entsteht in diesem Fall für den Software-Architekten häufig der Mehraufwand, dass dieser bei weiteren Interviews ein präziseres Bild von der gewünschten Software Architektur erhält [?]. Dies resultiert häufig in Verzögerungen die sich dann darin widerspiegeln, dass Termine nicht eingehalten werden können. Betreibt der Software Architekt diesen Mehraufwand nicht und trifft eigene Annahmen bezüglich der Architektur der Software [?] kann dies wiederum zu einer geringeren Akzeptanz des Kunden und im schlimmsten Fall zum Auftragsverlust führen. Die Requirements Engineers wissen auf der anderen Seite wiederum nicht, welche Anforderungen konkret wichtig für den Software Architekten sind, da ihnen die entsprechende Fachkenntnis fehlt [?]. Somit können diese nicht zielführend die notwendigen Informationen mit dem Kunden erarbeiten. Ohne diese Informationen ist eine ausreichende Grundlage der Anforderungen für den Architekturentwurf nicht gegeben. Um diese Kluft zu überbrücken ist es notwendig Verfahren zu ermitteln, die es Requirements Engineers ermöglicht die richtigen Informationen einzuholen und die Zusammenarbeit mit den Software Architekten zu optimieren.

Im Folgenden werden zunächst vier Verfahren untersucht die das Potenzial haben die Zusammenarbeit zwischen Requirements-Engineer und Software-Architekt zu verbessern.

Diese vier Verfahren sind:

- ADD 3.0 wird beschrieben in III-A
- COSMOD-RE wird beschrieben in III-D
- Probing —TODO—
- Twin Peaks —TODO—

Diese vier Verfahren werden nach einem festgelegten Schema untersucht. Es werden folgende Punkte betrachtet:

- Ziele
- Funktionsweise

Mit den Zielen wird zunächst beschrieben, was die Methode bezwecken will. In der Funktionsweise werden folgende Aspekte untersucht:

- Randbedingungen  
In den Randbedingungen wird geklärt, welche Einschränkungen die Methode hat und welche Voraussetzungen geklärt sein müssen, um die Methodik anzuwenden.
- Eingabe  
In der Eingabe wird beschrieben, welche Eingaben die Methodik erhält.
- Vorgehensmodell  
Mit dem Vorgehensmodell wird beschrieben, wie die Methode anzuwenden ist.
- Ausgabe  
In der Ausgabe wird beschrieben, was am Ende das Resultat der Methode ist und welchen Mehrwert dieses haben könnte.

Nach der Untersuchung der einzelnen Methoden werden diese vergleichend in ?? ausgewertet. Hierfür werden die Methoden zunächst unter den eben genannten Punkten verglichen. Danach werden sie in den Kontext der in II genannten Probleme gebracht und hinsichtlich ihrer Fähigkeit, diese zu lösen betrachtet. Es wird überprüft, wo diese Verfahren ihre Stärken und Schwächen offenbaren.

Abschließend wird überprüft, wie diese Verfahren gewinnbringend kombiniert werden oder sich gegenseitig ergänzen können.

## II. PROBLEMSTELLUNG

Während der Zusammenarbeit zwischen Requirements Engineer und Software Architekt können vielfältige Probleme auftreten. Bei der Untersuchung möglicher Probleme lässt sich eine Kategorisierung dieser vornehmen. So sind einige Probleme bedingt durch den Menschen selbst, während andere Probleme sich aus den Anforderungen ergeben. Im folgenden werden die kategorisierten Probleme genauer ausgeführt.

### A. Durch den Menschen bedingt

Bei der Betrachtung der direkt durch den Menschen bedingten Probleme fallen folgende auf:

P1: *Schlechte Kommunikation*

P2: *Konkurrierende Interessen*

P3: *Fehlendes Know-How*

1) *Schlechte Kommunikation:* Eine schlechte Kommunikation ist in diesem Zusammenhang als eine Kommunikation zu sehen, bei der wesentliche Intentionen und Informationen nicht ermittelt werden. Ein wichtiger Aspekt schlechter Kommunikation ist die Distanz zwischen den Akteuren innerhalb eines Projektes.

Nach [?] kann vor allem Distanz ein Grund für schlechte Kommunikation sein. Distanzen in der Kommunikation zwischen Requirements Engineer und Software Architekt können die Koordination reduzieren, was wiederum Projektverzögerungen hervorrufen kann, sowie das nicht Erfüllen von Kundenwünschen. Distanzen werden nach [?] vor allem als räumliche Distanzen bei global agierenden Projektteams gesehen. Bei räumlichen Distanzen können nach [2] beispielsweise Begleitprobleme wie das Beherrschen einer Sprache aufkommen. So kann das schlechte Beherrschen einer Sprache zu Missverständnissen führen und Fehler in den Anforderungen hervorrufen. Zusätzlich können nach [?] soziale, kulturelle oder zeitliche Distanzen hinzukommen.

Bei nicht ausreichendem Kommunikationsfluss zwischen Requirements Engineer und Software Architekt kann das Problem aufkommen, dass der Requirements Engineer, während der Anforderungsgewinnung, keine Rücksprache mit dem Software Architekten hält [?]. Ohne ausreichende Rücksprache kann beispielsweise eine Beeinträchtigung der Qualität der architekturrelevanten Anforderungen auftreten [?]. Dies kann von fehlenden bis fehlerhaften Anforderungen reichen. Ein weiteres mögliches Problem ist, dass der Software Architekt dem Requirements Engineer nicht ausreichend vermittelt, welche Informationen er für einen gültigen Architekturentwurf benötigt [?]. Aus diesen Kommunikationsproblemen können Missverständnisse in den Verantwortlichkeiten folgen [?]. Auch hier ist eine mögliche Folge die negative Beeinflussung der Qualität der Architektur Anforderungen.

2) *Konkurrierende Interessen:* Unter konkurrierenden Interessen ist zu verstehen, dass die Stakeholder des Entwicklungsprozesses der Software-Architektur in der Projektarbeit verschiedene Interessen verfolgen, die miteinander in Konflikt stehen. Die Stakeholder haben zwar eine gemeinsame Vision von dem fertigen Produkt, verfolgen aufgrund variierender Interessen jedoch eine unterschiedliche Art der Zielerreichung [?]. Weiter kann es sein, dass Stakeholder durch ihre unterschiedlichen Sichten auf das Projekt auf verschiedene Arten mit diesem

interagieren. Dies kann wiederum mit den Projektzielen konkurrieren. Der Requirements Engineer vertritt hierbei eher die Sicht des Kunden, während der Software Architekt, den Standpunkt des Entwicklers vertritt [?]. Ein Problem kann ebenfalls auftreten, wenn Stakeholder verschiedene Tools, Prozesse und Arbeitsweisen nutzen um das Projektziel zu erreichen. Wenn hier die Kompromissbereitschaft fehlt kann dies die Arbeit erschweren [?].

3) *Fehlendes Know-How*: Fehlendes Know-How kann zu diversen Problemen führen. Dabei kann hier zwischen zwei Ebenen unterschieden werden. Einerseits können Probleme aus fehlendem Know-How bei der Methodik der jeweils anderen Rolle entstehen [?][?]. Hierbei können unter anderem Probleme bei der Planung und Umsetzung eines Projektes resultieren, da hierdurch potenziell relevante Vorgehensweisen oder Methoden nicht berücksichtigt werden können. Andererseits können Probleme aus inhaltlich fehlendem Know-How entstehen [?]. So kann es dem Requirements-Engineer zum Beispiel schwer fallen die Schwerpunkte so zu setzen, dass der Software-Architekt die Informationen erhält, die er benötigt [?]. Dies wird dann zum Problem wenn der Requirements-Engineer nicht weiß welche Informationen der Software-Architekt für die Umsetzung benötigt. Ferner können Missverständnisse auftreten, wenn der Software-Architekt bei fachspezifischen Begriffen ein anderes Verständnis hat als der Requirements-Engineer [?].

## B. Qualität der Anforderungen

Mit der Untersuchung der Probleme, die sich auf die Qualität der Anforderungen beziehen, fallen folgende auf:

P4: *Zu restriktive / detaillierte architekturelevante Anforderungen*

P5: *Ungenau/ fehlende architekturelevante Anforderungen*

P6: *Nicht klar hervorgehobene architekturelevante Anforderungen*

P7: *Wechselwirkungen zwischen architekturelevanten Anforderungen*

1) *Zu restriktive / detaillierte architekturelevante Anforderungen*: Vor allem bei sehr großen Projekten kann es passieren, dass es tausende von Anforderungen, die für die Architektur relevant sind, gibt [?]. Es besteht eine hohe Wahrscheinlichkeit, dass in einem solchen Fall viele Anforderungen Widersprüche erzeugen und den Lösungsraum zu sehr einschränken. Hier können restriktive oder detaillierte Anforderungen die Arbeit von Requirements-Engineer und Software-Architekt unnötig einschränken und dadurch erschweren.

2) *Ungenau / Fehlende architekturelevante Anforderungen*: In der Anforderungsgewinnung kann es passieren, dass wichtige architekturelevante Anforderungen nicht erhoben werden, oder wichtige Details fehlen [?][?].

Dies kann jedoch weitreichende Auswirkungen auf die Software-Architektur haben, da dem Software-Architekt notwendige Informationen bei der Konzeption fehlen. Dieser sieht sich dann gezwungen entweder eine Software-Architektur frei zu entwerfen oder zusätzliche Klärungsgespräche mit dem Kunden zu führen. Auch hier kann es passieren, dass bei besonders großen Projekten mit mehreren tausenden Anforderungen, wichtige nicht erhoben werden [?]. Vor allem im Kontext der agilen Softwareentwicklung bedingt ein iteratives Vorgehen, dass manche architekturelevanten Anforderungen erst während der Modellierung oder sogar Implementierung der Architektur gewonnen werden [?].

3) *Nicht klar hervorgehobene architekturelevante Anforderungen*: Eines der Hauptprobleme architekturentwerfender Anforderung liegt darin, dass es üblicherweise schwer ist diese zu identifizieren und zu spezifizieren [?]. Werden architekturelevante Anforderungen für den Software-Architekten nicht hervorgehoben, kann es passieren, dass dieser wichtige Anforderungen erst spät, wenn überhaupt wahrnimmt [?]. Dies kann zur Folge haben, dass wichtige architekturelevante Entscheidungen nicht rechtzeitig getroffen werden können und zu einem Mehraufwand zu einem späteren Zeitpunkt führen [?]. Zusätzlich liegt ein Problem in der fehlenden Ausführung der Auswirkungen von architekturelevanten Anforderungen auf die spätere Software-Architektur [?]. So kann hieraus folgen, dass wichtige Projektziele nicht erfüllt werden, da der Zeitrahmen des Projektes überschritten wird.

4) *Wechselwirkung zwischen Anforderungen*: Unter Wechselwirkung zwischen Anforderung ist zu verstehen, dass Anforderungen teilweise voneinander abhängig sein können. Diese Abhängigkeit kann auf verschiedene Arten und Weisen erfolgen. Ist beispielsweise ein Ansatz gegeben, bei dem mehrere Abstraktionsstufen vorhanden sind kann es passieren dass wichtige Abhängigkeiten, die über mehrere Abstraktionsstufen hinweg reichen nicht berücksichtigt werden oder weitere Probleme verursachen [?]. Änderungen von Anforderungen auf verschiedenen Hierarchieebenen treffen kontinuierlich ein und können somit Einfluss auf die Architektur haben und erheblichen Änderungsbedarf verursachen [?]. Auch ist es schwierig die Konsistenz und Nachverfolgbarkeit der Projektion von Anforderungen auf die Software-Architektur zu gewährleisten, da eine Anforderung mehrere architekturelevante Artefakte betreffen kann. Zusätzlich kann ein architekturelevantes Artefakt mehrere Anforderungen betreffen [?].

Neben den aufgeführten Problemen gibt es weitere, die hier nicht näher behandelt werden. Darunter fällt zum Beispiel ein phasenbezogenes Requirements Engineering.

### III. UNTERSUCHUNG GEGEBENER METHODEN (JEM)

Da die beiden Bereiche des Requirements-Engineering und der Software-Architektur ein enormes Ma an Wissen und Fähigkeiten erfordern werden diese in der Regel von verschiedenen Teams betreut [8]. Wie bereits genannt, entstehen hier häufig Probleme durch fehlendes technisches Know-How bei der Software-Architektur spezifische Aspekte bei den Requirements-Engineers. Auch können diese oft nicht zwischen funktionalen Anforderungen (FR) und architektur-spezifischen funktionalen Anforderungen (ASFR) unterscheiden [9]. ASFRs sind FRs welche kritisch, sehr risikobehaftet, volatil und bei Änderungen ein aufwendiges oder teures Refactoring mit sich bringen würden oder einen anderweitig großen Impact auf die zu konzipierende Software-Architektur hätten [8]. Durch das fehlende Know-How entstehen unvollständige Anforderungs-Artefakte, in welchen wesentliche ASFRs für den Software-Architekten fehlen.

Im Folgenden sollen vier verschiedene Methoden untersucht werden, welche den Anspruch haben die Kluft in der Zusammenarbeit zwischen Software-Architekt und Requirements Engineer zu überbrücken. Dabei werden zunächst die Ziele der Methode herausgearbeitet und anschließend die Funktionsweise näher erläutert, bevor die vier Methoden im nachfolgenden Kapitel ausgewertet werden. Bei der Funktionsweise wird auf folgende Aspekte genauer eingegangen:

- *Randbedingungen*: Gibt es Einschränkungen oder Vorbedingungen für die Ausführung der Methode?
- *Eingabe*: Welche Artefakte werden für die Ausführung der Methode benötigt?
- *Vorgehensmodell*: Wie wird die Methode durchgeführt?
- *Ausgabe*: Was liefert die Methode für ein Ergebnis?

#### A. ADD 3.0 (FDD)

1) *Beschreibung*: Attribut-driven-Design (ADD) bezeichnet ein Vorgehensmodell, bei dem iterativ ein Architekturdesign ausgearbeitet wird. ADD wird in Form von sogenannten Design Rounds durchgeführt. Eine Design-Round kann hierbei beispielsweise einem Sprint in SCRUM zugeordnet werden. Dies bedeutet in einem Projekt kann es mehrere Design-Rounds geben, mit denen die Software-Architektur verfeinert wird.

Eine Eigenschaft, die bei ADD besonders hervorsteicht ist, dass es innerhalb der Design-Rounds eine klare Folge von Anweisungen gibt, die auszuführen sind um die Software-Architektur zu entwickeln. Hierbei ist relevant zu erwähnen, dass in ADD die Dokumentation und Analyse als wichtigste Elemente zur Entwicklung der Software-Architektur betrachtet werden. Nachteil bei ADD ist jedoch, dass die Voraussetzung hierfür ist, dass bereits primäre funktionale Anforderungen und Szenarien erhoben sind. Dies bedeutet ADD findet nicht

direkt in der Anforderungsgewinnung Anwendung, sondern erst danach. Insgesamt umfasst ADD sieben Schritte die innerhalb einer Design-Round auszuführen sind.

Diese sind:

- 1: Review Inputs
- 2: Establish Iteration goal by selecting drivers
- 3: Choose one or more elements of the system to refine
- 4: Choose one or more design concepts that satisfy the selected drivers
- 5: Instantiate architectural elements, allocate responsibilities and define interfaces
- 6: Sketch views and record design decisions
- 7: Perform analysis of current design and review iteration goal and achievement of design purpose

Um bei ADD eine Design-Round durchführen zu können sind jedoch zunächst einige Eingaben für den Prozess vorzubereiten.

Diese sind:

- *bergeordnete Zielstellung*
- *Primäre funktionale Anforderungen*
- *Szenarien*
- *Einschränkungen*

a) *Step 1 - Überprüfung der Eingaben*: Zunächst muss sichergestellt werden, dass die *bergeordnete Zielstellung* für die darauffolgenden Design-Aktivitäten festgelegt ist. Diese kann beispielsweise die erstmalige Erstellung eines Design-Entwurfes oder die Verbesserung eines vorhandenen Architektur-Designs sein. Danach wird überprüft, ob die für die Design-Round relevanten Anforderungen und Szenarien korrekt sind. Hier ist unter anderem zu prüfen ob alle relevanten Stakeholder berücksichtigt werden und ob die erhobenen Anforderungen richtig priorisiert sind. Zuletzt muss noch geprüft werden, ob es Einschränkungen bezüglich der Software-Architektur gibt, die in der Design-Round zu berücksichtigen sind.

b) *Step 2 - Festlegung des Ziels der Iteration durch Auswahl von Artefakten*: Eine Design-Runde

#### B. Probing (JEM)

1) *Ziele der Methode*: Das Ziel des Probing ist es, Requirements Engineers mit Fragen auszustatten die eine Erhebung architekturrelevanter Anforderungen, den ASFRs, ermöglichen. Diese Fragen werden Probing Questions (PQ) genannt.

Die Haupttreiber architekturrelevanter Entscheidungen sind nicht-funktionale (NFR) bzw. qualitative Anforderungen [9]. Diese haben eine explizite Auswirkung auf die zu erstellende Architektur. Bei funktionalen Anforderungen bzw. ASFRs sind diese Auswirkungen meist implizit und müssen zunächst durch weitere Interviews mit dem Kunden herausgearbeitet werden. Auch sind hier die für einen Software-Architekten erforderlichen Informationen nicht immer klar

in der Anforderung aufgeföhrt [9]. Dies kann zu falschen architektonischen Entscheidungen föhren. Um dieses Problem anzugehen beschäftigen sich die Probing Questions mit den ASFRs. Requirements Engineers sollen mit Probing Questions ausgestattet werden um zusätzliche relevante Fragen stellen zu können. Dadurch soll während der Anforderungserhebung eine vollständigere Anforderungsspezifikation erstellt werden, in welcher die ASFRs aussagekräftiger sind [9]. Auch helfen diese den Requirements Engineers ein genaueres Verständnis für den Software-Architekten aufzubauen, welche Informationen er für die Konzeption und Implementierung der Software-Architektur benötigt.

2) *Funktionsweise der Methode:* Funktionsweise der Methode

- a) *Randbedingungen:* Randbedingungen
- b) *Eingabe:* Eingabe
- c) *Vorgehensmodell:* Vorgehensmodell

TODO: PQ Kategorien und Typen aus (8).

TODO: PQ-Flow aus (6).

#### **Paper so far:**

- (6) Probing for Requirements Knowledge to Simulate Architectural Thinking
- (8) What you see is what you get: Understanding Architecturally Significant Functional Requirements
- (9) Identifying Architecturally Significant Functional Requirements

- d) *Ausgabe:* Ausgabe

#### **C. Twin Peaks (JEM)**

1) *Beschreibung:* TODO

#### **Paper so far:**

- (5) Weaving Together Requirements and Architecture
- (6) The Twin Peaks of Requirements and Architecture
- (7) Towards Bridging the Twin Peaks of Requirements and Architecture

2) *Bewertung:*

#### **D. sCenario and gOal based SysteM develOpment methoD (COSMOD) (FDD)**

COSMOD-Requirements Engineering (COSMOD-RE) beschreibt ein iteratives Vorgehen zum gleichzeitigen Design von Anforderungen und Software-Architektur. Kerngedanke bei COSMOD-RE ist eine Aufteilung in vier Hierarchiestufen, wo sowohl Architektur als auch Anforderungen definiert werden. In diesen vier Hierarchiestufen wird einerseits aus der Anforderungssicht und andererseits aus der Architektursicht betrachtet, welche Anforderungen und Komponenten dem System zuzuordnen sind.

1) *Ziele der Methode:* Kernziel von COSMOD-RE ist es die Entwicklung von Anforderungs- und Architekturartefakten für softwareintensive eingebettete Systeme zu unterstützen. Ein Ziel- und Szenario-basierter Ansatz wie COSMOD-RE, der das Co-Design von Architekturartefakten und Anforderungen ermöglicht muss jedoch einige Anforderungen erfüllen um einen Nutzen zu haben. Die Anforderungen an eine solchen Methodik sind [?]

- **Parallele Entwicklung von Anforderungen und Architekturartefakten**  
Da Anforderungen und die Software Architektur die Kerntreiber hinter innovativen Projekten sind ist es wichtig beide gleichermaßen zu nutzen, sodass keines von beiden in den Vordergrund tritt [?].
- **Unterstützung der Anordnung von Anforderungen und Architekturartefakten**  
Werden Anforderungen und Architekturartefakte parallel entwickelt kann es passieren, dass Inkonsistenzen auftreten. Um diese zu vermeiden ist es notwendig, dass COSMOD-RE diese erkennt und behebt [?].
- **Definition detaillierter Anforderungen auf der Basis von Architekturartefakten**  
Ohne technisches Wissen fällt es Stakeholdern schwer notwendige Details für die Erhebung architekturelevanter Anforderungen zu liefern. Daher sollte es COSMOD-RE ermöglichen die detaillierten Anforderungen erst nach der initialen Definition von Architekturartefakten zu liefern [?].
- **Nutzung einer Abstraktionshierarchie**  
Da bei einem Co-Design Vorgehen eine große Komplexität bestehen kann ist es notwendig eine Abstraktionshierarchie einzuführen um diese entsprechend zu behandeln [?].

2) *Funktionsweise der Methode:* Das wichtigste Element von COSMOD-RE ist die Abstraktionshierarchie. Bei dieser werden alle Aktivitäten die im Rahmen der Methodik stattfinden eingeordnet und miteinander in einen Kontext gesetzt. So wird auf jeder Ebene der Abstraktionshierarchie eine Architektursicht und eine Anforderungssicht erzeugt. Hierbei ist jedoch zu beachten, dass bei COSMOD-RE nicht unbedingt ein Top-Down-Ansatz zu wählen ist, da Anforderungsartefakte und Architekturartefakte auf allen Ebenen gleichzeitig bearbeitet werden können.

a) *Randbedingungen:* Bei der Anwendung von COSMOD-RE ist zu beachten, dass einige Bedingungen erfüllt sein müssen um die Methodik richtig anzuwenden:

- **Definition einer Abstraktionshierarchie**  
Es gibt keine Richtlinien bezüglich der Definition der Abstraktionshierarchie. Dies bedeutet, dass die verwendeten Hierarchiestufen in jedem Projekt neu festgelegt werden müssen. Ferner ist hierbei von Relevanz, zu definieren,

wo die Trennlinien zwischen verschiedenen Abstraktionsstufen sind [?].

- Verknüpfung von Anforderungs- und Architekturmodellen  
Da im Rahmen von COSMOD-RE Anforderungsartefakte und Architekturartefakte parallel entwickelt werden ist es notwendig diese miteinander zu verknüpfen um sie in den Kontext des Ziels der Software zu bringen. Für diese Verknüpfung ist die Anwendung von Methodenfragmenten notwendig [?].
- Definition von Konsistenzbedingungen  
Für die Definition von Konsistenzbedingungen gibt es keinen allgemeingültigen Ansatz. Daher ist es notwendig in jedem Projekt neu zu definieren, ob Konsistenz gegeben ist oder nicht [?].

Sind diese Randbedingungen geklärt ist es möglich COSMOD-RE anzuwenden.

*b) Eingabe:* Als Eingabe benötigt COSMOD-RE eine System-Vision. Dies bedeutet in initialen Gesprächen mit Stakeholdern muss bereits festgehalten sein, welche Vision von dem zu konzipierenden System gegeben ist.

*c) Vorgehensmodell:* —TODO— Vorgehensmodell beschreiben

- 1 System: Die Systemebene beschreibt die oberste Ebene, bei der in der Anforderungssicht das System als ganzes betrachtet wird. Im Fokus stehen dabei die Interaktionen mit dem System. Weiter werden funktionale Anforderungen und Qualitätsanforderungen erstellt, die sich auf das Gesamtsystem beziehen. Die Architektursicht konzentriert sich hier auf die Definition von externen System-schnittstellen. Hier definierte Artefakte sollen primär die Kommunikation mit Stakeholdern unterstützen.
- 2 Komponenten: Die Komponentenebene bezeichnet die Aufteilung des Systems in einzelne Komponenten aus denen sich dieses zusammensetzen soll. Für jede Komponente werden funktionale Anforderungen und Qualitätsanforderungen formuliert. Da in dieser Ebene die Basis für die Systemarchitektur gelegt wird, hat die Kommunikation zwischen dem Software-Architekten und dem Requirements Engineer von besonderer Bedeutung.
- 3 Hard-/Software Komponenten: Auf dieser Ebene werden die zuvor erstellten Komponenten in Hard- und Software Komponenten aufgeteilt und weiter verfeinert. Anforderungen auf dieser Ebene sind somit speziell auf die Komponentenart bezogen.
- 4 Deployment: Auf dieser Ebene werden Softwarekomponenten programmierbaren Hardwarekomponenten zugeordnet. Anforderungen auf dieser Ebene beziehen sich auf das Deployment der Softwarekomponenten und ihrem Einfluss auf vorher definierte Anforderungen.

Da die Zusammenarbeit zwischen Software-Architekt und Requirements Engineer vor allem in den obersten beiden Ebenen von Relevanz ist, sind die unteren beiden Ebenen in

diesem Kontext zu vernachlässigen.

Im Rahmen der Erstellung der Software-Architektur und der Anforderungen gibt es drei Co-Design Prozesse die sich wiederum in fünf Sub-Prozesse unterteilen lassen. Bei Ausführung der Prozesse werden als Artefakte sowohl die System-Architektur als auch Ziele, Szenarien und Anforderungen erzeugt.

*d) Ausgabe:*

3) *Beschreibung:* In der Anforderungserhebung soll es möglich sein Anforderungen in einer Form zu erheben, die es Software Architekten einfacher macht, den Architekturentwurf zu konzipieren. Um dies zu realisieren bietet sich eine Kombination aus Ziel-basierten Ansätzen und Szenario-basierten Ansätzen an. Die Kombination ist deswegen von Relevanz, weil ein Ansatz allein nicht ausreichen kann um die Anforderungen in angemessener Weise zu erheben.

4) *Ziel-basierte Ansätze:* Ziel-basierte Ansätze zielen vorrangig darauf ab, ein umfassendes Verständnis der Wünsche und Ziele der Stakeholder sowie auf die zu erzielenden Auswirkungen auf die Systemumgebung ab (Silkora Referenz S.18). Dies bedeutet, dass es bei Ziel-basierten Ansätzen vor allem darauf ankommt, zu verstehen, welche Vision der Stakeholder von dem zukünftigen System hat. Bei der Erfassung dieser Vision ist ein natürlichsprachlicher Ansatz fehlerbehaftet, da hier sehr aufwändige manuelle Konsistenzprüfungen notwendig wären. Deswegen bieten sich hier vor allem Modell-basierte Ansätze an.

Ein gutes Beispiel für ein Modell-basierten Ansatz ist der KAOS-Ansatz. Dieser Ansatz bietet den Vorteil, dass er mit wenigen präzise formulierten Modellierungsobjekten auskommt. Dies ist deswegen ein Vorteil, weil so kein besonders tief reichendes Fachwissen notwendig ist um das Modell zu interpretieren. Ferner ist der Ansatz für die Konzeption softwareintensiver eingebetteter Systeme geeignet, was eine verzahnte Entwicklung von Anforderungen und Architektur über mehrere Abstraktionsstufen hinweg ermöglicht (Silkora Referenz S.31).

*a) KAOS:* Lamsweerde (Lamsweerde Referenz) beschreibt einen modellbasierten Ansatz zur Darstellung von Zielen und den Referenzen innerhalb von Zielen. Hierfür muss zunächst eine genauere Betrachtung der Zieldefinition vorgenommen werden. So sind in dem Kontext der KAOS-Methode Ziele in Behavioral-Goals und Soft-Goals zu unterteilen.

Behavioral-Goals beschreiben eine deklarative Sicht auf Ziele, die beschreibt, wie ein System sich zu verhalten hat. Dies bedeutet, dass in diesem Fall besonders das Verhalten von Systemen im Fokus steht. Gültig ist eine endliche Menge

von Verhaltensweisen des Systems.

Grundsätzlich lassen sich Behavioral-Goals in zwei Kategorien aufteilen, die Achieve-Goals und die Maintain/Avoid-Goals. Die Achieve-Goals beschreiben Systemverhalten, bei dem es darauf ankommt, dass ein System zu einem definierten Zeitpunkt einen definierten Zustand erreicht. Maintain/Avoid-Goals beschreiben Systemverhalten, bei dem es darauf ankommt, dass ein System über einen definierten Zeitraum hinweg einen definierten Zustand aufrechterhält, oder einen definierten Zustand vermeidet.

Soft-goals beschreiben Präferenzen innerhalb von gültigen Systemverhaltensweisen. Diese lassen sich zunächst in funktionale Ziele und nicht funktionale Ziele aufteilen. Die funktionalen Ziele können die folgenden Kategorien haben:

- Satisfaction: Funktionale Ziele, die sich damit beschäftigen User-Anfragen zu beantworten.
- Information: Funktionale Ziele, die damit beschäftigt sind User über wichtige Systemzustände zu informieren.
- Stim-response: Funktionale Ziele, die sich damit beschäftigen auf Events eine angemessene Reaktion zu erzeugen.

Mit dem gegebenen Ziel die Zusammenarbeit zwischen Requirements Engineer und Software Architekt zu optimieren sind vor allem die funktionalen Ziele der Soft-Goals und die Behavioral-Goals von Relevanz.

der KAOS Ansatz, der die zuvor beschriebenen Zielarten als Grundlage nutzt, verwendet zur Modellierung Und-/Oder-Graphen, die in diesem Kontext Zieliagramm genannt werden. Jedes im Graphen modellierte Ziel wird zunächst durch eine Reihe von Eigenschaften in einer Zielschablone charakterisiert. Die Eigenschaften können unter anderem Name, Definition, Quelle, Zielkategorie und Priorität sein.

–TODO– Erzeugung Abbildung Und-Oder-Graph + Beschreibung der Elemente

Das Problem von Ziel-basierten Ansätzen wie dem KAOS Ansatz ist, dass eine Software Architektur allein basierend auf den Zielen wichtige Aspekte vernachlässigt. So zum Beispiel welche Rolle diese Ziele erreichen soll und gegebenenfalls wie die dieses Ziel umzusetzen ist. Daher ist es notwendig Szenario-basierte Ansätze zu betrachten.

5) *Szenario-basierte Ansätze*: Szenario-basierte Ansätze zielen vorrangig darauf ab, die wesentlichen geforderten Interaktionen des Systems mit dessen Umgebung zu definieren und mit den Stakeholdern abzustimmen (Silkora Referenz S.18).

Um die optimale Zusammenarbeit zu stützen ist es notwendig, die Szenarien in einer Kombination aus Anwendungsfalldiagrammen und Message Sequence Charts graphisch zu modellieren. Dadurch, kann der

Zusammenhang zwischen Szenarien aufgezeigt werden und wichtige Aspekte, die von Bedeutung für den Design der Softwarearchitektur sind, berücksichtigt werden. Es ist mit diesem Vorgehen möglich Szenarien zu komplexeren Szenarien zusammenzusetzen und des weiteren Iterationen und alternative Szenarienvläufe abzubilden (Silkora Referenz S.33). Bei der Betrachtung von Szenarien ist es jedoch auch hier notwendig, diese zunächst in Schablonen zu dokumentieren, um eine präzise Beschreibung zu haben. In einer Schablone soll unter anderem festgehalten werden, welche primären und sekundären Akteure gegeben sind. Ferner sollen eine Kurzbeschreibung und mit dem Szenario verknüpfte Ziele gegeben sein, um klar hervorzuheben, in welchem Kontext das Szenario zu sehen ist.

a) *Anwendungsfalldiagramm*: Das Anwendungsfalldiagramm zeigt eine Übersicht über die Anwendungsfälle eines Systems oder einer Systemkomponente. Es stellt zudem die Akteure des Systems (der Komponente) und deren Beteiligung an den Anwendungsfällen grafisch dar (Silkora Zitat S.34).

Anwendungsfälle können als Oberbegriff für Szenarien betrachtet werden, da ein Szenario sich aus einem Anwendungsfall generieren lässt. Dies bedeutet ein Anwendungsfall kann Grundlage für eine Vielzahl von Szenarien sein.

–TODO– Anwendungsfalldiagramm beispielabbildung mit Notationselementen

Zu den Notationselementen eines Anwendungsfalldiagramms zählen:

- Akteur: Als Akteur lässt sich eine Person oder Entität bezeichnen, die mit dem zu konzipierenden System in Beziehung steht. Diese können zum Beispiel Nutzer sein, die über eine Eingabemaske Daten eintragen sollen.
- Systemgrenze: Systemgrenzen umschließen das geplante System. Akteure, die mit dem System interagieren befinden sich außerhalb der Systemgrenze, während die dem System zugeordneten Anwendungsfälle sich innerhalb der Systemgrenze befinden.
- Anwendungsfall: Ein Anwendungsfall beschreibt eine Funktionalität des geplanten Systems. In Kombination mit dem Akteur, der in Relation zu einem Anwendungsfall steht, wird dargestellt, was das System machen soll.
- Erweiterung eines Anwendungsfalls: Ein Anwendungsfall kann mittels Include- oder Extend-Beziehung erweitert werden. Dies soll es ermöglichen komplexere Anwendungsfälle abzubilden und gleichzeitig die Anzahl der Redundanzen möglichst gering halten. Eine Include-Beziehung bedeutet hierbei, dass ein Anwendungsfall einen weiteren Anwendungsfall beinhaltet. Die Extend-Beziehung bedeutet, dass es zu einem Anwendungsfall eine Erweiterung unter einer Bedingung gibt. Die Bedingung gibt an welche Konditionen erfüllt sein müssen dass die Erweiterung greift.

Anwendungsfalldiagramme stellen Akteure und Anwendungsfälle dar. Dadurch wird veranschaulicht, welche Akteure mit welchen Anwendungsfällen in Beziehung stehen und ob es wichtige Beziehungen zwischen verschiedenen Anwendungsfällen gibt. Grundsätzlich ist es möglich, Anwendungsfalldiagramme auf verschiedenen Abstraktionsstufen abzubilden, um so sowohl für das Gesamtsystem, als auch für die Teilsysteme die Beziehungen zu betrachten. Um jedoch Szenarien präzise zu spezifizieren, reichen Anwendungsfalldiagramme nicht aus.

b) *Message-Sequence-Charts*: Mithilfe von Message-Sequence-Charts ist es möglich, eine präzise Spezifikation der verschiedenen Szenarien eines Anwendungsfalles zu generieren (Silkora Zitat S.37). Bei der Betrachtung der Zusammenarbeit zwischen Software-Architekt und Requirements-Engineer ist zunächst nur eine Teilmenge der Notationselemente der Message-Sequence-Charts relevant. Wichtigster Aspekt der Message-Sequence-Charts ist, dass diese vor allem die Akteure und ihre Interaktionen mit dem System hervorheben.

Grundsätzlich werden in einem Message-Sequence-Chart Nachrichten und Instanzen abgebildet. Eine Instanz kann hierbei z.B. ein System oder ein Akteur sein. Als Nachricht wird im Kontext der Message-Sequence-Charts der Austausch einer Information bezeichnet. Dies können z.B. Signale oder Daten sein, die zwischen den Instanzen versendet werden (Silkora Referenz S.38).

–TODO– Abbildung zu MSC und Beschreibung

Neben den einfachen Message-Sequence-Charts gibt es High-Level-Message-Sequence-Charts (HLMSC), die es ermöglichen, eine Komposition mehrerer Message-Sequence-Charts zu bilden und so komplexere Zusammenhänge darzustellen. Grundsätzlich besteht ein HLMSC aus einem Start- und Endknoten, einem oder mehreren Verweisknoten und einer Menge von Kanten. Die Start- und Endknoten dienen hierbei der Begrenzung des Anwendungsfalles. Die Verweisknoten sind als Verweise zu einfachen Message-Sequence-Charts oder weiteren HLMSC zu sehen. Ungerichtete Kanten dienen der Darstellung der Komposition und mithilfe von gerichteten Kanten lässt sich Iteration darstellen.

–TODO– Abbildung zu HLMSC

Wenn Anwendungsfälle in Kompositionen zusammengefasst werden, lässt sich argumentieren, dass man bei hinreichenden Kompositionen in den Bereich der Ziele gelangt. Somit wird deutlich, dass der Szenario-basierte Ansatz implizit schon eine Betrachtung der Ziele fordert.

6) *Kombination der Ansätze*: Wenn die wesentlichen Ziele der Stakeholder bekannt sind, besteht die Möglichkeit, diejenige Architekturalternative auszuwählen, mit der die

Ziele am besten erfüllt werden können (Silkora Referenz S.18).

Sowohl Ziel-basierte Ansätze als auch Szenario-basierte Ansätze sind alleinstehend nicht ausreichend, um eine Grundlage für einen guten Architekturansatz auf der Basis von Anforderungen zu generieren. Hierfür ist es notwendig, die Ansätze zu kombinieren.

Ein Beispiel für eine solche Kombination ist der COSMOD-RE (sCenario and gOal based System development method) Ansatz.

a) *COSMOD-RE*: –TODO– Abbildung zu den Subprozessen des Designs

7) *Bewertung*: In der Betrachtung Ziel-basierter Ansätze wird deutlich, dass diese allein nicht ausreichen, um weitreichende Architekturentscheidungen zu treffen. Hauptproblem ist hier, dass mithilfe der Ziele lediglich angegeben wird, was am Ende aus der Sicht der Stakeholder erreicht werden soll und nicht konkret, welche Rolle in dem System welche Aufgaben hat und wie das Ziel zu erreichen ist.

Szenario-basierte Ansätze deuten an, dass diese alleinstehend ebenfalls nicht ausreichen, um eine Software-Architektur zu entwerfen, da hier vor allem das Problem besteht, dass alleinstehende Szenarien ohne Zusammenhang am Ende der Entwicklung keine konsistentes System ergeben können. Durch Ziele werden sie in den richtigen Kontext gesetzt, was bei Szenario-basierten Ansätzen die Formulierung von Zielen voraussetzt.

Gemeinsame Ansätze wie der COSMOD-RE Ansatz verbinden Szenario- und Ziel-basierte Ansätze und ermöglichen es so mithilfe eines iterativen Vorgehens, die Grundlage für einen Architekturentwurf zu generieren, dass sowohl die Wünsche des Kunden widerspiegelt als auch ausführlich genug ist.

#### IV. AUSWERTUNG DER METHODEN

Das Beste ist, wenn beide Parteien ausreichendes Know-How über die andere Fachdisziplin besitzen. Dadurch können viele Probleme, die in der Problemstellung unter XXX angesprochen wurden, umgangen werden. Experience -> [2], [4]

##### A. Vergleich

”Einführungstabelle” - Vergleich der Hartfacts (z.B. Input, Output, Mehrwert, Rollen, Zeit, Projektgröße, Randbedingungen etc.).

”Matrix” - Behandelte Probleme vs. Methoden

a) A - B: Vor und Nachteile im Bezug auf: Parallelität, Probleme, (Zeit, Projektgröße etc.)



b) A - C: Vor und Nachteile im Bezug auf: Parallelitt, Probleme, (Zeit, Projektgre etc.)

c) A - D: Vor und Nachteile im Bezug auf: Parallelitt, Probleme, (Zeit, Projektgre etc.)

d) B - C: Vor und Nachteile im Bezug auf: Parallelitt, Probleme, (Zeit, Projektgre etc.)

e) B - D: Vor und Nachteile im Bezug auf: Parallelitt, Probleme, (Zeit, Projektgre etc.)

f) C - D: Vor und Nachteile im Bezug auf: Parallelitt, Probleme, (Zeit, Projektgre etc.)

## B. Bewertung

### V. FAZIT

TODO: ae, ue, und oe im Text ersetzen!

### VI. AUSBLICK

#### REFERENCES

- [1] E. Bjarnason, *Distances between Requirements Engineering and Later Software Development Activities: A Systematic Map*, Springer Berlin, 2013
- [2] J. D. Herbsleb, *Global Software Engineering: The Future of Socio-technical Coordination*, IEEE Computer Society Washington, 2007