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*—Bei der Realisierung eines Softwareentwicklungsprojektes besteht hufig eine Kluft zwischen den Software-Architekten auf der einen Seite und den Requirements Engineers auf der anderen Seite. Software Architekten sind zum Beispiel hufig mit dem Problem konfrontiert, dass Anforderungsdokumente nicht ausreichend sind um weitreichende Architekturentscheidungen in Bezug auf den Softwareentwurf zu treffen. Daher ensteht in diesem Fall fr den Software-Architekten hufig der Mehraufwand, dass dieser ber weitere Interviews ein przises Bild von der gewnschten Software Architektur erhlt. Dies resultiert hufig in Verzgerungen die sich dann darin wiederspiegeln, dass Termine nicht eingehalten werden knnen. Betreibt der Software Architekt diesen Mehraufwand nicht und trifft eigene Annahmen bezglich der Architektur der Software kann dies wiederum zu einer geringeren Akzeptanz des Kunden und im schlimmsten Fall zum Auftragsverlust fhren. Die Requirements Engineers wissen auf der anderen Seite wiederum nicht, welche Anforderungen konkret wichtig fr den Software Architekten sind, da ihnen die entsprechende Fachkenntnis fehlt. Somit knnen diese nicht zielfhrend die notwendigen Informationen mit dem Kunden erarbeiten. Ohne diese Informationen ist eine ausreichende Grundlage der Anforderungen fr den Architekturentwurf nicht gegeben. Um diese Kluft zu berbrcken ist es notwendig Verfahren zu ermitteln, die es Requirements Engineers ermglicht die richtigen Informationen einzuholen und die Zusammenarbeit mit den Software Architekten zu optimieren.**

Im folgenden werden Verfahren untersucht die das Potenzial haben die Zusammenarbeit zwischen Requirements-Engineer und Software-Architekt zu verbessern. Ferner wird berprft wo diese Verfahren ihre Strken und Schwchen offenbaren und wie diese gewinnbringend kombiniert werden oder sich gegenseitig ergnzen knnen. Das zentrale Problem der Zusammenarbeit scheint eine mangelnde Verstndigung zwischen den Requirements Engineer und dem Software Architekten zu sein. Software Architekten sehen sich gezwungen, entweder eine Architektur anzunehmen, oder weiterfhrende Gesprche mit dem Kunden zu fhren. 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 ermglichen ist es zunchst notwendig zu identifizieren, warum die Anfordungsdokumente, die der Requirements Engineer generiert, nicht ausreichen um einen vollstndigen Architekturentwurf herzustellen und wie Erkenntnisse des Software-Architekten wieder in die Anforderungs-Spezifikationen einflieen knnen. Auerdem wird recherchiert ob Zusammenarbeit und Kommunikation durch eine passende Tool-Untersttzung verbessert werden kann.

Um eine Optimierung zu ermglichen bieten sich verschiedene Vorgehensweisen an. Im folgenden wird eine Kategorisierung von Anforderungen vorgenommen, die Relevant fr die Architektur einer Software sein knnen. Diese Kategorisierung bietet Requirements Engineers die Mglichkeit speziellere Fragen zu der gewnschten 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 mglich 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 ermglicht eine engere Kommunikation zwischen Software-Architekt und Requirements Engineer. Anschlieend wird dieser Ansatz mit dem Twin Peaks Modell verglichen und beurteilt welches der beiden Verfahren den greren Mehrwert bringt oder ob diese sinnvoll kombiniert werden knnen.

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 Mglichkeit gegeben sein, abzuwgen welche in der individuellen Situation am besten geeignet sind.

I. Einführung

II. Problemstellung

Whrend der Zusammenarbeit zwischen Requirements Engineer und Software Architekt knnen vielfltige Probleme auftreten. Bei der Untersuchung mglicher Probleme lsst sich eine Kategorisierung dieser vornehmen. So sind einige Probleme bedingt durch den Menschen selbst, whrend andere Probleme sich aus den Anforderungen ergeben. Im folgenden

werden die kategorisierten Probleme genauer ausgefhrt.

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 bermittelt werden. Ein wichtiger Aspekt schlechter Kommunikation ist die Distanz zwischen den Akteuren innerhalb eines Projektes.

Nach [**?**] kann vor allem Distanz ein Grund fr schlechte Kommunikation sein. Distanzen in der Kommunikation zwischen Requirements Engineer und Software Architekt knnen die Koordination reduzieren, was wiederum Projektverzgerungen hervorrufen kann, sowie das nicht erfllen von Kundenwnschen. Distanzen werden nach [**?**] vor allem als rumliche Distanzen bei global agierenden Projektteams gesehen. Bei rumlichen Distanzen knnen nach [2] beispielsweise Begleitprobleme wie das Beherrschen einer Sprache aufkommen. So kann das schlechte Beherrschen einer Sprache zu Missverstndnissen fhren und Fehler in den Anforderungen hervorrufen. Zustzlich knnen 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, whrend der Anforderungsgewinnnung, keine Rcksprache mit dem Software Architekten hlt [**?**]. Ohne ausreichende Rcksprache kann beispielsweise eine Beeintrchtigung der Qualitt der architekturrelevanten Anforderungen auftreten [**?**]. Dies kann von fehlenden bis fehlerhaft Anforderungen reichen. Ein weiteres mgliches Problem ist, dass der Software Architekt dem Requirements Engineer nicht ausreichend vermittelt, welche Informationen er fr einen gltigen Architekturentwurf bentigt [**?**]. Aus diesen Kommunikationsproblemen knnen Missverstndnisse in den Verantwortlichkeiten folgen [**?**]. Auch hier ist eine mgliche Folge die negative Beeinflussung der Qualitt der Architekturanforderungen.

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, whrend 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 fhren. Dabei kann hier zwischen zwei Ebenen unterschieden werden. Einerseits knnen Probleme aus fehlendem Know-How ber die Methodik der jeweils anderen Rolle entstehen [**?**][**?**]. Hierbei knnen unter anderem Probleme bei der Planung und Umsetzung eines Projektes resultieren, da hierdurch potenziell relevante Vorgehensweisen oder Methoden nicht bercksichtigt werden knnen. Andererseits knnen 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 erhlt, die er bentigt [**?**]. Dies wird dann zum Problem wenn der Requirements-Engineer nicht wei welche Informationen der Software-Architekt fr die Umsetzung bentigt. Ferner knnen Missverstndnisse auftreten, wenn der Software-Architekt bei fachspezifischen Begriffen ein anderes Verstndnis hat als der Requirements-Engineer [**?**].

B. Qualität der Anforderungen

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

P4: *Zu restriktive / detaillierte architekturrelevante Anforderungen*
P5: *Ungenaue/ fehlende architekturrelevante Anforderungen*
P6: *Nicht klar hervorgehobene architekturrelevante Anforderungen*
P7: *Wechselwirkungen zwischen architekturrelevanten Anforderungen*

1) Zu restriktive / detaillierte architekturrelevante Anforderungen: Vor allem bei sehr groen Projekten kann es passieren, dass es tausende von Anforderungen, die fr die Architektur relevant sind, gibt [**?**]. Es besteht eine hohe Wahrscheinlichkeit, dass in einem solchen Fall viele Anforderungen Widersprche erzeugen und den Lsungsraum zu sehr einschrnken. Hier knnen restriktive oder detaillierte Anforderungen die Arbeit von Requirements-Engineer und Software-Architekt unntig einschrnken und dadurch erschweren.

2) Ungenaue / Fehlende architekturrelevante Anforderungen: In der Anforderungsgewinnung kann es passieren, dass wichtige architekturrelevante 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 zustzliche klrende Gesprche mit dem Kunden zu fhren. Auch hier kann es passieren, dass bei besonders groen Projekten mit mehreren tausenden Anforderungen, wichtige nicht erhoben werden [**?**]. Vor allem im Kontext der agilen Softwareentwicklung bedingt ein iteratives Vorgehen, dass manche architekturrelevanten Anforderungen erst whrend der Modellierung oder sogar Implementierung der Architektur gewonnen werden [**?**].

3) Nicht klar hervorgehobene architekturrelevante Anforderungen: Eines der Hauptprobleme architekturrelevanter Anforderung liegt darin, dass es blicherweise schwer ist diese zu identifizieren und zu spezifizieren [**?**]. Werden architekturrelevante Anforderungen fr den Software-Architekten nicht hervorgehoben, kann es passieren, dass dieser wichtige Anforderungen erst spt, wenn berhaupt wahrnimmt [**?**]. Dies kann zur Folge haben, dass wichtige architekturrelevante Entscheidungen nicht rechtzeitig getroffen werden knnen und zu einem Mehraufwand zu einem spteren Zeitpunkt fhren [**?**]. Zustzlich liegt ein Problem in der fehlenden Ausfhrung der Auswirkungen von architekturrelevanten Anforderungen auf die sptere Software-Architektur [**?**]. So kann hieraus folgen, dass wichtige Projektziele nicht erfllt werden, da der Zeitrahmen des Projektes berschritten wird.

4) Wechselwirkung zwischen Anforderungen: Unter Wechselwirkung zwischen Anforderung ist zu verstehen, dass Anforderungen teilweise voneinander abhngig sein knnen. Diese Abhngigkeit kann auf verschiedene Arten und Weisen erfolgen. Ist beispielsweise ein Ansatz gegeben, bei dem mehrere Abstraktionsstufen vorhanden sind kann es passieren dass wichtige Abhngigkeiten, die ber mehrere Abstraktionsstufen hinweg reichen nicht bercksichtigt werden oder weitere Probleme verursachen [**?**]. nderungen von Anforderungen auf verschiedenen Hierarchieebenen treffen kontinuierlich ein und knnen somit Einfluss auf die Architektur haben und erheblichen nderungsbedarf verursachen [**?**]. Auch ist es schwierig die Konsistenz und Nachverfolgbarkeit der Projektion von Anforderungen auf die Software-Architektur zu gewhrleisten, da eine Anforderung mehrere architekturrelevante Artefakte betreffen kann. Zustzlich kann ein architekturrelevantes Artefakt mehrere Anforderungen betreffen [**?**].

Neben den aufgefhrten Problemen gibt es weitere, die hier nicht nher behandelt werden. Darunter fllt zum Beispiel ein phasenbezogenes Requirements Engineering.

III. Untersuchung gegebener Methoden

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 durchgefhrt. 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 hervorsticht ist, dass es innerhalb der Design-Rounds eine klare Folge von Anweisungen gibt, die auszufhren sind um die Software-Architektur zu entwickeln. Hierbei ist relevant zu erwhnen, 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 hierfr ist, dass bereits primre funktionale Anforderungen und Szenarien erhoben sind. Dies bedeutet ADD findet nicht direkt in der Anforderungsgewinnung Anwendung, sonder erst danach. Insgesamt umfasst ADD sieben Schritte die innerhalb einer Design-Round auszufhren 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 desing concepts that satisfy the selected drivers
5: Instantiate architectural elements, allocate responsiblities and define interfaces
6: Sketch views and record design decisions
7: Perform analysis of current design and review iteration goal and achievement of desing purpose

Um bei ADD eine Design-Round durchfhren zu knnen sind jedoch zunchst einige Eingaben fr den Prozess vorzubereiten.

Diese sind:

- bergeordnete Zielstellung
- Primre funktionale Anforderungen
- Szenarien
- Einschrnkungen

a) Step 1 - berprfung der Eingaben: Zunchst muss sichergestellt werden, dass die bergeordnete Zielstellung fr die darauffolgenden Design-Aktivitten festgelegt ist. Diese kann beispielsweise die erstmalige Erstellung eines Design-Entwurfes oder die Verbesserung eines vorhandenen Architektur-Designs sein. Danach wird berprft, ob die fr die Design-Round relevanten Anforderungen und Szenarien korrekt sind. Hier ist unter anderem zu prfen ob alle relevanten Stakeholder bercksichtigt werden und ob die erhobenen Anforderungen richtig priorisiert sind. Zuletzt

muss noch geprft werden, ob es Einschrnkungen bezglich der Software-Architektur gibt, die in der Design-Round zu bercksichtigen sind.

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

B. Probing (JEM)

1) Beschreibung: TODO: ASFRs frher erklren??!

Da die beiden Bereiche des Requirements-Engineering und der Software-Architektur ein enormes Ma an Wissen und Fhigkeiten erfordern werden diese in der Regel von verschiedenen Teams betreut. Wie bereits genannt, entstehen hier hufig Probleme durch fehlendes technisches Know-How ber Software-Architektur spezifische Aspekte bei den Requirements-Engineers. Auch knnen diese oft nicht zwischen funktionalen Anforderungen (FR) und architekturspezifischen funktionalen Anforderungen (ASFR) unterscheiden. ASFRs sind FRs welche kritisch, sehr risikobehaftet, volatil und bei nderungen ein aufwendiges oder teures Refactoring mit sich bringen wrden oder einen anderweitig groen Impakt auf die zu konzipierende Software-Architektur htten. Durch das fehlende Know-How entstehen unvollstndige Anforderungs-Artefakte, in welchen wesentliche ASFRs fr den Software-Architekten fehlen.

Das Ziel des Probing ist es, Requirements Engineers mit Fragen auszustatten die eine Erhebung architekturrelevanter Anforderungen, den ASFRs, ermglichen. Diese Fragen werden Probing Questions (PQ) genannt. Software-Architekten stellen PQs meist intuitiv auf Erfahrung basierend.

TODO: PQ Kategorien und Typen aus (2).

TODO: PQ-Flow aus (1).

Paper so far:
(1) Probing for Requirements Knowledge to Simulate Architectural Thinking
(2) What you see is what you get: Understanding Architecturally Significant Functional Requirements
(3) Identifying Architecturally Significant Functional Requirements

2) Bewertung: Bewertung (Chancen / Grenzen) (Was bleibt offen) und Ausblick

To be continued ...

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

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:
2) Funktionsweise der Methode:
a) Randbedingungen:
b) Eingabe:
c) Vorgehensmodell:
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 eine 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 Bahavioral-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 Bahavioral-Goals in zwei Kategorien aufteilen, die Achive-Goals und die Maintain/Avoid-Goals. Die Achive-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: Funtionale Ziele, die sich damit beschäftigen User-Anfragen zu beantworten.
- Information: Funtionale 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 Zieldiagramm 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 Szenarienverläufe 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 fr 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 erfllt sein mssen dass die Erweiterung greift.

Anwendungsfalldiagramme stellen Akteure und Anwendungsflle dar. Dadurch wird veranschaulicht welche Akteure mit welchen Anwendungsfllen in Beziehung stehen und ob es wichtige Beziehungen zwischen verschiedenen Anwendungsfllen gibt. Grundstzlich ist es mglich Anwendungsfalldiagramme auf verschiedenen Abstraktionsstufen abzubilden um so sowohl fr das Gesamtsystem, als auch fr die Teilsysteme die Beziehungen zu betrachten. Um jedoch Szenarien przise zu spezifizieren reichen Anwendungsfalldiagramme nicht aus.

b) Message-Sequence-Charts: Mithilfe von Message-Sequence-Charts ist es mglich eine przise Spezifikation der verschiedenen Szenarien eines Anwendungsfalls zu generieren (Silkora Zitat S.37). Bei der Betrachtung der Zusammenarbeit zwischen Software-Architekt und Requirements-Engineer ist zunchst 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.

Grundstzlich werde 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 knnen 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 ermglichen eine Komposition mehrerer Message-Sequence-Charts zu bilden und so komplexere Zusammenhnge darzustellen. Grundstzlich 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 Anwendungsfalls. 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 lsst sich Iteration darstellen.

–TODO– Abbildung zu HLMSC

Wenn Anwendungsflle in Kompositionen zusammengefasst werden lsst sich argumentieren, dass man bei hinreichenden Kompositionen in den Bereich der Ziele gelangt. Somit wird deutlich, dass der Szenario-basierte Ansatze 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 Anstze als auch Szenario-basierte Anstze sind alleinstehend nicht ausreichend, um eine Grundlage fr einen guten Architekturansatz auf der Basis von Anforderungen zu generieren. Hierfr ist es notwendig, die Anstze zu kombinieren.

Ein Beispiel fr eine solche Kombination ist der COSMOD-RE (sCenario and gOal based SysteM develOpment methoD) Ansatz.

a) COSMOD-RE:

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 Qualittsanforderungen erstellt, die sich auf das Gesamtsystem beziehen. Die Architektursicht konzentriert sich hier auf die Definition von externen Systemschnittstellen. Hier definierte Artefakte sollen primr die Kommunikation mit Stakeholdern untersttzen.
2 Komponenten: Die Komponentenebene bezeichnet die Aufteilung des Systems in einzelne Komponenten aus denen sich dieses zusammensetzen soll. Fr jede Komponente werden funktionale Anforderungen und Qualittsanforderungen formuliert. Da in dieser Ebene die Basis fr die Systemarchtitektur 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 Hardwarekomponeten 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 vernachlssigen.

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

–TODO– Abbildung Zu den Sub-prozessen des Designs

7) Bewertung: In der Betrachtung Ziel-basierter Anstze 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 Anstze 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 knnen. Durch Ziele werden sie in den richtigen Kontext gesetzt, was bei Szenario-basierten Anstzen die Formulierung von Zielen voraussetzt.

Gemeinsame Anstze wie der COSMOD-RE Ansatz verbinden Szenario- und Ziel-basierte Anstze und ermglichen es so mithilfe eines Iterativen Vorgehens die Grundlage fr ein Architekturentwurf zu generieren, dass sowohl die Wnsche des Kunden widerspiegelt als auch ausfhrlich genug ist.

IV. Auswertung der Methoden

Das beste ist, wenn beide Parteien ausreichen Know-How ber die andere Fachdisziplin besitzen. Dadurch knnen viele Probleme, die in der Problemstellung unter XXX angesprochen wurden, umgangen werden. Experience -¿ [2], [4]

V. Fazit

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