

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 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 Verfahren untersucht die das Potenzial haben die Zusammenarbeit zwischen Requirements-Engineer und Software-Architekt zu verbessern. Ferner wird überprüft wo diese Verfahren ihre Stärken und Schwächen offenbaren und wie diese gewinnbringend kombiniert werden oder sich gegenseitig ergänzen können. 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 Anforderungsspezifikationen einfließen können. Außerdem wird recherchiert ob Zusammenarbeit und Kommunikation durch eine passende Tool-Unterstützung verbessert werden kann.

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.

I. EINFÜHRUNG

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 der Qualität der 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 besonders auf:

- Schlechte Kommunikation
- Konkurrierende Interessen
- Fehlendes Know-How

1) *Schlechte Kommunikation:* Die Probleme im Bezug auf eine schlechte Kommunikation sind als grundstzliches Problem in der Zusammenarbeit zwischen mehreren Personen zu sehen. In diesem Zusammenhang lassen sie sich in zwei Klassen aufteilen. Der ersten Klasse lassen sich Probleme zuordnen, bei denen die Gre des Kommunikationsflusses zwischen Requirements Engineer und Software Architekt nicht ausreichend ist. Der zweiten Klasse werden die Probleme zugeordnet, die aus einem beidseitigen Monolog entstehen. Unter einem beidseitigen Monolog ist hierbei zu verstehen, dass in der Kommunikation zwischen Requirements Engineer und Software Architekt kein richtiger Dialog stattfindet, sondern lediglich Informationen und Handlungsanweisungen ausgetauscht werden. ¡REWORK¿

Bei nicht ausreichendem Kommunikationsfluss zwischen Requirements Engineer und Software Architekt kann das Problem aufkommen, dass der Requirements Engineer, whrend der Anforderungsgewinnung, keine Rcksprache mit dem Software Architekten hlt. Ohne ausreichende Rcksprache kann beispielsweise eine Beeintrchtigung der Qualitt der architekturelevanten Anforderungen auftreten. Dies kann von fehlenden bis fehlerhaft Anforderungen reichen. Der Verursacher dieses Problems ist am ehesten der Requirements Engineer. Ein weiteres mgliches Problem ist, dass der Software Architekt dem Requirements Engineer nicht ausreichend vermittelt, welche Informationen er fr einen gltigen Architekturentwurf bentigt. Auch hier ist eine mgliche Folge die negative Beeinflussung der Qualitt der Architekturanforderungen.

In der zweiten Klasse werden Probleme gruppiert, bei denen die Gesprchspartner keinen zielfhrenden Dialog fhren, d.h. aneinander vorbei reden. Dies kann kann der Fall sein, wenn zwei Gesprchspartner sich nicht gegenseitig zuhren oder aber die besprochenen Inhalte anschlieend nicht bercksichtigen. Wenn ein Software Architekt einem Requirements Engineer zum Beispiel nicht zuhrt, kann es passieren, dass Anweisungen missverstanden werden und die konzipierte Software Architektur nicht den Wnschen des Kunden entspricht. Ein weiteres Problem ist, dass der Software Architekt die Vorgaben des Requirements Engineer ignoriert und die besprochenen Inhalte nicht bercksichtigt.

2) *Konkurrierende Interessen:* Unter konkurrierenden Interessen ist zu verstehen, dass die Hauptverantwortlichen des Entwicklungsprozesses der Software-Architektur in der Projektarbeit verschiedene Interessen verfolgen, die miteinander in Konflikt stehen. Die Verantwortlichen haben zwar eine gemeinsame Vision von dem fertigen Produkt, verfolgen aufgrund variierender Interessen jedoch eine unterschiedliche Art der Zielerreichung. So ist es das Ziel des Kunden am Ende der Entwicklung ein kostengnstiges Produkt zu haben mit dem er effizient Arbeiten kann. Hier ergeben sich auf der Ebene des Requirements-Engineers und des Software-Architekten bereits Unterschiede. Ziel des Requirements-Engineers ist es hierbei die Vision des Kunden in einem mglichst korrekten und vollstndigen Anforderungsdokument festzuhalten. Whrenddessen ist es das Ziel des Software Architekten eine Software-Architektur zu entwickeln, welche bestimmte Qualittsattribute wie Wartbarkeit und Erweiterbarkeit bestmglich erfllt.

Da die Bedeutung des Kunden in diesem Zusammenhang eine untergeordnete Rolle spielt, werden die den Kunden betreffenden Konflikte nicht nher untersucht.

Whrend der Requirements-Engineer die Interessen des Kunden mglichst genau dokumentiert und umgesetzt haben mchte ist es Ziel des Software-Architekten eine mglichst korrekte Software-Architektur in Hinblick auf seine Prferenzen zu realisieren. Diese Punkte knnen abhngig von der Ausfhrlichkeit und dem Inhalt der Anforderungsdokumente jedoch in Konflikt stehen.

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. *Qualitt der Anforderungen*

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

- Zu restriktive / detaillierte architekturelevante Anforderungen
- Fehlende architekturelevante Anforderungen
- Ungenaue / sich widersprechende architekturelevante Anforderungen
- Nicht klar hervorgehobene architekturelevante Anforderungen

1) *Zu restriktive / detaillierte architekturelevante Anforderungen:* Restriktive oder detaillierte Anforderungen können die Arbeit von Requirements-Engineer und Software-Architekt unnötig einschränken und dadurch erschweren. Zu genau beschriebene Anforderungen können verursachen, dass das Projekt nicht erfolgreich abgeschlossen werden kann, da die Zeitplanung dadurch erschwert wird. Ferner kann dadurch der Gesamtüberblick verloren gehen. Zu restriktive Anforderungen, können Widersprüche erzeugen, die eine korrekte Umsetzung unmöglich machen, da sie den Lösungsraum zu sehr einschränken.

2) *Fehlende architekturelevante Anforderungen:* In der Anforderungsgewinnung kann es passieren, dass wichtige architekturelevante Anforderungen nicht erhoben werden. 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.

3) *Ungenau / sich widersprechende architekturelevante Anforderungen:* Ungenau oder sich widersprechende Anforderungen verhalten sich ähnlich wie zu detaillierte oder restriktive Anforderungen. Wenn der Requirements-Engineer in den architekturelevanten Anforderungen wichtige Informationen nicht präzise oder überhaupt nicht formuliert, ergeben sich für den Software-Architekten Probleme während der Architekturkonzeption. Weiter können ebenso Probleme auftreten wenn die definierten Anforderungen sich widersprechen. Auch hier muss der Software-Architekt zusätzliche Rücksprache halten um die problematischen Anforderungen zu klären. Alternativ kann dieser innerhalb seiner Entscheidungskompetenz eine eigene Entscheidung treffen, mit der Gefahr, dass diese im späteren Verlauf unerwünschte Konsequenzen nach sich ziehen.

4) *Nicht klar hervorgehobene architekturelevante Anforderungen:* 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.

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

A. ADD 3.0 (FDD)

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:*

B. *Probing*

C. *Twin Peaks*

D. *Ziel- und Szenario-basierte Ansätze (FDD)*

1) *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

Anstzen an. Die Kombination ist deswegen von Relevanz, weil ein Ansatz allein nicht ausreichen kann um die Anforderungen in angemessener Weise zu erheben.

2) *Ziel-basierte Anstze*: Ziel-basierte Anstze zielen vorrangig darauf ab, ein umfassendes Verstdnis der Wnsche 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 Anstzen vor allem darauf ankommt, zu verstehen, welche Vision der Stakeholder von dem zukünftigen System hat. Bei der Erfassung dieser Vision ist ein natrlichsprachlicher Ansatz fehlerbehaftet, da hier sehr aufwändige manuelle Konsistenzprfungen notwendig wren. Deswegen bieten sich hier vor allem Modell-basierte Anstze an.

Ein gutes Beispiel fr ein Modell-basierten Ansatz ist der KAOS-Ansatz. Dieser Ansatz bietet den Vorteil, dass er mit wenigen przise 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 fr die Konzeption softwareintensiver eingebetteter Systeme geeignet, was eine verzahnte Entwicklung von Anforderungen und Architektur ber mehrere Abstraktionsstufen hinweg ermoglicht (Silkora Referenz S.31).

a) *KAOS*: Lamsweerde (Lamsweerde Referenz) beschreibt einen modellbasierten Ansatz zur Darstellung von Zielen und den Referenzen innerhalb von Zielen. Hierfr muss zunchst 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. Gltig ist eine endliche Menge von Verhaltensweisen des Systems.

Grundsztzlich 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 ber einen definierten Zeitraum hinweg einen definierten Zustand aufrechterhlt, oder einen definierten Zustand vermeidet.

Soft-goals beschreiben Prferenzen innerhalb von gltigen Systemverhaltensweisen. Diese lassen sich zunchst in funktionale Ziele und nicht funktionale Ziele aufteilen. Die funktionalen Ziele knnen die folgenden Kategorien haben:

- Satisfaction: Funktionale Ziele, die sich damit beschftigen User-Anfragen zu beantworten.
- Information: Funktionale Ziele, die damit beschftigt sind User ber wichtige Systemzustnde zu informieren.
- Stim-response: Funktionale Ziele, die sich damit beschftigen 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 zunchst durch eine Reihe von Eigenschaften charakterisiert. Diese knnen unter anderem Name, Definition, Quelle, Zielkategorie und Prioritt sein.

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

4) *Kombination der Anstze*: Wenn die wesentlichen Ziele der Stakeholder bekannt sind, besteht die Mglichkeit, diejenige Architekturalternative auszuwhlen, mit der die Ziele am besten erfllt werden knnen (Silkora Referenz S.18).

5) *Bewertung*:

IV. AUSWERTUNG DER METHODEN

V. FAZIT

VI. AUSBLICK

REFERENCES

- [1] H. Kopka and P. W. Daly, *A Guide to L^AT_EX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.