Formalisierung von Requirements durch Nutzung von Templates

Christian Kühl
Matrikelnummer:4233240
christian.kuehl87@googlemail.com

in Zusammenarbeit mit Berner&Mattner
Betreuer: Bernhard Kaiser

Fachbereich Mathematik und Informatik

Erstgutachter: Prof. Dr. Lutz Prechelt
Zweitgutachter: Prof. Dr. Elfriede Fehr

Berlin, den 29.12.2010

Eidesstattliche Erklärung

Hiermit versichere ich an Eides Statt, dass die vorliegende Bachelorarbeit von niemand anderem als meiner Person selbst verfasst wurde und dabei keine anderen als die angegebenen Quellen und Hilfsmittel verwendet wurden.
Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 29.12.2010

Christian Kühl

Kurzzusammenfassung:

Diese Arbeit stellt einen neuen Ansatz zur Erstellung von qualitativ besseren Anforderungen vor. Die Idee ist das Verwenden von variablen Templates. Man benutzt Templates, die vorher in verschiedene Klassen eingeteilt wurden, um daraus Anforderungen zu erstellen. Jedes Template besitzt dabei optionale Teile, sowie Variablen. Dadurch entsteht eine Variabilität eines Templates. Die einzelnen Variablen in einem Template werden mit Hilfe von Formalen Regeln durch die entsprechenden Elemente ersetzt. Somit ermöglicht man dem Requirements Engineer mit Hilfe einer kontrolliert-natürlichen Sprache Anforderungen zu entwickeln, die gemäß den Qualitätskriterien eine bessere Qualität aufweisen. Das Resultat der Methodik wurde durch eine wissenschaftliche Evaluation anhand des zusätzlich entwickelten Prototypen getestet.

Abstract:

This work presents a new approach for creating requirements, having a better quality. The idea is to use variable templates. The methodology uses templates in order to develop requirement. The templates were previously classified into different classes. Each template has optional parts, as well as variables. This results in a variation of a template. The individual variables in a template will be replaced by the appropriate elements using the specified formal rules . Thus, it allows the requirements engineer to develop requirements using a controlled natural language. The resulting requirements have a better quality according to the quality criteria. The result of the methodology was tested by a scientific evaluation by using the additional developed prototype.

Inhaltsverzeichnis

0. Motivation

Seit jeher bilden Pläne, Masken und Templates die Grundlage für Architekten, Maschinenbauer und Chipentwickler. In der Software Entwicklung ist der Begriff der Softwarearchitektur oder auch Designmuster gebräuchlicher. Jedoch werden diese Grundlagen hauptsächlich nur im Bereich Design und Implementierung eingesetzt.

Der Gebrauch von Templates bekommt eine immer größer werdende Bedeutung im Bereich der Anforderungserstellung. Man geht immer mehr dazu über die Anforderungen nicht mehr mit dem vollen Umfang der natürlichen Sprache zu definieren, sondern viel mehr auf Schablonen zurückzugreifen um einheitliche und verständliche Anforderungen zu erstellen. Dadurch entsteht eine Vielzahl neuer Möglichkeiten mit den Anforderungen zu arbeiten [Rupp10].

Obwohl der Gebrauch von Templates an Bedeutung gewinnt, gibt es momentan nur zwei sehr unterschiedliche Ansätze. Einmal der formale Ansatz eine eigene formale Sprache zu definieren um Templates zu schreiben. Diese werden dadurch jedoch weniger verständlich. Der andere Ansatz ist der Gebrauch von syntaktischen Schablonen, die jedoch nur den Satzbau vorgeben und damit noch zu viel Variabilität zulassen.

Der Inhalt dieser Arbeit soll einen Ansatz aus beiden Varianten enthalten und einen kleinen Beitrag zur Verbreitung von Anforderungstemplates leisten. Es werden Templates verwendet, die durch formale Regeln soweit erweitert werden bis eine korrekte Anforderung entstanden ist.

1. Einleitung

1.1 Aufgabenstellung und Lösungsansatz

Gegenstand dieser Bachelorarbeit ist die Erarbeitung einer Methodik zur Verbesserung der Qualität natürlichsprachlicher Anforderungen mittels Formalisierung durch das Benutzen von Anforderungstemplates. Die Anforderungsformulierung soll erleichtert werden, so dass Anforderungen schneller erstellt werden können und dennoch die Qualitätsmerkmale[1], die an Anforderungen gestellt werden, auch einhalten.

Teil der Aufgabenstellung ist die Erstellung einer Menge von Templates. Diese entstehen durch Nachforschungen in der Literatur und durch Untersuchungen realer Anforderungsspezifikationen.
Weiterhin müssen formale Regeln entwickelt werden, damit aus den Templates korrekte Anforderungen entstehen. Diese werden durch das Begutachten und Analysieren von Kundenanforderungen ermittelt und mit Hilfe der Backus Naur Form niedergeschrieben.
Erst durch die Kombination von Templates und formalen Regeln entstehen fertige Anforderungen, so wie sie auch in der Industrie gefunden werden.

Zur Validierung ist ein prototypisches Werkzeug für die Erstellung und Verwaltung von Templates , sowie das Erstellen von Anforderungen aus diesen Templates zu erstellen. Am Schluss wird das Werkzeug von Requirements Engineers in Sachen Funktionalität und Ergonomie bewertet werden.

1 Merkmale, die eine Anforderung haben sollte (siehe Kapitel 2.1)

1.2 Aufbau der Arbeit

In Kapitel 2 erfolgt eine Erklärung der Grundlagen. Dies beinhaltet die Einführung in den Anforderungsbegriff, Grundlagen des Requirements Engineering, der System Begriff und die informatischen Grundlagen.

Danach erfolgt in Kapitel 3 ein Überblick über den momentanen Stand der Wissenschaft und Technik im Bereich des Anforderungsmanagement unter besonderer Berücksichtigung der Automobilbranche.

In Kapitel 4 wird die Methode erläutert und in Kapitel 5 erfolgt die Erklärung der use cases. Kapitel 6 beinhaltet die Fakten zum Prototypen. Die Evaluation der Methode geschieht in Kapitel 7. Kapitel 8 beinhaltet das Fazit der Arbeit und bietet einen kleinen Ausblick darauf, inwiefern die entwickelte Methodik erweitert werden kann.

2.Grundlagen

2.1 Einführung in den Anforderungsbegriff

2.1.1 Definition

Eine Anforderung ist:

> „Eine Bedingung oder Fähigkeit, die ein System oder Teilsystem erfüllen muss, beziehungsweise von einem Benutzer (Person oder System) zur Lösung eines Problems oder Erreichung eines Ziels benötigt wird."
>
> [Quelle: Klaus Pohl & Chris Rupp]

Somit bilden Anforderungen die Grundlage für jedes Projekt. Sie definieren die beinhaltenden Aspekte des Gesamtsystems, welches entwickelt werden soll. Die Anforderungen bilden weiterhin die Basis für wichtige Prozessbereiche, die da wären Projektplanung, Risikomanagement und Akzeptanztest.

2.1.2 Arten

Man unterteilt Anforderungen in funktionale und nicht funktionale Anforderungen. Dabei beschreiben die funktionalen was das System leisten muss, die nicht funktionalen beschreiben hingegen die Eigenschaften des Systems. Diese Einteilung ist jedoch kritisch zu sehen, da dadurch die nicht funktionalen Anforderungen für Personen oft nicht wichtig erscheinen. Diese Annahme ist jedoch falsch. Wenn die nicht funktionalen Anforderungen nicht erfüllt werden ist das entwickelte System genau nicht benutzbar, als wenn die funktionalen Anforderungen nicht erfüllt wurden.

2.1.3 Qualitätskriterien

Anforderungen sollten dabei nach [Rupp07] folgende Qualitätskriterien besitzen:

- vollständig: Jede Anforderung muss die zu erfüllende Funktionalität vollständig beschreiben.
- korrekt: Anforderungen sind korrekt, wenn sie den Vorstellungen der Stakeholder entsprechen.
- klassifizierbar: Für jede Anforderung sollte eine rechtliche Relevanz festgelegt werden. Dadurch wird auch gleichzeitig die Bedeutung für die Vertragspartner festgelegt. Außerdem wird die Einklagbarkeit als rechtlich verbindlicher Vertragsbestandteil sicher gestellt. Eine Klassifizierung könnte verbal durch Modalverben (muss, sollte, wird) geschehen.
- konsistent: Eine Anforderung darf in sich selbst keinen Widerspruch haben und sich gegenüber aller anderen Anforderungen ebenfalls nicht widersprechen.
- prüfbar: Anforderungen müssen so formuliert werden, dass sie testbar bzw. validierbar sind. Es muss also eine Möglichkeit geben sie durch eine Messung nachweisen zu lassen. Sie sollten so formuliert werden, dass sie sich leicht in einen Testfall überführen lassen, z. B. durch Angabe von Grenzen.
- eindeutig: Eine Anforderung muss so formuliert sein, dass es nur eine Interpretationsmöglichkeit gibt. Sie darf nicht mehrere Interpretationen erlauben.
- verständlich: Alle Anforderungen müssen von den Stakeholdern verstehbar sein. Dies kann durch Verwendung einer einfachen, konsistenten und genauen Ausdrucksweise bei der Formulierung geschehen.
- gültig und aktuell: Eine Anforderung muss beschreiben, was der Auftraggeber sich vorstellt. Ändert sich ein Aspekt den die Anforderung beschreibt, muss sich die Anforderung anpassen.
- realisierbar: Jede Anforderung muss im Rahmen der technischen Möglichkeiten umsetzbar sein. Deshalb ist es auch ratsam beim Erstellen der Anforderungen mit den Entwicklern zusammen zu arbeiten.
- notwendig: Die Anforderungen sollen Leistungen fordern, die für das endgültige Produkt wichtig sind.
- verfolgbar: Anforderungen müssen über den gesamten Systemlebenszyklus eindeutig identifizierbar sein. Bei jeder Anforderung muss eindeutig erkennbar sein, wodurch sie realisiert wurde. Umgekehrt muss jeder Implementierung oder Umsetzung eine Anforderung zu Grunde liegen.

Qualitätskriterien nach [ISO26262]

Es folgt eine Unterteilung in einzelne Anforderungen und einer Menge von Anforderungen.

Eine einzelne Anforderung sollte folgende Bedingungen erfüllen:

- eindeutig: Eine Anforderung ist eindeutig, wenn es ein einheitliches Verständnis zur Bedeutung der Anforderung gibt.
- verständlich: Eine Anforderung ist verständlich, wenn der Leser (Stakeholder oder Verwender der Anforderung) diese versteht.
- atomar: Eine Anforderung ist atomar, wenn sie so formuliert sind, dass es nicht möglich ist sie in mehr als eine Anforderung zu zerlegen.

- Intern konsistent: Eine Anforderung ist in sich konsistent, wenn sie sich in sich selbst nicht widerspricht.
- Machbar: Eine Anforderung ist machbar, wenn sie mit den gegebenen Bedingungen implementiert werden kann.
- Prüfbar:

Eine Menge von Anforderungen sollte folgende Bedingungen erfüllen:

- hierarchische strukturiert: Eine Menge von Anforderungen ist hierarchisch strukturiert, wenn die Anforderungen in verschiedene Level aufgeteilt sind.
- Organisatorisch strukturiert gemäß eines Schemas: Eine Menge von Anforderungen ist organisatorisch strukturiert gemäß eines Schemas, wenn Anforderungen des gleichen Hierarchielevels einer gemeinsamen Gruppe angehören.
- Vollständig: Eine Menge von Anforderungen ist vollständig, wenn die Anforderungen eines Levels die Anforderungen des vorhergehenden Levels vollständig abarbeiten.
- Extern konsistent: Eine Menge von Anforderungen ist extern konsistent, wenn sich keine Anforderungen miteinander widersprechen.
- Keine Informationen sollen doppelt vorkommen: Der Inhalt einer Anforderung darf in keiner anderen Anforderung mehr auftauchen.
- wartbar: Eine Menge von Anforderungen ist wartbar, wenn sie modifiziert oder verändert werden kann.

Betrachtet man nun die Qualitätskriterien von Rupp und von der ISO26262 fällt einem erst einmal auf, dass sie sich recht ähnlich sind. Weiterhin sind es viele Kriterien, die so eine Anforderung erfüllen muss und es fällt oft recht schwer den Überblick zu behalten. Ich behaupte, dass nur wenige Requirements Engineers wirklich alle notwendigen Qualitätskriterien kennen und anwenden.

Außerdem sind einige etwas vage formuliert. Betrachtet man zum Beispiel die Korrektheit von Rupp ist es schwer zu erfassen, dass der Stakeholder nun zufrieden mit dieser Anforderung ist. Zumal in einigen Unternehmen weniger Kommunikation zwischen Requirements Engineers und Auftraggeber statt findet, wodurch diese Aussage abermals schwerer zu treffen ist.

Die Qualitätskriterien bilden die Grundlage für korrekte Anforderungen. Im Rahmen dieser Arbeit liegt der Fokus auf den folgenden Qualitätskriterien, da die zu entwickelnde Methodik diese verbessern soll:

- atomar: Man gibt dem Requirements Engineer ein Template vor, welches er durch formale Regeln zu einer fertigen Anforderung bauen kann. Somit ist gewährleistet, dass diese Anforderung auch atomar ist.
- verständlich: Da man den Requirements Engineers die formalen Regeln vorgibt mit denen die Anforderungen erstellt werden, enthalten diese immer die gleiche Ausdrucksweise. Es entsteht eine Art „gute Sprache“, das heißt es werden keine Floskeln mehr verwendet.
- prüfbar: Bedingungen werden durch Regeln wie zum Beispiel: „Wenn x größer ist als y“ ersetzt. Dadurch lassen sich in einem 2. Schritt leicht Testfälle aus diesen Anforderungen entwickeln, wodurch sie prüfbar werden.

- eindeutig: Da man die formalen Regeln vorgibt wird eine Anforderung, die eine Funktionalität beschreibt immer wieder auf gleiche Weise gebildet, wodurch egal wer sie schreibt immer die gleiche Anforderung entstehen wird. Somit wird jede Anforderung eindeutig, da sie nur eine Interpretationsmöglichkeit zulässt. Dies wird vor allem durch einheitliche, gleich bleibende Begriffe realisiert.

Die restlichen Aspekte werden durch diese Methode kaum bis gar nicht beeinflusst.

2.2 Grundlagen des Requirements Engineering

2.2.1 Definition

Requirements Engineering (RE) ist ein vielseitiges Gebiet. Es befasst sich mit der Systementwicklung, mit der Softwareentwicklung und mit dem Produktmanagement. Dabei wird ermittelt, was für Eigenschaften der Entwicklungsgegenstand erfüllen muss und welche Bedingungen dafür gelten müssen. Die Verwendung von Anforderungstemplates kann auf all diese Bereiche Einfluss nehmen. Verständliche Anforderungen ermöglichen es schnell zu erfassen, welche Bedingungen für den Entwicklungsgegenstand gelten müssen, wodurch die System- und Softwareentwicklung verbessert wird. Außerdem entstehen durch Anforderungstemplates eindeutige Anforderungen, egal ob zehn verschiedene Requirements Engineers eine Anforderung formulieren. [Ebert 10].

2.2.2 Voraussetzungen

Jede Anforderungsermittlung beginnt mit einem zugrundeliegenden Problem, welches gelöst werden muss. Dies kann mehrere Ursachen haben.
Einerseits besteht die Möglichkeit, dass Personen oder auch Organisationen mit der momentanen Situation unzufrieden sind und sie wollen eine Lösung für ihr Problem entwickelt haben.
Außerdem besteht die Möglichkeit, dass sich eventuell eine Marktlücke aufgetan hat, die eine Organisation jetzt abdecken möchte. Somit brauchen sie eine Lösung, damit sie womöglich als erster diese Marktlücke nutzen können.
Eine andere Möglichkeit ist die Ersparnis von Kosten, Zeit oder Ressourcen.

Damit ein Requirements Engineer dieses Problem erfolgreich lösen kann, sollte er genug Erfahrung in der entsprechenden Problemdomäne besitzen. Nur so ist er in der Lage die Probleme und Möglichkeiten zu identifizieren. Somit ist er in der Lage valide, konsistente und vollständige Anforderungen zu entwickeln und zu analysieren.
[Prech et East 10].

2.2.3 Ermittlungstechniken

Die Ermittlungstechniken kann man grob in vier Kategorien einordnen. Dazu gehören die traditionellen, die darstellungsorientierten, die sozialen und die Kognitiven Techniken.

Traditionelle Techniken sind zum Beispiel die Selbstprüfung (Introspection), Interview und Gruppendiskussionen.

Zu den darstellungsorientierten Techniken gehören zum Beispiel Szenarios und use cases. Ein Szenario beschreibt in kurzen Schritten einen Vorgang und stellt meist die Grundlage für einen use

case dar. Ein use case ist eine detaillierte Beschreibung eines Vorgangs. Er kann dabei auf verschiedenen Abstraktionsebenen erstellt werden. Auf diesen Bereich der Ermittlungstechniken kann die entwickelte Methode keinen Einfluss nehmen, da sie sich ausschließlich mit dem verbalen Formulieren von Anforderungen befasst.

Zu den sozialen Techniken zählt zum Beispiel die Eingliederung. Dies bedeutet man arbeitet für eine gewisse Zeit, zum Beispiel drei-sechs Monate in dem Unternehmen des Auftraggebers mit.

[Prech et East 10].

2.3 System

Ganz allgemein gesprochen ist ein System eine Menge von Objekten zusammen mit ihren Wechselwirkungen, die von ihrer Umgebung abgegrenzt werden können.

Somit bezeichnet man als System die Gesamtheit aller Gegenstände, egal ob konkret oder abstrakt, die zueinander in Wechselwirkung oder Beziehung stehen. Wenn die Gegenstände ihrerseits wieder Systeme sind spricht man von Subsystemen.

Ein System besitzt viele verschiedene Eigenschaften, zum Beispiel offen oder geschlossen, statisch oder dynamisch, diskret oder kontinuierlich und stabil oder instabil.

Die Beziehungen zwischen Komponenten werden als Relationen bezeichnet.

Der Begriff des Systems findet in der Informatik viele Anwendungen, zum Beispiel bei Hardwaresystemen in der technischen Informatik, bei Betriebssystemen in der theoretischen Informatik oder auch bei kontextspezifische Anwendungssystemen aus der angewandten Informatik [Broso et al 06].

2.4 Informatische Grundlagen

2.4.1 Backus-Naur-Form

Im Rahmen dieser Arbeit wird die Backus-Naur-Form (BNF) für die Definition der Formalen Regeln verwendet.

Ihren Namen hat die BNF von ihren beiden Erfindern John Backus und Peter Naur.

Die BNF ist ein Beschreibungsmittel für die Syntax von zum Beispiel höheren Programmiersprachen und wird auch als Metasprache bezeichnet. Es handelt sich dabei um eine textbasierte Alternative zu Syntaxgraphen. Mit Hilfe der BNF lassen sich kontextfreie Grammatiken darstellen.

Bei der BNF verwendet man anstelle des Gleichheitszeichen und des Vereiningungsoperators

::= als Definitionszeichen und

| als Oderzeichen.

Damit besteht jede BNF aus drei Bestandteilen:

- eine Variable links vom Definitionszeichen,
- das Definitionszeichen und
- eine Formel von Variablen oder Terminalsymbolen mit Oderzeichen verknüpft.

Die Variable links vom Definitionszeichen wird somit durch die Folge definiert. Es erfolgt dabei eine strikte Trennung zwischen Variablen und Terminalsymbolen, denn Variablen werden durch spitzwinklige Klammern nochmals gesondert dargestellt. Dabei sollte der Name der Variable in der Form gewählt werden, dass aus diesem die Bedeutung, die Struktur und die Verwendung der Variable deutlich wird.

Die terminalen Zeichen hingegen repräsentieren sich selbst. [ITWissen].

Abschließend ein kleines Beispiel

Term ::=<Zahl><Operator><Zahl>

Zahl::=<Ziffer>|<Ziffer><Zahl>

Ziffer::=0|1|2|3|4|5|6|7|8|9

Operator::= +|-|*|/

Beispielterme: 1+15, 4/7, 13792*3213

2.4.2 Unified Modeling Language (UML) Diagramme

UML ist eine Familie grafischer Notationen. Die Notationen sollen bei der Beschreibung und Entwicklung von Softwaresystemen helfen.

Diagramme die nach [Prech,Brueg,Dut] häufige Verwendung finden sind:

- Use case Diagramme
- Klassendiagramme
- Sequenzdiagramm
- Aktivitätsdiagramm
- Zustandsdiagramm

Im weiteren Verlauf werden die use case- und Aktivitätsdiagramme noch etwas genauer erklärt, da sie in dieser Arbeit Verwendung finden um die einzelnen Anwendungsfälle grafisch darzustellen..

2.4.2.1 Aktivitätsdiagramm

Ein Aktivitätsdiagramm stellt den Informationsfluss eines Systems dar. Dabei ist ein simples Aktivitätsdiagramm ähnlich einem Zustandsdiagramm. Der Unterschied besteht lediglich darin, dass bei einem Aktivitätsdiagramm die Zustände Aktionen, besser gesagt Funktionen des Systems, sind. Man unterscheidet dabei zwei verschiedene Zustände.

Der erste Zustand ist der „Action State“. Dieser kann nicht weiter verfeinert werden und stellt somit eine direkte Funktionalität des Systems dar

Der zweite Zustand ist der „Activity State“, welcher diesmal noch weiter verfeinert werden kann und somit selbst ein Aktivitätsdiagramm darstellt.[Prech,Brueg,Dut].

2.4.2.2 Use case und use case Diagramme

Ein use case beschreibt das Verhalten des Systems als Antwort auf einen Auftrag eines Hauptakteurs, der mit seinem Auftrag ein bestimmtes Ziel verfolgt.

Sie finden häufig Verwendung im Bereich der Anforderungsermittlung. Man formuliert also use cases um sich im Klaren darüber zu werden, was das System leisten muss. Also muss man sich mit den Beteiligten des Projekts auf ein Verhalten des Systems einigen und dieses Verhalten als use case darstellen. Use cases werden meist als strukturierter Text formuliert.

Somit beschreibt ein use case eine Menge von Szenarios, die in ihrem Ziel ähnlich sind.

Ein use case Diagramm kann nun dazu genutzt werden, um eine Übersicht der Zusammenhänge zwischen den vorher formulierten use cases zu beschreiben .

2.4.3 XML (Extensible Markup Language)

XML ist eine generische Auszeichnungssprache, die das erstellen von strukturierten Inhalten erlaubt. Version 1.0 erschien 1998 und ist so wie auch heute noch eine Teilsprache von SGML (Standard Generalized Markup Language). Inhalt wird dabei, ähnlich wie HTML in Form von Tags dargestellt.Jedoch sind diese bei XML nicht vorgeschrieben, wodurch sich leicht hierarchische Strukturen darstellen lassen. Dadurch erfolgt außerdem eine Trennung zwischen den Daten und der Präsentation dieser, da XML an sich für normale Personen schwer zu lesen ist.

Ein kleines Beispiel illustriert dabei die Einfachheit von XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<Anforderungsliste>
        <Überschrift >
                <Anforderung>Text</Anforderung>
                <Anforderung>Text</Anforderung>
        </Überschrift>
        <Überschrift>
                <Anforderung>Text</Anforderung>
        </Überschrift>
</Anforderungsliste>
```

XML findet in dieser Arbeit Verwendung bei Speicherlösungen für Anforderungslisten, Formale Regeln und Templatelisten[2].

2 Siehe Kapitel 6.3 Speicherlösungen und Datenstrukturen

3. Stand der Wissenschaft und Technik

3.1 Wissenschaftliche Ansätze

Es gibt mehrere Möglichkeiten Anforderungsschablonen zu erstellen, wobei zwei in diesem Kapitel genauer erläutert werden. Anschließend erfolgt eine Eingliederung dieser Arbeit.

3.1.1 Templates nach Christine Rupp

„Eine Anforderungsschablone ist ein Bauplan, der die syntaktische Struktur einer einzelnen Anforderung festlegt.“ [Rupp et al 07]

Bereits mit dieser Definition behaftet Rupp die Anforderungsschablonen als rein syntaktische Methode. Sie sagt aus, dass Anforderungsschablonen einem Requirements Engineer die Syntax vorgeben und der Requirements Engineer nur noch die notwendige Semantik definieren muss.
Christine Rupp möchte dabei die natürliche Sprache erhalten, nur soll diese in dem Maße eingeschränkt werden, dass „perfekte“ Anforderungen entstehen. Im Mittelpunkt ihrer Schablonen steht dabei das Prozesswort, welches ausdrückt was gemacht werden soll. Somit muss sich der gesamte Satzbau und der Inhalt der Anforderung an es anpassen. Ein weiterer Punkt ist die Systemaktivität. Dort trifft sie drei Unterteilungen:

- selbstständige Systemaktivität,
- Benutzerinteraktion und
- Schnittstellenanforderung.

Der nächste Aspekt ist das Festlegen der rechtlichen Verbindlichkeit. Da mit jeder Anforderungsliste und jeder Anforderung ein Vertrag einhergeht. Dabei bestimmen die Modalverben, wie wichtig die Anforderung sein soll. Sie benutzt dabei die Modalverben

1. muss.
2. sollte und
3. wird

Nun werden die Schablonen nur noch um ergänzende Objekte und Bedingungen erweitert. Letztendlich erhält man die Möglichkeit semantisch und syntaktisch korrekte Anforderungen zu bauen (siehe Abbildung 1).

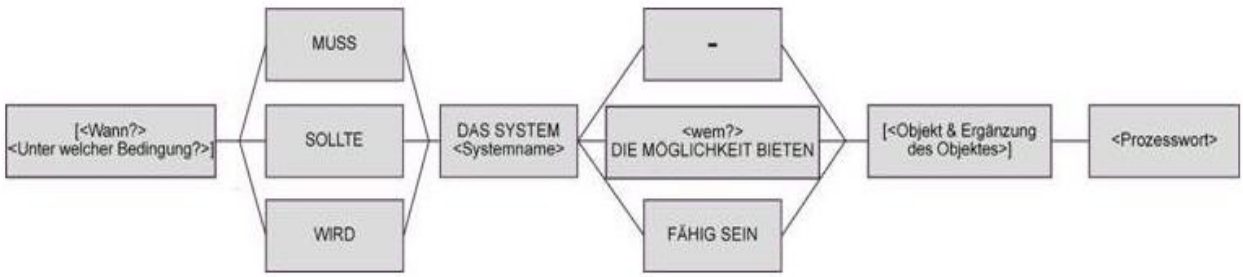

Abbildung 1: Fertige Anforderungsschablone nach [Rupp et al 07]

Beispiel:

[Wann? Unter welcher Bedingung] [muss, sollte, wird] das System [Wem? die Möglichkeit bieten, Fähig sein] [Objekt und Ergänzung des Objekts] [Prozesswort].

Daraus wird:
Wenn der Kunde eine Rechnung wünscht, muss das System dem Kellner ermöglichen die Rechnung zu drucken.
[Rupp et. al. 07]

3.1.2 Templates nach Toro et al.

Toro et al. Stellen genauso wie Rupp die natürliche Sprache in den Vordergrund. Bei ihnen erfolgt eine Unterteilung zwischen L- und R Patterns. Dabei stehen L-Patterns für linguistic Patterns und R-Patterns für Requirement Patterns. Mit linguistic Patterns meinen sie häufig verwendete Sätze in natürlich-sprachlichen Anforderungsschablonen. Diese linguistic Pattern ähneln in etwa den Anforderungsschablonen von Rupp.

Der Hauptaspekt liegt jedoch auf den Requirements Patterns. Dabei unterteilen sie diese in drei verschiedene Patterns:

- Information Storage Requirements,
- Functional Requirements Template und
- Non-Functional Requirements.

Sie benutzen dabei für jedes der drei Patterns eine Tabelle. Die Information Storage Requirements sollen dabei den Anwendern helfen herauszufinden welche Informationen von dem System gespeichert werden müssen. Dadurch entstehen viele R-Patterns, die sich lediglich in dem Punkt der Relevanz unterscheiden. So gibt es zum Beispiel welche für Kunden, Produkte, Aufträge, Rechnungen und so weiter.

Das Functional Requirements Template beschreibt use cases und unterstützt Kunden und Nutzer in der Frage, was sie mit den gespeicherten Informationen anstellen wollen. Hier gibt es jedoch nur genau vier R-Patterns, die da wären Create, Read, Update, Delete (CRUD-Patterns).

Der dritte Teil der Non-Functional Requirements deckt jetzt nach Toro et al lediglich den Rest, also die übrig gebliebenen Pattern ab. [toro et al 99]

Diese Methodik findet kann jedoch keine Verwendung bei eingebetteten System finden, da sie vorgegebenen Muster bei weitem nicht ausreichen um alle notwendigen Anforderungen zu beschreiben.

3.1.3 Formale Methoden

Formale Methoden sind ein auf Mathematik basierender Ansatz zur Spezifikation, Entwicklung und Verifikation von Software- und Hardwaresystemen. Der Grundgedanke ist dabei, die Erfüllung einer Anforderung nicht nur zu testen, sondern auch mathematisch zu beweisen. Dies ist ähnlich der Cleanroom Methode zur Programmierung.

Der Vorteil dieses Ansatzes ist, dass die Requirement Engineers sehr diszipliniert vorgehen müssen und dadurch Fehler eher entdecken können. Weiterhin sind die entwickelten Anforderungen und Systeme präziser, als wenn sie auf herkömmliche Weise entwickelt wurden.

Die Nachteile dieses Ansatzes sind die hohen Kosten die damit einher gehen, weshalb diese Methodik hauptsächlich Verwendung bei sicherheitskritischen System findet.

Ein Beispiel hierfür ist Esterel. Esterel ist eine synchrone Programmiersprache für die Entwicklung von komplexen reaktiven Systemen. [Coll99]

3.2 Stand der Technik

Es gibt nicht viele Werkzeuge, die das Erstellen von Anforderungen unterstützen sollen, aber DESIRe ist eines von ihnen.

3.2.1 DESIRe

DESIRe ist ein von der HOOD Group entwickeltes Werkzeug, dass Requirements Engineers beim Erstellen von natürlichsprachlichen Anforderungen unterstützen soll. Dadurch sollen mehrdeutige Wörter entfernt werden, indem DESIRe vordefinierte Fragen stellt. Durch das gedankliche Beantworten dieser Fragen macht sich der Requirements Engineer nochmals Gedanken über diese Anforderung und kann für sich validieren ob sie gemäß der Qualitätskriterien korrekt ist. Am Screenshot kann man das Vorgehen ziemlich genau erkennen. DESIRe durchsucht die Anforderung nach Wörtern, die vorher als kritisch (zum Beispiel mehrdeutig) definiert wurden und stellt dem Requirements Engineer die vorher definierten Fragen zu diesem Wort. Dieser kann nun entscheiden, ob er die Anforderung ändern will oder so belassen.

Abbildung 2: Screenshot DESIRe

3.2.2 weitere

Ein weiteres Tool, auf das ich aber nicht näher eingehen möchte, wurde von Frau Rupp und den Sophisten entwickelt um ihre entwickelte Methodik zu unterstützen.

Andere Programme wie zum Beispiel DOORS oder Requisite Pro dienen lediglich zur Verwaltung von Anforderungen. Sie unterstützen einem nicht beim Erstellen von Anforderungen gemäß den Qualitätskriterien.

3.3 Diskussion und Bewertung

Lange Zeit gab es beim Erstellen von Anforderungen aus Schablonen nur zwei Extrema. Ein Ansatz konzentriert sich darauf die Reviews zu erleichtern, wie zum Beispiel DESIRe. Der andere Ansatz sind formale Methoden, wie sie zum Beispiel von Esterel umgesetzt werden. Der Nachteil des formales Ansatzes ist, dass die natürliche Sprache gänzlich verloren geht. Es entsteht ein Wulst von formalen Regeln. Dadurch ist es für die Personen, die diese formale Methode nicht kennen nicht möglich die Anforderungen zu verstehen. Aber für alle anderen Personen sind die Anforderungen eindeutig. Die formalen Methoden finden meist nur in sehr sicherheitskritischen Anwendungen Verwendung.

Der Nachteil von DESIRe ist, dass erst einmal Anforderungen formuliert werden müssen bevor diese mit Hilfe von DESIRe einem Review unterzogen werden können. Weiterhin habe ich beim testen von DESIRe ein weiteres Manko festgestellt. Es ist nicht in der Lage Zusammhänge zu erkennen. Die Methode von DESIRe wird einfach stupide für jedes Wort angewendet, wodurch es bei schon qualitativ hochwertigen Anforderungen schnell lästig werden kann. Die Fragen werden konsequent weitergestellt, obwohl die Antworten bereits in der Anforderung stehen.

Die Methode von Frau Rupp bildet in meinen Augen eine Grundlage auf der man aufbauen kann, aber für die tatsächliche Benutzung noch erweitert werden muss.

Mittlerweile sind aber auch viele Forschungen gemacht worden einen Mittelweg zwischen diesen beiden Extrema zu finden. Ein Ansatz ist zum Beispiel der Artikel von Herr Holtmann [Holt10]. Dieser nennt die Probleme, die entstehen, wenn man Anforderungen mit reiner natürlicher Sprache schreibt und wie diese durch Verwendung von Satzmustern behoben werden können. Dabei werden den Requirements Engineers Anforderungssätze vorgegeben und sie müssen in diesen die entsprechenden Satzteile verändern, so dass eine vollständige Anforderung entsteht. Herr Holtmann legt dabei großen Wert darauf, dass die Anforderungen so geschrieben werden, dass es später möglich ist automatisch ein Modell daraus zu entwickeln. Er gibt dafür eine Unterteilung in 4 Kategorien an und erwähnt das momentan circa 100 verschiedene Satzmuster existieren. Ein Beispiel eines Satzmuster ist:

Systemzustandswechsel:
„((Wenn | Sobald) im | Vom) [Zustand] (Startzustand | <Ursprungszustand>) ((des Systems „<System>“ [(das Ereignis <Ereignis> eintritt | die Zeit abgelaufen ist) [und die Bedingung <Bedingung> erfüllt ist], geht es) | (geht das System „<System>“)) in den [Zustand] (<Zielzustand> | Endzustand) über [; (parallel dazu | zugleich) wird [die Funktion „<Funktion>“ (gestartet | aktiviert | reaktiviert | gestartet oder reaktiviert)]]. [Dies darf [minimal <Min-Zeit>] [und] [maximal <Max-Zeit>] dauern.] „

Dabei sind | Auswahl einer Alternative, [] optionale Teile, <> Stelle, in der Aufzählungen oder gewisse Freitextanteile eingesetzt werden können.
Meiner Meinung nach sind diese Satzmuster zu groß und zu komplex um sie wirklich anzuwenden. Es ist schwer festzustellen welche Anforderungen mit diesem Satzmuster entwickelt werden können und erst durch Probieren sind die Ausmaße wirklich erkennbar. [Holt10]

Der Vorteil von Anforderungsschablonen ist, dass Anforderungen in natürlicher Sprache geschrieben werden, aber durch einen beschränkten Wortschatz in der Form vereinfacht, dass sie leichter zu verstehen sind und auch eindeutiger werden. Durch die Vorgabe der Syntax erhalten die Anforderungen eine einheitliche Struktur, wodurch einem auch die Möglichkeit geboten wird Anforderungen einfacher wieder zu verwenden. Zur Erstellung solcher Anforderung ist jedoch ein geeignetes Werkzeug unabdingbar, da das Erstellen von Hand für den Requirements Engineer mühsam wäre.

Der Ansatz dieser Arbeit ist ähnlich dem von Herrn Holtmann, jedoch verfolgt sie ein anderes Ziel. Es geht darum Anforderungen einheitlicher, leichter, verständlicher und auch schneller zu formulieren ohne dabei auf natürliche Sprache zu verzichten. Somit steht am Ende nicht das Modell im Vordergrund sondern die Anforderungsliste. Außerdem wird bei Herrn Holtmann nicht erwähnt wie die Variablen in den Satzmustern gewählt werden. Vielleicht eine hierarchische Struktur, vielleicht aber auch eine große Liste. Aus den Screenshots lässt sich lediglich erahnen, dass wahrscheinlich eine hierarchische Struktur verwendet wird.

3.4 Offene Probleme

Die Analyse ist der erste Schritt der Systementwicklung und entscheidet somit maßgeblich über den Erfolg oder Misserfolg des Projektes. Nach wissenschaftlichen Studien wird das Entfernen von Fehlern in späteren Entwicklungsphasen immer um ein vielfaches teurer als in der Phase, in der der Fehler entstanden ist. Rupp [Rupp 2006] gibt zum Beispiel an, dass sich 65% der schwerwiegenden Fehler auf fehlerhaftes Verhalten in der Analyse zurückführen lassen.

Man bräuchte somit eine Methode, welche einem Requirements Engineer ermöglicht den Qualitätskriterien gemäß korrekte Anforderungen mit Hilfe von Templates zu erstellen.

4.Die Methodik der variablen Templates

4.1 Vorstellung der Methodik

Diese Arbeit schlägt einen neuen Ansatz zur Verwendung von Templates im Bereich des Requirements Engineering vor. Sie soll einen Mehrwert für die Forschung und Entwicklung für Methoden zur besseren Formulierung von Anforderungen beisteuern. Dabei stellt die Verwendung von variablen Templates eine neue Möglichkeit zur Erstellung von Anforderungen zur Verfügung.

Ein Beispiel für ein variables Template ist:

„The [component] shall {detect [failure]}{test [function]}."

Für das schreiben der Templates wurde eine Templatemetasprache gewählt. Dabei stellen die [] die Variablen dar, die {} die optionalen Teile und der Rest ist konstant in jeder Anforderung, die aus diesem Template entsteht.

Wie schon in der Einleitung beschrieben, geht es darum den Qualitätskriterien genügende, natürlichsprachliche Anforderungen zu erstellen. Um dies zu erreichen, bedient man sich der Hinzunahme von Templates. Damit aus diesen Templates vollständige Anforderungen entstehen müssen die Variablen gemäß den Regeln[3] abgeleitet werden. Zusätzlich muss eine Entscheidung getroffen werden welche optionalen Teile des Templates hinzugefügt werden sollen und welche nicht.

Somit entsteht aus dem variablem Template von eben eine Anforderung der Form:

„*The SW shall detect Buffer Overflows."*

Es wurde nur der optionale Part 1 gewählt, wodurch Part 2 wegfällt. Die Variablen [component] und [failure] wurden gemäß den Regeln durch SW und Buffer Overflow ersetzt.

Die Templates und die Regeln, wie aus ihnen Anforderungen werden, zusammen, geben dem Requirements Engineer eine eingeschränkte Menge der natürlichen Sprache vor. Es entsteht eine Controlled Natural Language (CNL).

4.1.1 Ziel der Methodik

Das Ziel der entwickelten Methodik ist die Verbesserung der Qualität von Anforderungen. Dabei sollen durch die Variabilität, durch Regeln und durch Listen aus möglichst wenig Templates möglichst viele Anforderungen entwickelt werden. Die Anforderungen besserer Qualität sollen außerdem mit weniger Zeitaufwand für die Requirements Engineers entwickelt werden können.

4.1.2 Einführung

Die Entwicklung der Methodik gemäß den Anforderungen werden am Beispiel der Automobilbranche durchgeführt. Diese Auswahl wurde getroffen weil der Arbeitgeber als Berater der Automobilbranche fungiert. Die Methodik ist aber ohne weiteres in anderen Industriebereichen einsetzbar, nur mit abweichenden Templates und Regeln.

Die Anforderungen in der Automobilbranche sind heutzutage überwiegend informell und verbal vorhanden. Sie sind in Prosatext geschrieben und Templates oder andere Regeln werden, wenn überhaupt, nur intuitiv angewandt. Oft werden dabei Tools wie DOORS oder auch Requisite Pro verwendet. Andere Möglichkeiten für die schriftliche Weitergabe sind Fließtexte in Word oder atomisierte Anforderungen in Excel und Word.

3 Formale Regeln siehe Kapitel 4.1.6

4.1.3 Entwickeln der Templates

Beim Entwickeln der Templates wurde das Ziel gesetzt, für jede Klasse von Anforderungen[4] eine kleine, übersichtliche Menge von Templates bereitzustellen. Diese Menge sollte ungefähr zwischen drei und sieben Templates liegen. Dadurch fällt es den Requirements Engineers leichter, das gesuchte Template zu finden. Weiterhin sollten diese Templates so universell sein, dass sie einen großen Prozentsatz von echten Kundenanforderungen abdecken können und durch die Variabilität ein breites Spektrum an Anforderungen abdecken.

Somit war es notwendig für das Entwickeln der Templates Kundenanforderungen zu sichten. Dafür wurde eine Menge von typischen Kundenanforderungen aus realen Projekten von Kollegen erstellt. Diese Menge wurde dann analysiert. Dazu erfolgte anfangs eine Einteilung der Anforderungen gemäß ihrer Möglichkeit sie durch Templates darzustellen in vier Kategorien, gut, mittel, schwer und speziell. Gut bedeutet die Anforderung kann eins zu eins in ein Template umgewandelt werden. Bei mittleren Anforderungen mussten diese vorher etwas umformuliert werden, damit sie einem Template entsprechen, was ein Requirements Engineer ohne besondere Projektkenntnisse können sollte. Schwere Anforderungen konnten nur mit großen Aufwand und unter Mitwirkung eines Fachexperten umgeformt werden, ohne den Inhalt zu verfälschen. Spezielle Anforderungen sind entweder zu lang (verletzen somit höchstwahrscheinlich die Atomizität), schlecht strukturiert oder passen zu keiner Kategorie von Anforderungen.

Qualität	Beispiel
Gut	*The system shall have an overall weight less than 8.5 kg*
Mittel	*Before signalling to be ready for power on command, the SW shall make sure that all onboard HW has been initialized successufully.*
Schlecht	*When maximum current is reached, it has to switch to current controlled mode. Reduction of current shall only be allowed if a system derating is necessary.*
Speziell	*The SW shall cyclically measure at least one analog input with two different ADCs with low delay, compare the results with appropriate tolerance and in case of deviation perform the specified reaction.*

Aus den guten und mittleren Anforderungen wurden im nächsten Schritt die Templates entwickelt.

Für das Darstellen der Variablen wurde während der Analyse für jeden Variablentyp eine andere Farbe gewählt. Dadurch erlangte man einen schnellen Überblick über die Vielzahl der verschiedenen Variablen. Außerdem wurde schnell erkenntlich wie aus dem Template die zugehörige Anforderung entsteht.

Beispiel:

„The [Component] shall {[adverb]} check if [Condition] and, {in case of violation trigger the appropriate reaction as specified in the fault database} {perform the following reactions:[List of Reactions]}“

4 Siehe Kapitel 4.1.4 Klassifizierung von Requirements

4.1.4 Klassifizierung von Requirements

Damit es eine gewisse Ordnung der Templates gibt, sodass Requirements Engineers nur in den entsprechenden Kategorien nach 3-7 Templates suchen müssen, anstatt aus einer Menge von 50 Templates das richtige zu finden, erfolgte eine Klassifizierung der Anforderungen.
Es folgt eine Tabelle, die die Arten von Requirements darstellt.
Durch die verschiedenen Darstellungsarten wird dem Requirements Engineer eine weitere Möglichkeit gegeben die entsprechende Anforderungsklasse zu finden, indem er sich Gedanken darüber macht, wie die zu entwickelnde Anforderung anders als in natürlicher Sprache dargestellt werden kann.
So können zum Beispiel Anforderungen, die Zustände beschreiben zusätzlich durch Zustandsdiagramme dargestellt werden.

Types of requirement	**Domain**	**Alternative types of presentation**	**Requirement wording Example**
Operation Modes	ECU wakeup/go-to-sleep behaviour, operation states of a continuous function	• Structured Text • Table • State diagram	• When the System is in state Standby and PowerOn button is pushed, the System shall enter state Active. • Anytime when the command GoToSleep is received via CAN, the ECU shall prepare PowerDown.
Continuous Function	Signal Processing, Closed-Loop Control of a motor	• Structured Text • Formel • Block Diagram (e.g. Simulink)	• The SW shall cyclically calculate the Converter Actual Power as a product of Converter Actual Current and Converter Actual Voltage. • The System shall limit the Output Current to 150A. • The Engine Controller shall control the Engine Speed in accordance with the Engine Speed Reference Signal.
Algorithm	Diagnostic, calibration or reconfiguration procedure Procedure to be performed at startup or change of operation states	• Structured Text • Activity diagram • Sequence diagram	• After PowerUp, the SW shall measure the Sensor Offset values for each of the Phase Current Sensors.
Feature		• Structured Text	• The ECU shall persistently store the last 10 Failure Codes. • The System shall provide a function for setting the Speed Reference manually.

Types of requirement	Domain	Alternative types of presentation	Requirement wording Example
Performance	Response time, Controller overshoot, measurement accuracy	• Structured Text • Data table • Timing Diagram • Screenshot Oszillosgraph	• The cycle time for the Current Regulator task shall be not more than 10ms. • The response time of the Calibration service shall be not more than 500ms. • The SW shall be capable of processing the measurement value of the LV current in the range from 0A to 254A. • The Current Controller shall have an accuracy of better than 1.5% at any point of the operation range. • The Power Stage shall have a Continuous Output Current of at least 200A.
Environmental Conditions	Operation Temperature Range Withstand capabilities against water or dust	• Structured Text • Table • Datasheet	• The HW shall be able to operate in a temperature range from -40°C to 200°C. • The SW shall be able to run on Windows XP, Windows Vista and Windows 7 operation systems.
Constraints	Weight, dimensions, color, surface	• Structured Text	• The weight of the ECU shall be not more than 150g. • The SW shall consume not more than 60% of the available RAM in normal RUN mode.
Compliance	Conformity to some standard e.g. EMC immunity, Process requirements	• Structured Text • Reference to documents	• The SW architecture shall be compliant with AUTOSAR standard. • The EMC immunity of the ECU shall be according to 2004/108/EG.
Interface	CAN Matrix, Measurement Ranges, Voltage Levels for Input Terminal, Program Language Variable Type Definition	• Structured Text • Table /Datasheet • Standard Data File Format (e.g. dbc) • Sequence Diagram (for Protocols) • Data Type Definitions (for SW Interfaces)	• The SW shall read the crank status from terminal 15. • The SW shall take the Operation Mode Command from the CAN signal EM_OPMODE. • The input range for the Temperature Sensor Connector shall be from 0V to 10V. • The Current Sensor Driver shall put its output value in a 32 bit signed integer variable scaled by 0.01A per LSB.

Types of requirement	Domain	Alternative types of presentation	Requirement wording Example
Structure		• Structured Text • Block diagram • Drawings	• The System shall consist of an ECU and an Energy Storage Device. • The Microprocessor shall be equipped with an External Watchdog. • The SW architecture shall provide a dedicated HW abstraction layer.
Exception Handling		• Structured Text	• If the function SQRT is called with a negative argument, the SW shall throw an InvalidValue exception. • If the ECU receives no further CAN messages within 10s after WakeUp, the ECU shall prepare PowerDown.
Diagnostics		• Structured Text • Table (e.g. Failure Detection / Reaction Matrix)	• The SW shall cyclically check the Temperature Sensor value against specified upper and lower limits and in case of violation request a Driver Warning “Red”. • The HW shall continuously compare the Motor Temperature against a limit of 120°C and if higher switch off the Power Stage.

Im nächsten Schritt erfolgte die Validierung der bisher entwickelten Templates.

4.1.5 Validierung der Templates

Somit entstanden im ersten Ansatz eine große Menge an Templates, die sich auf die verschiedenen Anforderungsklassen aufteilten. Diese entsprachen aber noch nicht dem gesetzten Ziel, denn es gab einige Klassen, die zu viele Templates, gegenüber der Zielsetzung eine kleine Menge von Grundtemplates zu erstellen, besaßen und andere Klassen, die nur wenige beinhalteten. Wenn es zu viele Templates waren, wurde versucht einige ähnliche Templates zusammenzulegen.

Beispiel:

Anfangs gab es diese zwei Templates:

1. *„If [Condition], the [component] shall{ [adverb]} {activate [mode] } „*
2. *„If [Condition], the [component] shall{ [adverb]} enter the state [state] {with the following settings: [list of [volatile value]= [nonvolatile value]]}.“*

Es ist leicht zu erkennen, dass sich diese Templates ähneln. Sie unterscheiden sich lediglich in dem Punkt was getan werden soll, wenn die bestimmte Bedingung eintritt. Somit werden beide Templates zu einem zusammengelegt und es entsteht ein neues, welches aber die Ausdrucksstärke beider vorangegangen Templates besitzt.

„If [Condition], the [component] shall{ [adverb]} {activate [mode] } {enter the state [state]} {with the following settings:[list of (volatile value)= (nonvolatile value)] }.“

Der andere Fall ist, dass es so wenige Templates in der Anforderungsklasse gibt, dass die Suche nach dem Template unwesentlich beeinflusst wird, wenn dieser Klasse einige Templates hinzugefügt werden. Dabei werden nicht unbedingt neue Templates gesucht, sondern die vorhandenen werden geteilt, sodass neue Anforderungen entstehen.

„The [property] shall be {less than}{bigger than} [nonvolatile value].“

Da in der Klasse Constraints nur wenige Templates Verwendung fanden, konnte die Komplexität des Templates verringert werden, indem man aus einem drei Templates macht, ohne dass die entstehende Menge der Templates die Übersicht beeinflussen würde.

1. „The [property] shall be less than [nonvolatile value]“

2. „The [property] shall be bigger than [nonvolatile value]“

3. „The [property] shall be [nonvolatile value].“

Anschließend erfolgt eine Beschreibung der Formalen Regeln, sodass aus den entwickelten Templates Anforderungen oder weitere abgeleitete Templates (Templatevarianten) entstehen.

4.1.6 Formale Regeln

Die Formalen Regeln repräsentieren die Inhalte der Variablen. Sie wurden in Backus-Naur-Form geschrieben, damit die verschiedenen Hierarchiestufen leichter sichtbar wurden.

4.1.6.1 Bedingungen

Bedingungen spielen beim Bilden von semantisch und syntaktisch korrekten Anforderungen eine wichtige Rolle. Sie drücken aus, wann ein Akteur eine gewisse Aktion machen soll. Weiterhin muss es eine Möglichkeit geben die Bedingungen beliebig oft mit einander zu verknüpfen mittels Und- oder Oder-verknüpfungen.

Bedingung::= <Komplexbedingung>|<Wert-Vergleich>|<Zustand-Vergleich>|<Modus-Vergleich> | <Event-Vergleich>| not <Bedingung>

Komplexbedingung::= <Bedingung> <Konjunktion> <Bedingung>

Konjunktion::= and | or

Wert-Vergleich::= Werttyp<Vergleichsoperator> Werttyp

Zustand-Vergleich::=[Component] is in [state]

Modus-Vergleich::=[Component] is in [mode]

Event-Vergleich::= [event] has occured

Vergleichsoperator::= größer | gleich | kleiner

Somit ist eine Bedingung entweder eine Erweiterung, ein Wert-Vergleich, ein Zustand-Vergleich, ein Modus-Vergleich, ein Event-Vergleich oder eine negierte Bedingung . Eine Komplexbedingung ist eine Erweiterung einer Bedingung zu mindestens zwei Bedingungen, die durch eine Konjunktion

verknüpft sind. Ein Wert-Vergleich besteht aus Werttypen[5], die durch einen Vergleichsoperator miteinander verglichen werden. Ein Zustand-Vergleich vergleicht ob eine Komponente in einem bestimmten Zustand ist und ein Modus-Vergleich vergleicht ob eine Komponente in einem Modus ist. Bei einem Event-Vergleich wird überprüft ob ein bestimmtes Event eingetreten ist, zum Beispiel ob ein Fehler aufgetreten ist oder ob eine Nachricht empfangen wurde.

4.1.6.2 Werttyp (Value)

Wert-Typ::= <nicht veränderlich>| <veränderlich>

nicht veränderlich::= Parameter | <Konstante>

Konstante::=Zahl Einheit

veränderlich::= <Signal> | Physikalische Größe

Signal::=<I/O- Signal> | Sensor-Signal | BUS-Signal

I/O::= Analog-Signal | Digital-Signal

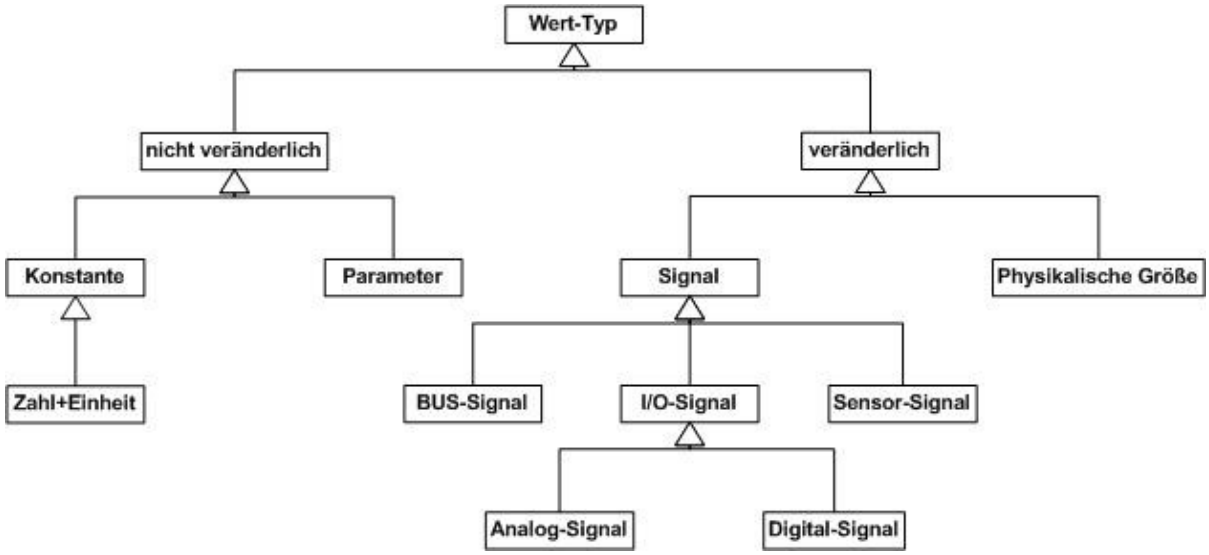

Abbildung 3: Struktur der Wert-Typen

Ein Wert-Typ (siehe Abbildung 3) ist somit ein veränderlicher oder nicht veränderlicher Wert. Ein nicht veränderlicher Wert ist dann wiederum ein Parameter oder aber eine Konstante. Eine Konstante ist darstellbar als eine Zahl und ihre entsprechende Einheit, welche aus einer gegebenen Liste ausgewählt wird. Bei einem veränderlichen Wert handelt es sich um Signale oder physikalische Größen. Bei den Signalen erfolgt eine Unterteilung von wem dieses Signal gesendet wurde. Dies kann entweder ein Sensor, ein BUS oder ein I/O Gerät gewesen sein. Das I/O Signal kann wiederum ein analoges oder digitales Signal sein. Bei binären Signal ist der Vergleich mit den Konstanten 0 oder 1 sowie true oder false möglich.

5 Nähere Erklärung erfolgt in Kapitel 4.1.6.2

4.1.6.3 Komponente (Component)

Component::=externalComponent | HardwareComponent | SoftwareComponent |SystemComponent

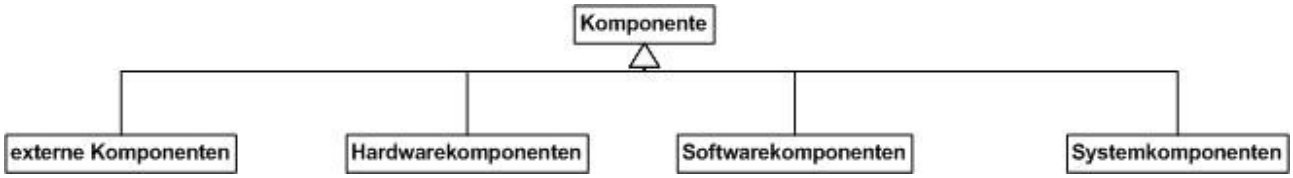

Abbildung 4: Struktur der Komponenten

Eine Komponente enthält die Unterklassen externe Komponenten, Hardwarekomponenten, Softwarekomponenten und Systemkomponenten.

4.1.6.4 Listen in Anforderungen

In einigen Anforderungen erscheinen Satzkonstruktionen der Form:

If Condition, the sw shall perform the following Reactions

- Reaction 1
- Reaction 2
- …
- Reaction n

Damit diese Form von Anforderungen durch die Methodik erfasst werden können, wurde die Variable [List of (Variable)] eingeführt. Dabei signalisiert das [List of …], dass an dieser Stelle eine Liste eingesetzt werden kann und (Variable) gibt an welche Typen die Liste repräsentieren. Die Variablen spiegeln dabei die einzelnen Satzbausteine wieder. Dadurch besteht die Möglichkeit an beliebiger Stelle innerhalb einer Anforderung eine Liste aufzubauen.

Somit entsteht zum Beispiel aus dem Template:

„The [Component] shall check if [Condition] and, in case of violation perform the following reactions: [List of (Reactions)]“

eine Anforderung der Form:

„The SW shall cyclically check the phase current measurement value against specified upper and lower limits and, in case of violation, perform the following reactions:

-force the Safe State regarding torque generation, as specified in the corresponding chapter
-communicate the failue on the CAN
-set an appropriate entry (DTC) in failure memory

Betrachtet man nun noch genauer den Fall der Liste von Reaktionen, stellt man fest, dass viele Kunden schon Fehlertabellen besitzen, weshalb die Nutzung der Variable „Reactions“ große Anwendung finden wird. Es müssen lediglich die Daten der Fehlerreaktionen in die Liste der Reactions übertragen werden.

4.1.6.5 Andere

Die übrigen Formalen Regeln werden hier nicht detaillierter vorgestellt.
Dazu gehören State, Mode, Adverb, Function und viele weitere.

4.1.7 Entscheidungen

Nach Befragungen von Requirements Engineers wurde die Entscheidung getroffen Templates wie:

„The range for [value] shall be from [nonvolatile value] to [nonvolatile value].“

in dieser Form bestehen zu lassen, anstatt zwei Templates zu entwickeln, die die einzelnen Grenzen beschreiben.

Diese hätten dann folgende Form:

„The upper range for [value] shall be [nonvolatile value].“

„The lower range for [value] shall be [nonvolatile value].“

Dies widerspricht zwar der Qualitätseigenschaft der Atomizität, dass jede Anforderung nur eine Funktion beschreiben soll, aber dafür trägt das Template in dieser Form maßgeblich zur Verständlichkeit bei. Da zusammengehörende Aspekte nicht mehr getrennt voneinander verfasst werden.

Die nächste Entscheidung betrifft die Universalität eines Templates. Anfangs gab es Templates der Form:

„The [component] shall be capable of processing [value] in the range from [nonvolatile value] to [nonvolatile Value].“

Später entstand zusätzlich das Template:

„The range for [value] shall be from [nonvolatile value] to [nonvolatile value].“

Da beide Formen für ähnliche, wenn auch nicht exakt gleiche Aussagen geeignet sind, wurde die erste Variante verworfen und die zweite Variante gewählt, da sie universeller ist. Man kann deutlich mehr Anforderungen aus ihr entwickeln. Dadurch scheinen zwar im ersten Augenblick Defizite in der Genauigkeit zu entstehen, da nicht klar ist zu welcher Komponente dieser Wert gehört. Dieses Defizit kann aber leicht umgangen werden, indem man zu den Werten noch die entsprechende Komponente addiert. Somit wird zum Beispiel aus „die Temperatur muss zwischen...“ „die Motortemperatur muss zwischen...“.

4.1.8 Anmerkungen

Bei der Erstellung von Anforderungen aus den Templates gilt die Annahme, dass im ersten Schritt Menschen diese Aufgabe vollführen. Dies ist notwendig, da die Möglichkeit besteht keine vollständigen Sätze zu erstellen, wenn zu viele optionale Teile gestrichen werden. Im weiteren Verlauf können jedoch automatisierte Prüfungen der Anforderungen vorgenommen werden.

Auch die Entscheidung im Zweifel die variablen Teile allgemeiner zu lassen, anstelle von mehr einzuschränken, fiel unter der Annahme, dass sich ein Mensch der Aufgabe annimmt, da auch hier das Problem besteht, dass unlogische Satzkonstruktionen entstehen könnten.

4.1.9 Die Listen

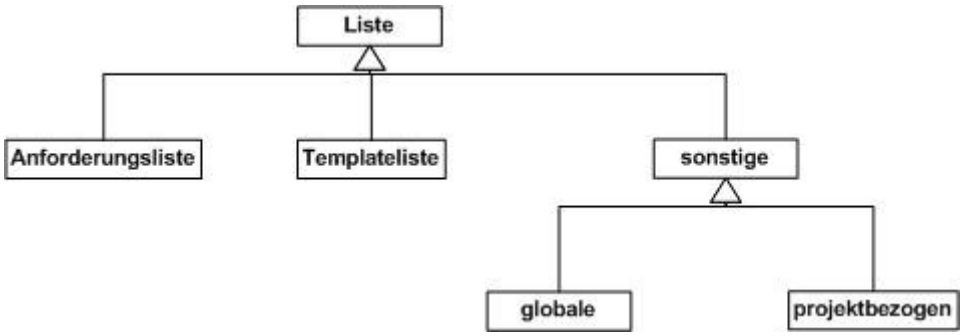

Abbildung 5: Struktur der Listen

Abbildung 5 zeigt die Struktur von Listen. Somit existieren drei verschiedene Arten von Listen, die da wären Anforderungslisten, Templatelisten und die restlichen Listen.

Die Anforderungslisten spiegeln die Anforderungen an das Projekt oder Teile des Projektes wieder. Sie sollen projektbezogen sein, das heißt ein Projekt kann mehrere Anforderungslisten besitzen, es sollte aber nicht vorkommen, dass eine Anforderungsliste mehrere Projekte beschreibt.

Die Templatelisten enthalten alle Templates, die für die Projekte benötigt werden. Dabei wird anfänglich eine globale Liste von Templates bereitgestellt, welche aber individuell durch Ableitungen von vorhandenen Templates oder durch das hinzufügen neuer Templates erweitert werden kann.

Die sonstigen Listen enthalten die Elemente, die letztendlich für die Variablen durch Verwendung der formalen Regeln in den Templates eingesetzt werden um fertige Anforderungen zu erhalten.

5 Use cases

5.1 Use case Gruppen

Nach Recherchen und Gesprächen mit Requirements Engineers ergab sich die Möglichkeit die Anforderungen in drei use case-Gruppen einzuteilen.

Wenn im folgenden Kapitel das Wort Listen erwähnt wird, sind damit immer die sonstigen Listen gemeint.

Weiterhin entstand die Idee, dass die Definition verschiedener Rollen für ein zugehöriges Tool ebenfalls notwendig wäre.

Somit finden im folgenden Kapitel die Rollen Requirements Engineer, Projektadministrator und Globaler Administrator Verwendung. Dabei nehmen die Rechte der Reihe nach zu.

Die erste Gruppe behandelt die Anforderungslisten (siehe Abbildung 6). Sie sagt aus, dass ein Requirements Engineer die Möglichkeit hat eine neue Anforderungsliste zu erstellen oder eine aktuelle zu bearbeiten. Dies beinhaltet ebenfalls die use cases, ein Requirement zu löschen, ein neues hinzuzufügen, eines zu ändern und die gesamte Struktur der Anforderungsliste zu verändern.

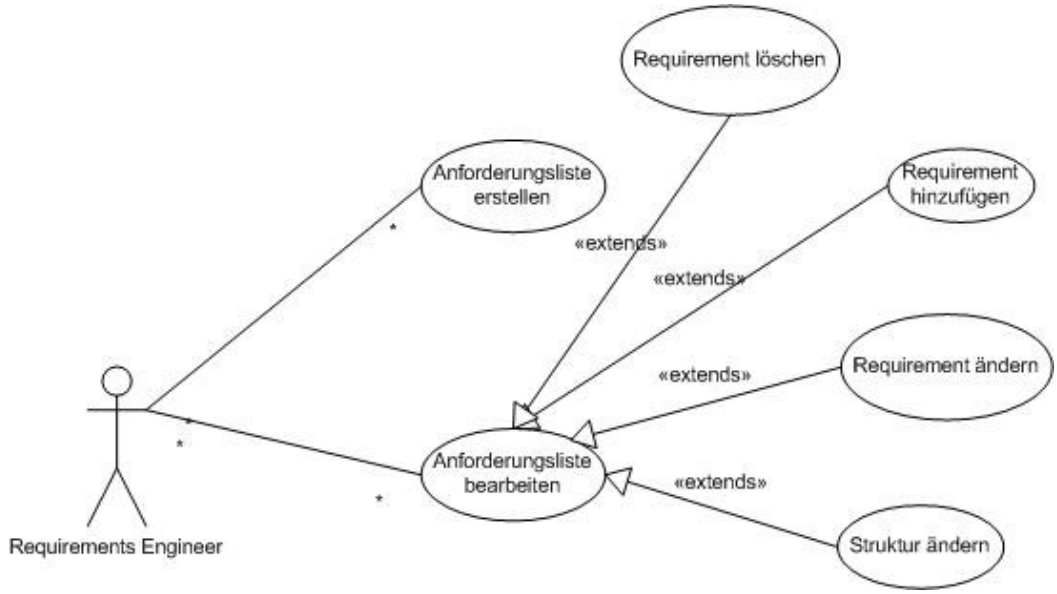

Abbildung 6: Use case Gruppe1: Anforderungsliste

Die zweite Gruppe beschreibt die Funktionen eines globalen Administrators (siehe Abbildung 7). Er hat zusätzlich die Möglichkeit ein neues Template zu erzeugen, ein bereits vorhandenes zu ändern, ein Template zu löschen oder aber auch globale Listen zu bearbeiten und auch zu erstellen. Weiterhin hat er ebenfalls die Rechte eines Requirements Engineers und eines Projektadministrators.

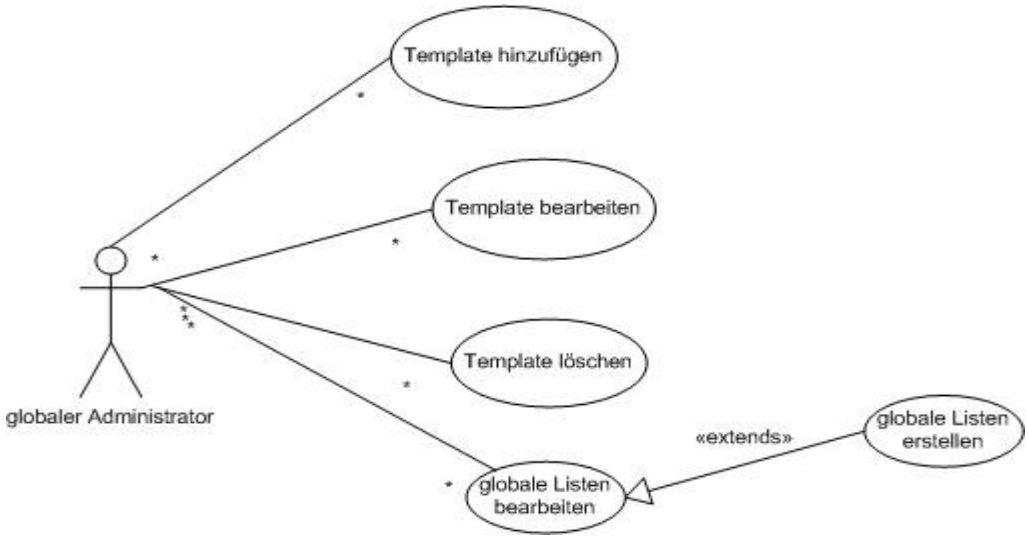

Abbildung 7: Use case Gruppe2: Templates

Die dritte und letzte Gruppe betrifft den Projektadministrator (siehe Abbildung 8). Er hat zusätzlich die Befugnisse Listen zu importieren. Diese Listen können zum Beispiel Parameterlisten, Signallisten, Komponentenlisten und Fehlerreaktionslisten sein. Weiterhin kann er eine vorhandene Liste bearbeiten was auch das erstellen sowie editieren von Listen beinhaltet. Die Listen sind jedoch nur projektbezogen. Weiterhin besitzt er die Rechte eines Requirements Engineers.

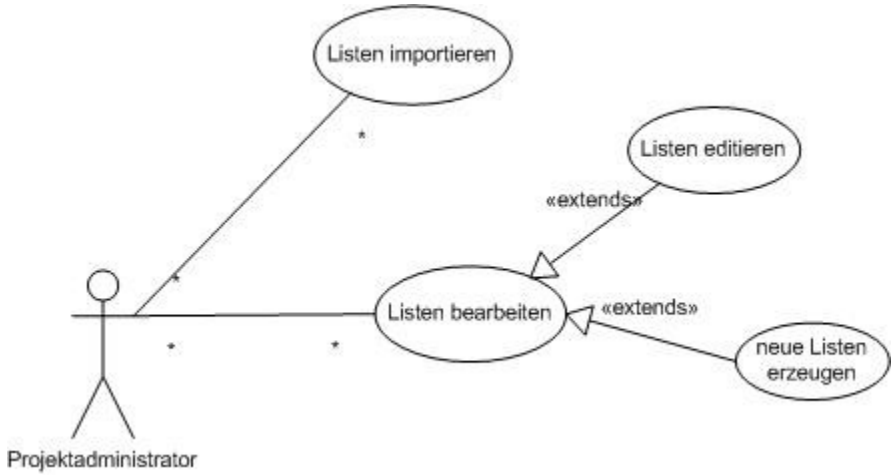

Abbildung 8: Use case Gruppe3: Listen

Es folgt eine nähere Betrachtung der einzelnen use cases.

5.2.3 Requirements Engineer

Abbildung 9 zeigt den Vorgang, wie ein Requirements Engineer eine neue Anforderungsliste erstellt. Es beginnt mit einer leeren Anforderungsliste, die durch den Benutzer bearbeitet wird und letztendlich durch einen Datei Speichern Dialog lokal gespeichert wird, um sie für spätere Zwecke verwendbar zu machen.Es besteht auch die Möglichkeit eine leere Liste erst einmal abzuspeichern. In beiden Fällen wurde am Ende eine neue Anforderungsliste erstellt.

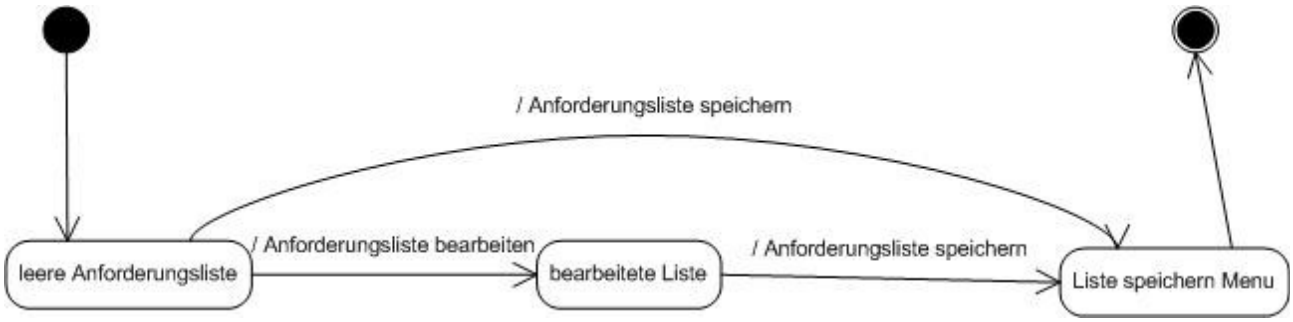

Abbildung 9: Use case 1.1 Anforderungsliste erstellen

Use case 1.2 (Abbildung 10) beschreibt die Vorgänge um eine Anforderungsliste zu bearbeiten.

Dafür ist es notwendig eine Liste geöffnet zu haben. Dies entsteht entweder durch das Laden einer Liste oder durch das Erstellen einer neuen Liste.

Wenn die Liste geöffnet ist gibt es mehrere Möglichkeiten diese zu bearbeiten.

Eine Möglichkeit ist, die Struktur der Liste zu verändern.

Eine weitere Möglichkeit ist das Ändern von vorhandenen Anforderungen. Weiterhin kann man auch vorhandene Anforderungen löschen oder gänzlich neue erstellen.

Eine weitere Möglichkeit die vorhandene Liste zu bearbeiten ist Anforderungen, die nicht mehr benötigt werden oder einfach falsch sind aus dieser Liste zu entfernen.

Letztendlich erhält man eine neue bearbeitete Anforderungsliste, die man abspeichern kann.

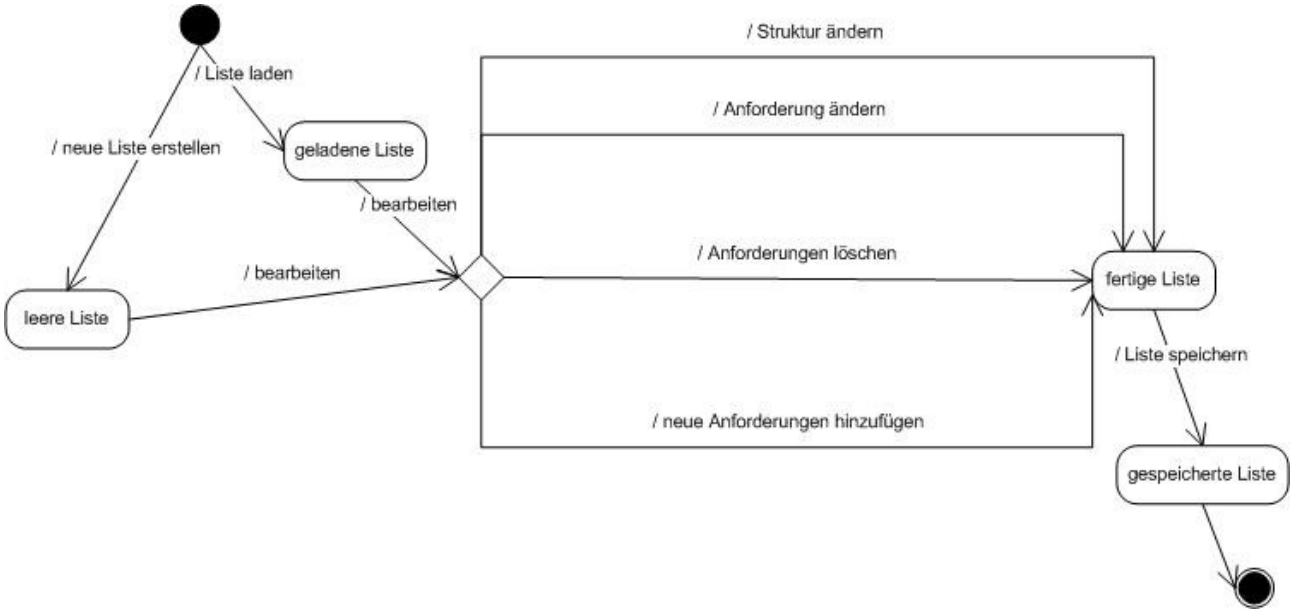

Abbildung 10: Use case 1.2 Anforderungsliste bearbeiten

Abbildung 11 beschreibt den Vorgang der Strukturänderung einer Anforderungsliste. Wenn eine Anforderungsliste geöffnet ist gibt es dafür mehrere Möglichkeiten.

Erstens man fügt neue Überschriften ein. Dies beinhaltet entweder Unterüberschriften für schon vorhandene Überschriften oder neue Überschriften. Die zweite Möglichkeit beschreibt das Entfernen von gesamten Blöcken aus der Anforderungsliste und drittens das verschieben von Blöcken an andere Positionen in der Anforderungsliste. Ein Block ist dabei eine Menge von aufeinanderfolgenden Anforderungen. In jedem Fall erhält man eine Anforderungsliste mit veränderter Struktur.

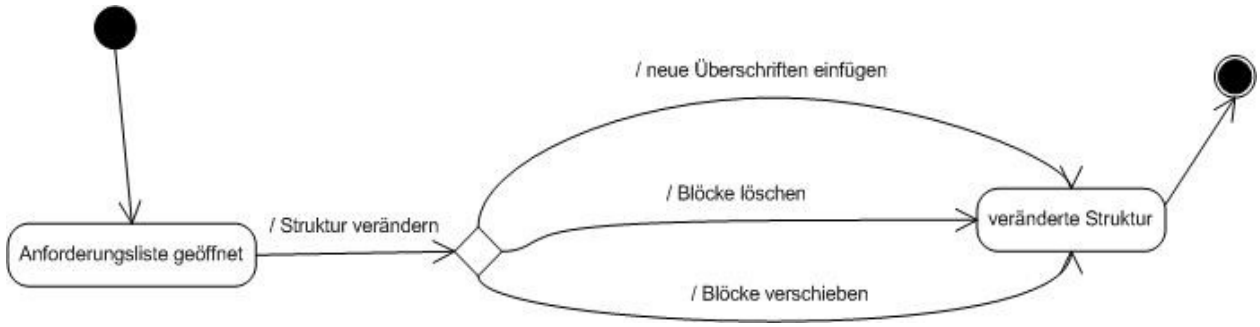

Abbildung 11: Use case 1.3 Struktur ändern

Abbildung 12 stellt dar, welche Schritte vollzogen werden müssen um eine Anforderung zu ändern. Die Anforderung muss erst einmal ausgewählt werden und das Ändern dieser durch Drücken eines Knopfes oder einer Taste bestätigt werden. Daraufhin öffnet sich der Anforderungseditor und der User kann eine neue Anforderung erstellen, die die vorhandene editiert.

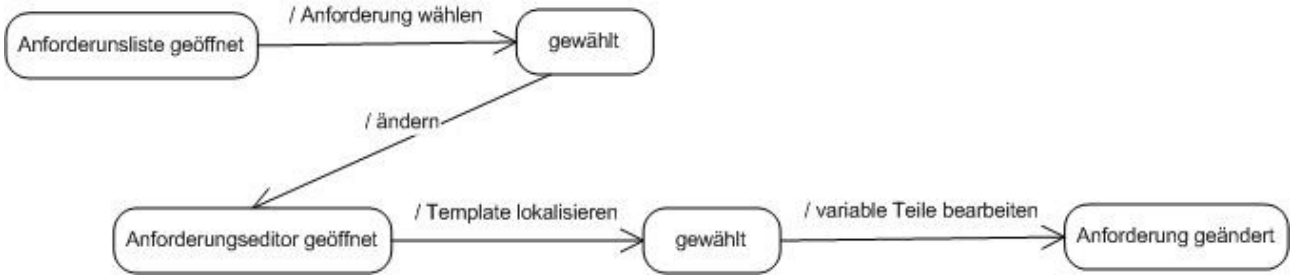

Abbildung 12: Use case 1.4 Anforderung ändern

Wenn eine Anforderung gelöscht werden muss (siehe Abbildung 13), muss diese vorher in irgendeiner Weise erst einmal ausgewählt werden. Durch Drücken eines Knopfes oder einer Taste kann dann die Anforderung gelöscht werden.

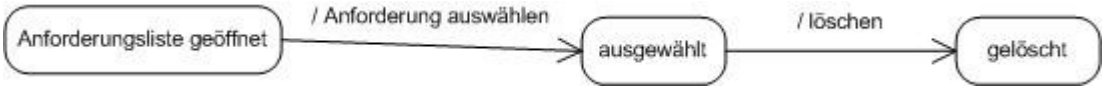

Abbildung 13: Use case 1.5 Anforderung löschen

Um neue Anforderungen hinzuzufügen (siehe Abbildung 14) wird der entsprechende Editor geöffnet. In dem Editor muss der Benutzer sein gewünschtes Template lokalisieren und fügt dieses dann zu dem Editor hinzu. Das Template wird dann bearbeitet, dies bedeutet der User wählt die optionalen Teile aus und befüllt die Variablen mit Werte bis die Anforderung fertig ist. Mit dem Beenden des Editors, wird die Anforderung zur Liste hinzugefügt. Den Anforderungen soll

zusätzlich eine ID gegeben werden, zwecks Identifizierbarkeit und Rückverfolgbarkeit.

Abbildung 14: Use case 1.6 Anforderung hinzufügen

5.2.4 globaler Administrator

Use case 2.1 (siehe Abbildung 15) zeigt, wie ein globaler Administrator ein neues Template hinzufügt. Dafür muss der Templateeditor geöffnet werden und der globale Administrator muss sich entscheiden, ob er ein komplett neues erstellen will oder aus einem vorhandenen Template ableiten will.

Wenn er aus einem vorhanden Template ableiten will, muss er das gewünschte Template erst einmal lokalisieren und auswählen. Dieses wird dann in den Editor eingefügt und man kann die entsprechenden Teile ändern.

Die Möglichkeit aus einem Template weitere abzuleiten steht zusätzlich auch dem Requirements Engineer zur Verfügung. Komplett neue Templates soll er aber nicht erstellen können.

Falls er nicht ableiten will, öffnet sich gleich der leerer Editor und er kann ein Template in der Template-Metasprache eingeben. Durch Bestätigen wird das Template der Templateliste hinzugefügt.

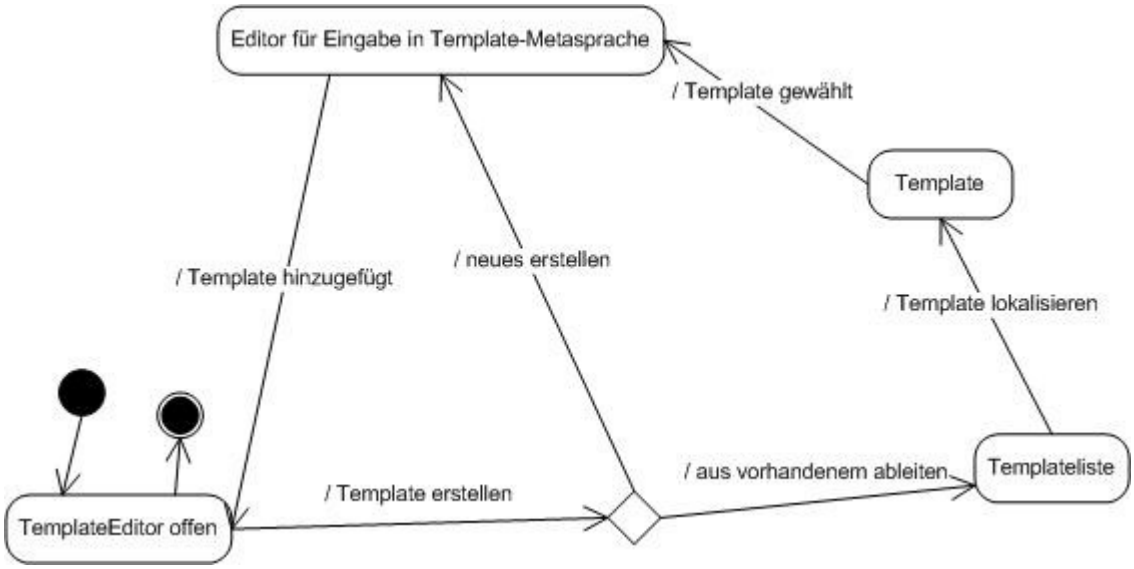

Abbildung 15: Use case 2.1 Template hinzufügen

Abbildung 16 zeigt wie vorhandene Templates bearbeitet werden. Dafür lokalisiert man das Template, welches man ändern möchte im Templateeditor. Anschließend öffnet sich der Editor zum Ändern. Nach dem Ändern werden je nach dem, ob von diesem Template abgeleitet wurde, die abgeleiteten Templates angezeigt. Diese können ebenfalls bearbeitet werden.

Letztendlich werden die gesamten Änderungen übernommen.

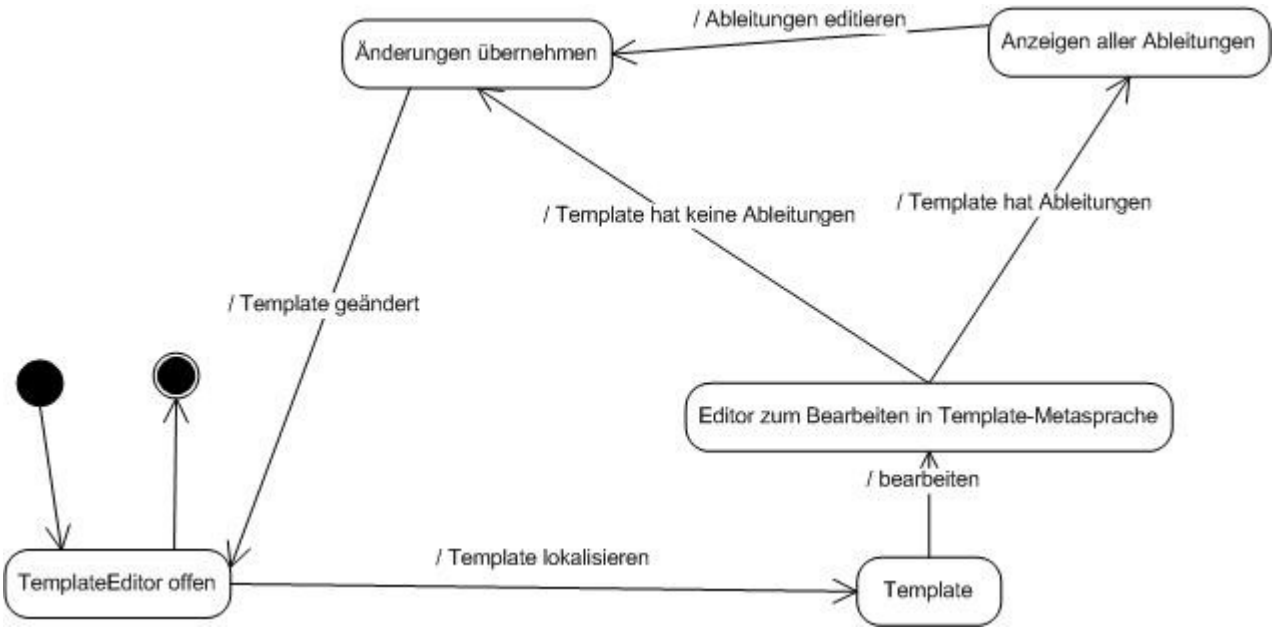

Abbildung 16: Use case 2.2 Template bearbeiten

Use case 2.3 beschreibt, wie existierende Templates entfernt werden (siehe Abbildung 17).

Dafür muss das entsprechende Template lokalisiert werden. Danach erfolgt eine Aktion abhängig davon, ob von diesem Template andere Templates abgeleitet wurden oder nicht.

Wenn von dem Template abgeleitet wurde, werden einem alle abgeleiteten Templates angezeigt, so dass diese eventuell auch gelöscht werden können.

Wenn es keine abgeleiteten Templates gibt wird das ausgewählte lediglich gelöscht.

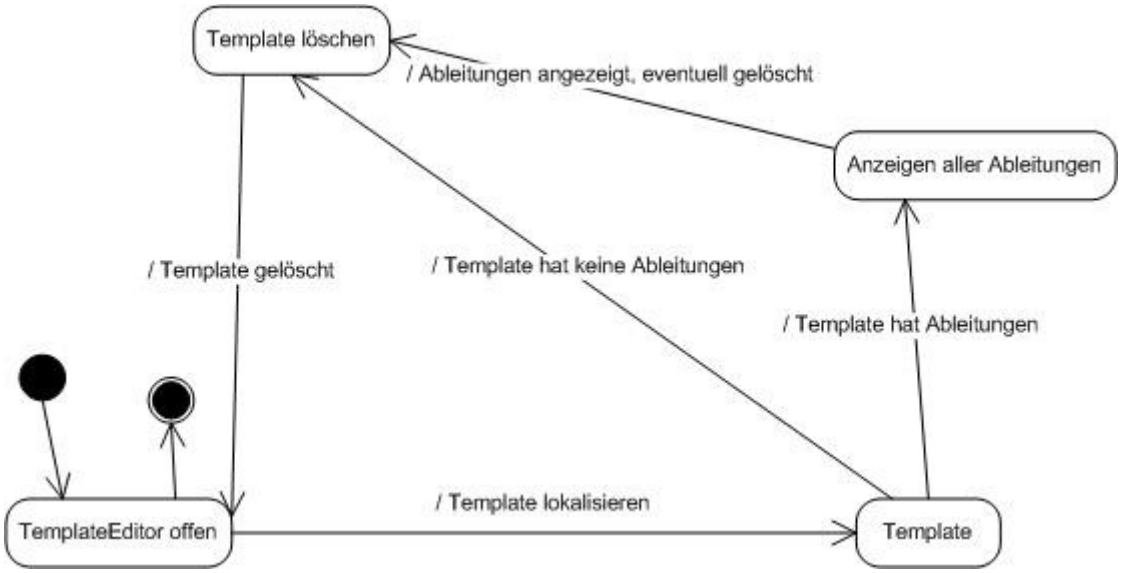

Abbildung 17: Use case 2.3 Template entfernen

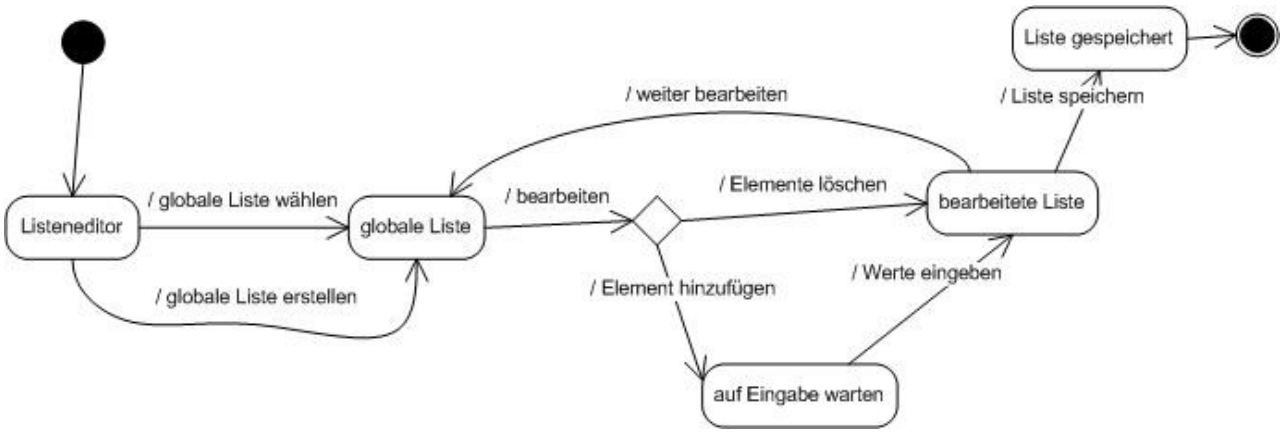

Abbildung 18: Use case 2.4 globale Listen bearbeiten

Abbildung 18 zeigt, wie der globale Administrator globale Listen bearbeitet. Dafür muss entweder eine vorhandene globale Liste gewählt werden oder eine neue globale Liste erstellt werden. Danach kann diese bearbeitet werden, was bedeutet es werden Elemente hinzugefügt oder entfernt. Beim Hinzufügen von Elementen müssen natürlich noch die notwendigen Werte eingegeben werden. Wenn man fertig ist wird die Liste gespeichert

Damit der globale Administrator eine neue globale Liste erstellen kann muss er sich in dem entsprechenden Editor befinden und dort eine neue Liste erstellen. Er erhält dann eine leere Liste, welche er abspeichern kann (siehe Abbildung 19).

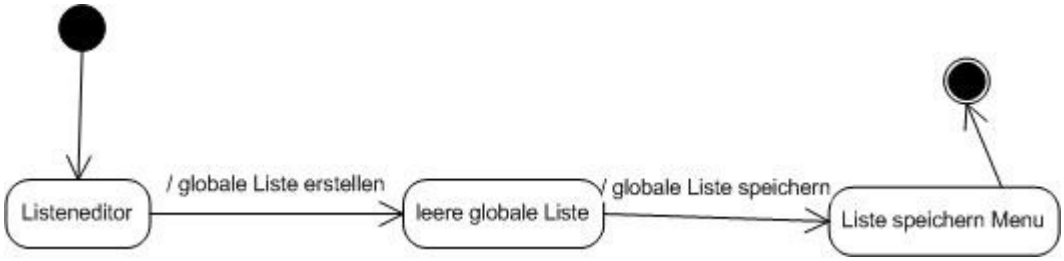

Abbildung 19: Use case 2.5 globale Liste erstellen

5.2.5 Projektadministrator

Abbildung 20 beschreibt, wie das Importieren von Listen funktioniert. Dafür muss sich der Projektadministrator im Listeneditor befinden. Dann kann er über ein Datei öffnen Dialog die entsprechende Liste auswählen, so dass diese Parameter zu der momentanen ausgewählten Liste hinzugefügt werden. Diese Liste wird dann wieder lokal gespeichert.

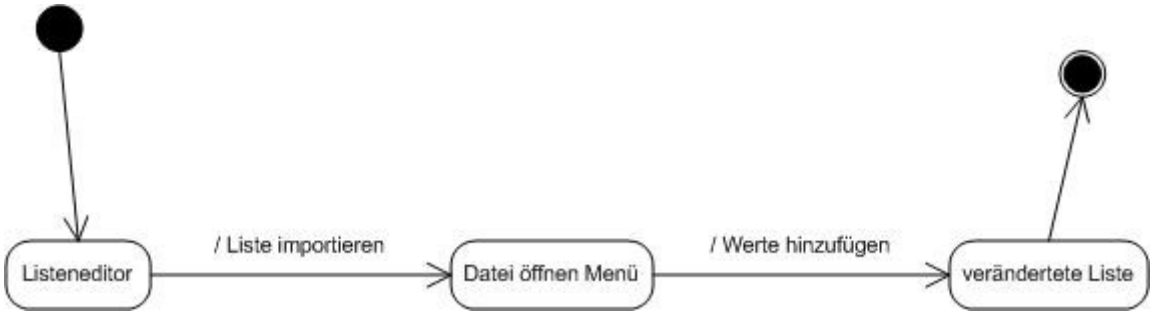

Abbildung 20: Use case 3.1 Liste importieren

Abbildung 21 zeigt wie Projektlisten bearbeitet werden. Dafür muss die gewünschte Liste entweder geöffnet oder neu erstellt werden. Nun besteht wieder die Möglichkeit Elemente zu entfernen oder neue Elemente hinzuzufügen, indem man die notwendigen Werte eingibt. Dies beinhaltet ebenfalls die use cases Liste editieren und neue Liste erstellen.

Diese Liste kann dann abgespeichert werden.

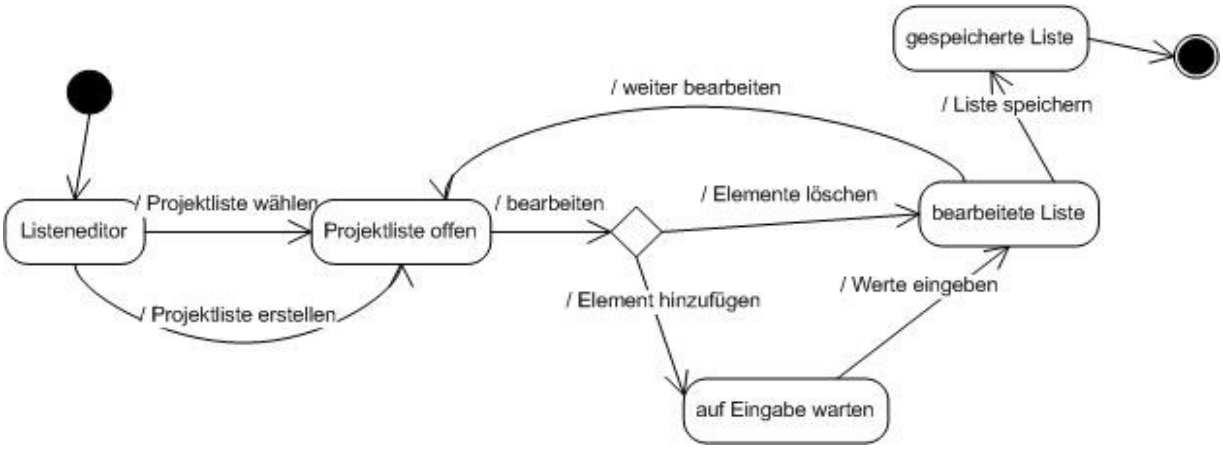

Abbildung 21: Use case 3.3 Projektliste bearbeiten

Der letzte use case beschreibt, wie Projektlisten überhaupt erst einmal erstellt werden. Dafür muss der Projektadministrator im Listeneditor eine neue Liste erzeugen und diese schließlich abspeichern (siehe Abbildung 22).

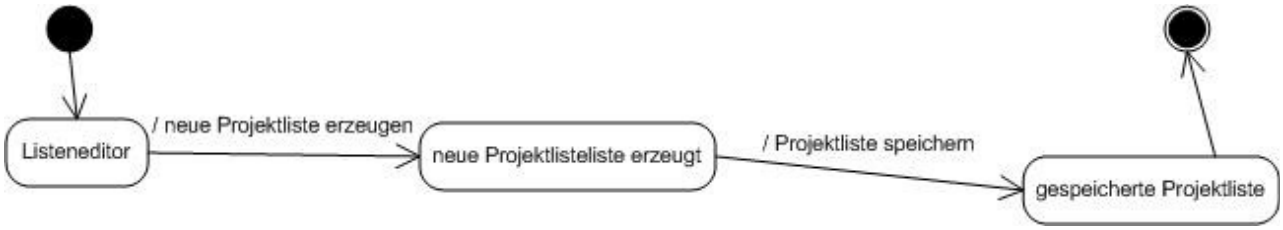

Abbildung 22: Use case 3.4 Projektliste erstellen

6.Prototypische Implementierung zur Evaluierung der Methodik

Der gesamte Prototyp wurde mit Java entwickelt. Dazu wurde Eclipse[6] zusammen mit dem visual Editor Plugin[7] verwendet. Das Plugin ermöglicht ein Zusammenbauen der Oberfläche und stellt die momentane Oberfläche parallel zum Programmieren dar. Dadurch wurde das Entwickeln des User Interfaces erheblich erleichtert, da nicht mehr alles manuell durch Code geschrieben wurde.

6.1 Anforderungen an den Prototypen

Nach dem Entwickeln der use cases sind folgende Anforderungen an den zu entwickelnden Prototyp entstanden.

6.1.1 Generelle Softwareanforderungen

REQ:1. der Prototyp muss aus folgenden Softwarekomponenten bestehen:

- Anforderungsliste
- Templateeditor
- Listeneditor
- Anforderungseditor

REQ:2. Der Prototyp muss eine Userverwaltung ermöglichen.
REQ:3. Der Prototyp muss dabei zwischen drei verschiedenen Usern unterscheiden können:

- Requirements Engineer
- Projektadministrator
- globaler Administrator

6.1.2 Anforderungsliste

REQ:4. Der Prototyp muss die Möglichkeit bieten neue Anforderungslisten zu erstellen.
REQ:5. Der Prototyp muss die Möglichkeit bieten Anforderungslisten zu bearbeiten.

REQ:6. Wenn keine Anforderungsliste vorhanden ist, muss der Requirements Engineer eine leere neue Liste erstellen können.

REQ:7. Wenn eine Anforderungsliste vorliegt, soll der Requirements Engineer diese bearbeiten können.
REQ:8. Der Requirements Engineer muss neue Anforderungen hinzufügen können.
REQ:9. Der Requirements Engineer muss Anforderungen löschen können.
REQ:10. Der Requirements Engineer muss die Struktur der Anforderungsliste verändern können.

REQ:11. Der Requirements Engineer muss neue Überschriften hinzufügen können.
REQ:12. Der Requirements Engineer sollte Blöcke der Anforderungsliste löschen können.
REQ:13. Der Requirements Engineer sollte Blöcke der Anforderungsliste verschieben können.

REQ:14. Der Requirements Engineer muss Anforderungslisten vom Dateisystem laden können.
REQ:15. Der Requirements Engineer muss Anforderungslisten speichern können.

6 http://www.eclipse.org/
7 http://www.eclipse.org/vep/

6.1.3 Templateeditor

REQ:16. Der globale Administrator muss neue Templates zu der Templateliste hinzufügen können.
REQ:17. Der globale Administrator muss vorhandene Templates bearbeiten können.
REQ:18. Der globale Administrator muss vorhandene Templates löschen können.
REQ:19. Der globale Administrator muss globale Listen bearbeiten können.

REQ:20. Der Templateeditor muss eine Möglichkeiten bieten vorhandene Templates schnell zu finden und anzuzeigen.

REQ:21. Der globale Administrator sollte neue Templates von vorhandenen Templates ableiten können.
REQ:22. Ableitungen sollten in den Templatelisten gespeichert werden können.

REQ:23. Der globale Administrator muss im Editor für die Template-Metasprache neue Templates eingeben können.
REQ:24. Der Editor für Template-Metasprache sollte die eingegebenen Templates validieren.
REQ:25. Der Editor für Template-Metasprache sollte die abgeleiteten Templates bzw. das Template von dem abgeleitet wurde anzeigen.

6.1.4 Listeneditor

REQ:26. Der Projektadministrator muss Parameterlisten importieren können.
REQ:27. Der Projektadministrator muss Signallisten importieren können.
REQ:28. Der Projektadministrator muss vorhandene Listen bearbeiten können.

REQ:29. Wenn eine Liste geöffnet ist, muss der Projektadministrator diese bearbeiten können.
REQ:30. Der Projektadministrator muss manuell neue Elemente hinzufügen können.
REQ:31. Der Projektadministrator muss Elemente aus der Liste entfernen können.

REQ:32. Der Projektadministrator muss eine existierende Liste laden können.

REQ:33. Der Projektadministrator muss eine Liste speichern können.

6.1.5 Anforderungseditor

REQ:34. Der Requirements Engineer muss eine Anforderung erstellen können.

REQ:35. Der Requirements Engineer muss Templates lokalisieren können.
REQ:36. Der Requirements Engineer muss Templates auswählen können.
REQ:37. Der Requirements Engineer muss das Template zur Anforderung vervollständigen können.

REQ:38. Der Requirements Engineer muss die erstellten Anforderungen zur Anforderungsliste hinzufügen können.

6.2 Umsetzung der use cases

In diesem Kapitel werden die use cases aus Kapitel 4.2 mit Hilfe von Screenshots beschrieben. Dabei finden nur die Erwähnung, die letztendlich implementiert wurden. Alle anderen wären durch einen deutlichen Mehraufwand zwar implementierbar gewesen, jedoch stellen sie keinen notwendigen Aspekt zur Darstellung der Methode dar.

6.2.1 Requirement Engineer

6.2.1.1 Anforderungsliste erstellen

Für das erstellen einer neuen Anforderungsliste muss das Tool gestartet werden, sodass eine neue, leere Anforderungsliste vorliegt.

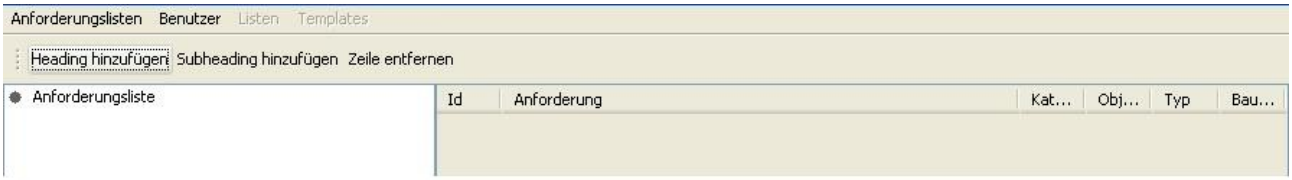

Abbildung 23: Ansicht einer leeren Anforderungsliste

Diese kann nun bearbeitet werden[8]. Anschließend kann die fertige Liste durch das Klicken auf “Anforderungslisten” und danach auf den Unterpunkt “Datei speichern” gespeichert werden.

Abbildung 24: Navigation um eine Anforderungsliste zu speichern

Danach öffnet sich der gewohnte “Datei speichern Dialog”, wo man seinen Speicherort, sowie den Namen der Datei angibt. Somit wurde eine neue Anforderungsliste erstellt und gespeichert.

6.2.1.2 Anforderungsliste bearbeiten

Damit eine Anforderungsliste bearbeitet werden kann muss erst einmal eine vorhanden sein. Dies geschieht entweder beim Starten des Tools, denn dann liegt eine neue, leere Anforderungsliste vor oder aber durch das Laden einer Anforderungsliste. Um eine Anforderungsliste zu laden muss auf die Schaltfläche “Anforderungsliste” gedrückt werden und danach auf den Unterpunkt “Datei öffnen”.

8 Siehe use case 5.2.1.1 Anforderungsliste bearbeiten

Abbildung 25: Navigation um eine Anforderungsliste zu laden

Dadurch öffnet sich der übliche "Datei öffnen Dialog" und man kann die gewünschte Datei laden.

Für das Bearbeiten der Anforderungsliste gibt es nun verschiedene Möglichkeiten. Dazu gehören das Verändern der Struktur der Anforderungsliste, das Hinzufügen von Anforderungen zu der Liste und das entfernen von Anforderungen von der Liste. Diese drei werden in den folgenden Abschnitten exakter beschrieben. Der use case eine Anforderung zu ändern wurde nicht implementiert, da dieses durch das Löschen und Hinzufügen realisiert werden kann.

6.1.1.3 Struktur ändern

Das ändern der Struktur beinhaltet das Hinzufügen von neuen Überschriften und Unterüberschriften. Neue Überschriften werden durch das Klicken des Buttons "Heading hinzufügen" realisiert. Nun muss noch der gewünschte Text der Überschrift eingegeben werden. Die Schriftgröße der Überschrift wird je nach Hierarchiestufe angepasst. Der Explorer im linken Bereich fängt ebenfalls an sich mit aufzubauen.

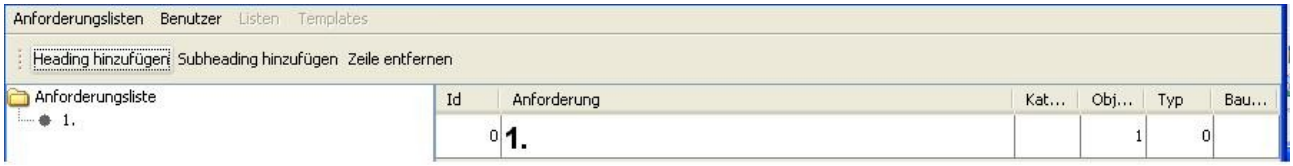

Abbildung 26: Anforderungsliste mit Hauptüberschriften

Damit jetzt unter Punkt 1. Unterüberschrift 1.1 entsteht muss zunächst die Überschrift zu der eine neue Unterüberschrift hinzugefügt werden soll markiert werden. Anschließend erfolgt ein Klick auf den Button "Subheading hinzufügen" und die Unterüberschrift 1.1 erscheint und benötigt noch einen eingegebenen Text.

Abbildung 27: Anforderungsliste mit Haupt- und Unterüberschriften

Das Löschen einer Überschrift oder Unterüberschrift verändert ebenfalls die Struktur der Anforderungsliste. Um eine Überschrift zu löschen muss diese vorher selektiert werden und danach durch Klicken des Buttons "Zeile entfernen" gelöscht werden.

Abbildung 28: Darstellung der Anforderungsliste nach dem Entfernen einer Zeile

Das Verschieben von Blöcken um die Struktur zu ändern ist zwar möglich, nur wird die Indexierung noch nicht mit angepasst.

6.2.1.3 Anforderung hinzufügen

Als Vorbedingung für den use case muss gelten, dass die entsprechenden Listen bereits mit Werten gefüllt wurden, damit die Variablen überhaupt ersetzt werden können.

Schritt 1: Wählen der Anforderungsklasse

Der Requirement Engineer öffnet den Anforderungseditor durch Klicken der Schaltfläche “Anforderungslisten” und dort den Unterpunkt “Anforderungen hinzufügen”.

Als nächstes muss gewählt werden zu welcher Kategorie die entstehende Anforderung gehören soll. Dabei sollen die Beispiele als kleine Hilfe dienen, wenn man sich nicht ganz sicher ist, wo die Anforderung nun einzuordnen ist..

Die Kategorien sind als Buttons implementiert und durch das Klicken der entsprechenden Kategorie gelangt man zum nächsten Schritt.

Abbildung 29: Darstellung der Kategorie von Anforderungen innerhalb des Prototypen

Schritt 2: Wähle das Template.

Der Requirements Engineer wählt das Template entsprechend der Anforderung aus. Dabei hat er die Möglichkeit ein Basistemplate zu verwenden oder aber eines seiner Eigenen. Diese müssen vorher gespeichert worden sein.

Durch das Klicken des Templates gelangt man zum nächsten Schritt. Durch Klicken des Buttons „Previous“ zum vorherigen Schritt.

Weiterhin wird für einen solchen Fall der Bauplan abgespeichert. Dadurch wird es ermöglicht, durch eine Edit Funktion, die nicht vollendete Anforderung an der Stelle fortzusetzen, wo der Vorgang unterbrochen wurde.

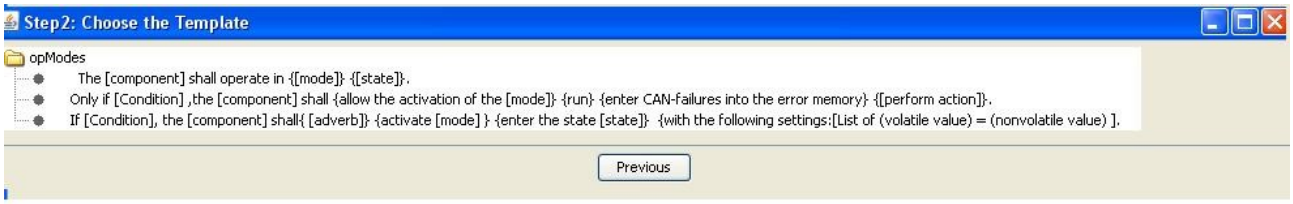

Abbildung 30: Darstellung der Templates in der gewählten Kategorie

Schritt 3a: Anpassen des Templates

In diesem Schritt muss der Requirements Engineer sein Template anpassen. Das heißt er wählt die optionalen Teile aus, die enthalten sein sollen, sowie Einschränkungen oder Erweiterungen für die Variablen. Zusätzlich erfolgt dazu das Befüllen der Variablen. Zu den Buttons kommt zusätzlich ein Button Save Template hinzu, welcher ermöglicht das Template, so wie es momentan ist, als selbst erstelltes Template unter dem gewählten Basistemplate abzuspeichern. Weiterhin wird noch ein Button “Save Requirement” eingefügt. Dieser speichert die Anforderung in jetzigen Zustand ab und beendet den Editor. Gibt es noch Variablen, die keine Werte enthalten wird dies in der Anforderung explizit mit einem “to do” gekennzeichnet. So wird zum Beispiel aus <value> → <value to do>. Somit weiß ein anderer Requirements Engineer, dass diese Anforderung noch nicht vollständig ist.

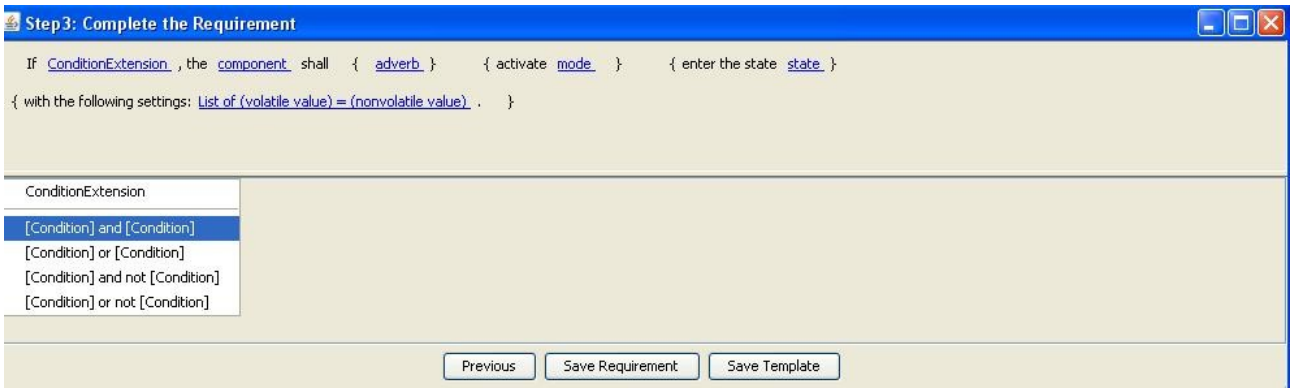

Abbildung 31: Darstellung der Erweiterung von Teilen zum Beispiel Conditions

Schritt 3b: Befüllen der Variablen durch Typenauswahl

Der Requirements Engineer muss nun die Variablen anklicken und befüllen.

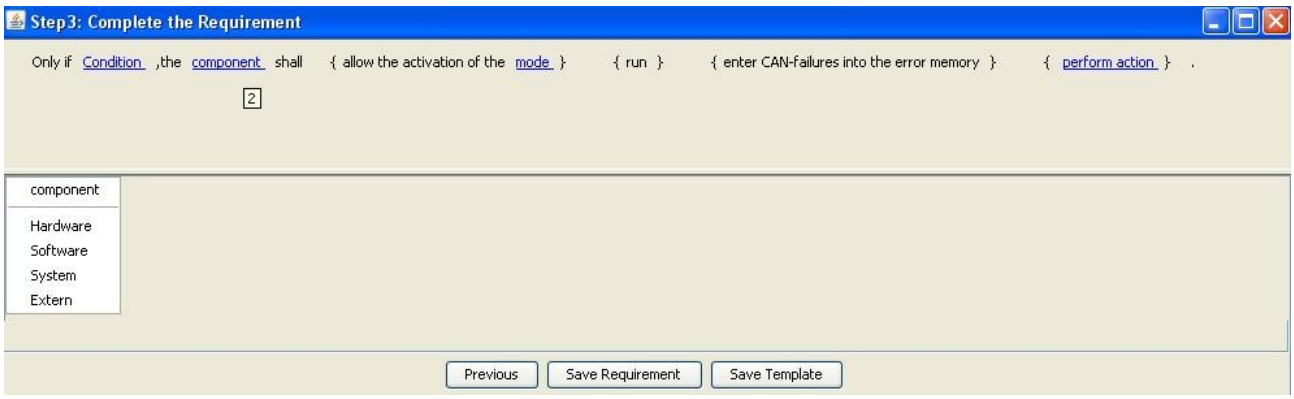

Abbildung 32: Darstellung der Befüllung von Variablen

Schritt 3c: Befüllen der Variablen durch Zahlenwerte

Es besteht die Möglichkeit nicht nur vordefinierte Werte hinzuzufügen, sondern auch einzelne Zahlenwerte mit Einheiten. Dafür wird in dem Textfeld der Zahlenwert eingegeben und die gewünsche Einheit selektiert. Zum Übernehmen muss dies mit “OK” bestätigt werden. Wurde keine Einheit ausgewählt wird nur der Zahlenwert eingefügt.

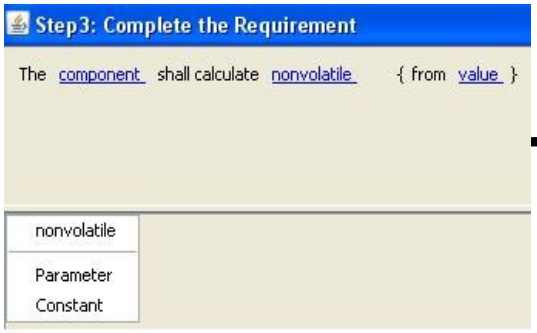

Abbildung 33: Darstellung der Werte beim Anklicken der Variablen "nonvolatile"

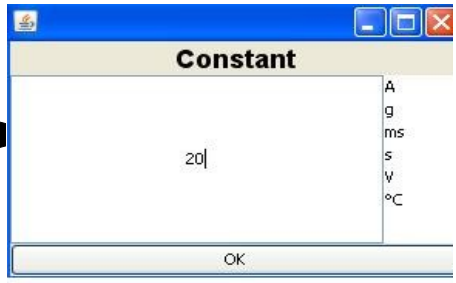

Abbildung 34: Editor zum Hinzufügen von Konstanten

6.2.2 Globaler Administrator

6.2.2.1 Template hinzufügen

Damit ein neues Template hinzugefügt werden kann, muss der Templateeditor geöffnet werden. Dazu muss in der Menüleiste der Punkt Templates und dort der Unterpunkt Templateeditor angeklickt werden.

Abbildung 35: Darstellung zum Starten des Templateeditors

Es öffnet sich ein Fenster, welches die Struktur der Templates als Explorer darstellt.

Möchte man nun ein neues Template hinzufügen, muss erst der Ordner oder das Template markiert sein, unter welchem das neue Template eingefügt werden soll. Anschließend kann das gewünschte Template unter Einhaltung der Templatemetasprache im Textfeld eingegeben werden. Durch Klicken des Buttons „Template hinzufügen“ wird das neue Template unter dem markierten Ordner/Template hinzugefügt. Es ist nicht möglich neue Anforderungsklassen hinzuzufügen.

Wenn alle gewünschten Templates hinzugefügt wurden, können die Änderungen durch Klicken des Buttons „Übernehmen“ in die Datei geschrieben werden.

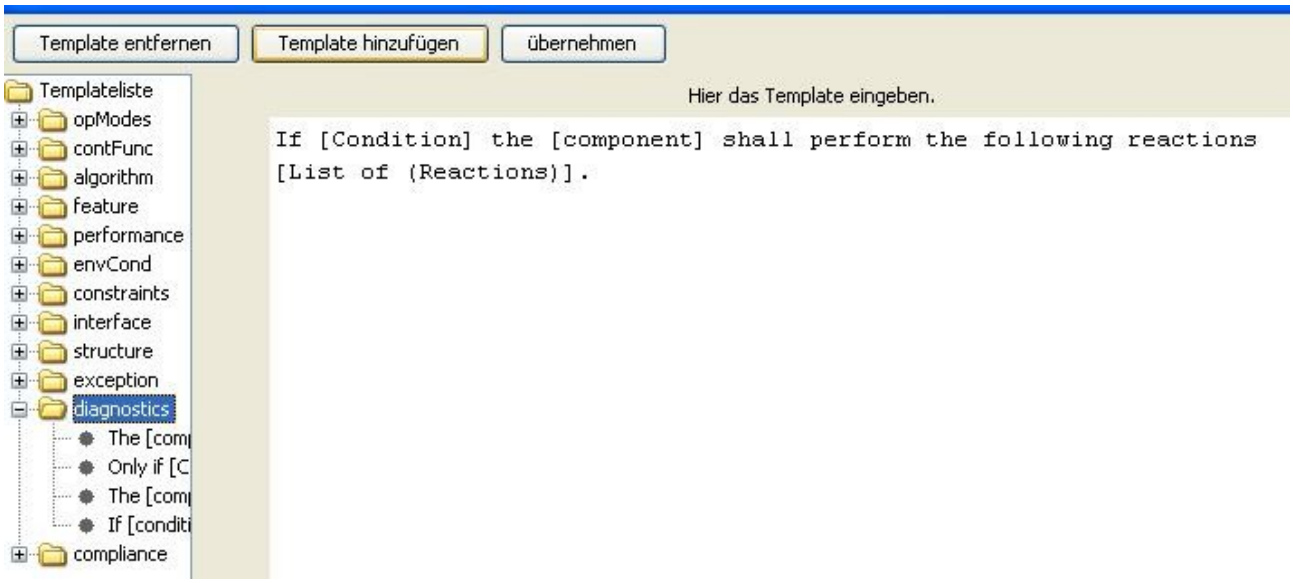

Abbildung 36: Darstellung zum Hinzufügen eines Templates

6.2.2.2 Template entfernen

Möchte man nun ein vorhandenes Template wieder entfernen, muss erst zum Templateeditor navigiert werden. Dort muss das zu entfernende Template markiert werden und durch Klicken des Buttons „Template entfernen“ wird dies aus dem Explorer gelöscht. Das endgültige Entfernen geschieht erst durch Klicken des Buttons „übernehmen“.

Abbildung 37: Darstellung zum Entfernen eines Templates

6.2.2.3 Globale Listen

Innerhalb des Prototypen existiert keine Unterscheidung zwichen projektbezogenen Listen und globalen Listen, deshalb wurden die use cases, die die Listen betreffen zusammengefasst. Dies betrifft use case 2.4: “Globale Liste bearbeiten” und use case 2.5: Globale Liste erstellen”.

6.2.3 Projektadministrator

6.2.3.1 Liste erstellen

Abbildung 38: Navigation zum Listeneditor

Damit der Projektadministrator die Möglichkeit besitzt eine neue Liste zu erstellen, muss er zum erforderlichen Editor navigieren. Dafür muss das Menü Listen und dort der Unterpunkt Listen angeklickt werden.

Es erscheint folgendes Fenster:

Abbildung 39: Darstellung des Listeneditors, wenn keine Liste gewählt wurde

In diesem Fenster muss nun der Button “Neue Liste erstellen” gedrückt werden. Dadurch erscheint ein Textfenster, in welchem der Name der Datei eingegeben werden muss. Dabei muss Groß- und Kleinschreibung beachtet werden, da der Prototyp ansonsten beim Erstellen der Anforderungen die entsprechende Datei nicht finden kann. Der Name der Datei muss somit identisch mit dem Namen der Regel sein.

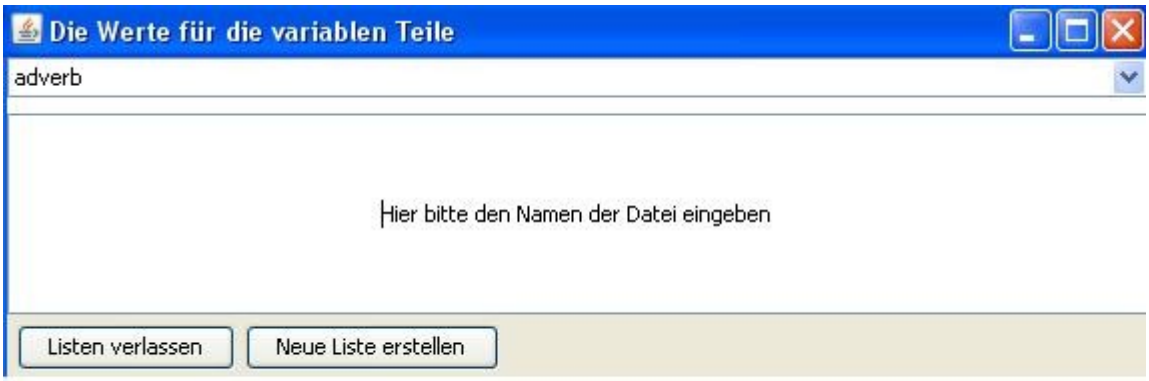

Abbildung 40: Darstellung des Listeneditors nach Klicken des Buttons "Neue Liste erstellen"

Anschließend erscheint die Datei in der obrigen Auswahlliste und kann bearbeitet werden.

6.2.3.2 Liste bearbeiten

Damit der Projektadministrator eine vorhandene Liste bearbeiten kann, muss erst einmal zum Listeneditor navigiert werden.

Im offenen Editor muss nun die Liste gewählt werden, welche bearbeitet werden soll. Wenn die gewüschte Liste angeklickt wurde, erscheinen die Werte in Form einer Tabelle. Zusätzlich erscheinen am unteren Bildrand drei neue Buttons, welche es ermöglichen die Liste zu bearbeiten.

Die Bearbeitungsmöglichkeiten sind das Hinzufügen von neuen Elementen und das Löschen von Elementen. Der “Übernehmen” Button sorgt dafür, dass die Änderungen übernommen werden.

Abbildung 41: Darstellung des Listeneditors, wenn eine Liste gewählt wurde

6.3 Speicherlösungen und Datenstrukturen

Die Elemente der einzelnen Variablen (Hardware, System, Failure, Test und weitere) werden als einfache Textdokumente gespeichert. Die Elemente werden durch Komma getrennt nacheinander aufgezählt.

Das Ziel ist es nicht den Text der Anforderung abzuspeichern, sondern ein Bauplan, der beinhaltet welches Template gewählt wurde, welche optionalen Teile gewählt wurden und welche Änderungen an den einzelnen Variablen vorgenommen wurden.

Dazu werden die Formalen Regeln als XML File gespeichert. Somit kann die notwendige Hierarchie, die die BNF vorgibt gut abgebildet werden und zusätzlich erhält jede Regel einen Identifikator der Form „r42“. Somit ist es möglich die einzelnen Veränderungen, die den Variablen widerfahren sind, nachzuvollziehen. Die Templatelisten werden ebenfalls als XML File abgespeichert. Einerseits damit die Hierarchie von Anforderungsklassen und ihren Templates besser dargestellt werden kann. Andererseits weil ebenfalls den Templates eigene Identifikatoren der Form „T5“ zugewiesen werden. Die Anforderungslisten werden zwecks der zahlreichen Zusatzinformationen (Identifikator, Bauplan, Kategorie, Hierarchiestufe) ebenfalls als XML File gespeichert. Außerdem ist die gesamte Hierarchie des Anforderungsdokumentes somit deutlich ersichtlicher.

Innerhalb des Prototypen werden für die Bereithaltung der Listenvariablen, sowie für die Anforderungsliste, Listen in Form vom Javatyp Vector verwendet, die die einzelnen Tabellenspalten beinhalten. Die Anforderungsliste wird, zwecks Navigation ‚zusätzlich noch in einer Baumstruktur bereitgestellt, damit der Explorer dargestellt werden kann.

Die Templatelisten werden einerseits als Parsebaum des XML Dokumentes bereitgestellt und andererseits als Baum für den Explorer zur Auswahl und zum Bearbeiten der Templates. Die Formalen Regeln sind ebenfalls als Parsebaum für die Verwendung im Prototypen bereitgestellt.

6.4 Anmerkungen

6.4.1 Der Bauplan der Anforderungen und JavaCC

Wie bereits erwähnt wird nicht der Inhalt einer Anforderung abgespeichert, sondern der Bauplan wie diese Anforderung entstanden ist. Dabei wird gespeichert welches Template gewählt wurde, welche optionalen Teile benutzt wurden und was mit den einzelnen Variablen geschehen ist. Die Variablen sind dabei durch Semikolons getrennt, damit die einzelnen Variablen unterscheiden

werden können. Der Bauplan wird letztendlich als ein zusammenhängender String dargestellt. Ein Bauplan sieht zum Beispiel folgendermaßen aus:

T4 O1:1 O2:0 O3:1 O4:1 r2 W: microcontroller; r17 W: automatically; r13 r10 W: analog input; W:default; r34 W: Terminal50; W:default;W:default;W:default;W:default;W:default;

Dabei steht T für Template, O für Optionaler Teil, r für eine Regel und W für einen Wert.

Die Schwierigkeit lag nun darin aus dem Bauplan wieder die Anforderung zu erstellen. Dabei fand JavaCC Verwendung.

JavaCC ist ein Plugin für Eclipse, welches es ermöglicht relativ einfach einen Compiler zu bauen. Dabei muss man als Entwickler die Zeichen der erwartenden Sprache definieren und in welche Reihenfolge diese auftreten. Dadurch ist es möglich einen Eingabestring zu validieren und auszuwerten.

Somit wurde der gelesene Bauplan an den Compiler weitergegeben. Da jeder Bauplan identisch aufgebaut ist, kann der Compiler den Bauplan validieren. Das heißt er überprüft, ob alle wichtigen Parameter enthalten sind, damit aus dem Bauplan eine Anforderung wird. Beim validieren speichert er die gefunden Werte in geeigneten Datenstrukturen ab. Wenn eine Anforderung fertig validiert wurde wird ein zweiter Compiler aufgerufen. Dieser bekommt das gewählte Template als Eingabestring und überprüft dieses Schritt für Schritt. Findet er dabei eine Variable, schaut er in der Datenstruktur, was mit dieser Anforderung bei ihrer Erstellung vollführt wurde und vollführt die entsprechenden Schritte. Dadurch entsteht aus dem Bauplan die vorher entwickelte Anforderung.

6.4.2 Einschränkungen

Da es sich lediglich um einen Prototypen handelt wurden einige Möglichkeiten innerhalb des Prototypen nicht generalisiert, sondern statisch implementiert.

Dies bedeutet das zum Beispiel innerhalb der Anforderungsliste nur Hierarchien bis zur Stufe 5 unterschieden werden. Alles was eine höhere Hierarchiestufe hat wird als 5 interpretiert.

Außerdem besteht nur die Möglichkeit Conditions einmal zu erweitern, also

Condition->Condition and Condition->Component is in mode and Component is in state

ist noch zulässig, nicht jedoch

Condition->Condition and Condition->Condition or Condition and Condition or not Condition

Diese Einschränkungen beziehen sich lediglich auf den entwickelten Prototypen. Auf die Methodik haben diese keine Auswirkungen.

7. Evaluation

Die Evaluation der Methodik erfolgt durch Verwendung des Prototypen. Somit wird dieser ebenfalls einer Evaluation unterzogen. Dafür wurde ein dreistündiger Termin mit Kollegen vereinbart, die als Probanden dienten.

7.1 Vorgaben vor dem Test

Die Vorgaben an die entwickelte Methodik und den damit verbundenen Prototypen waren folgendermaßen definiert.
Es sollen ungefähr 50% der Anforderungen von der Methodik abgedeckt werden können. Das heißt von allen Projektrequirements sollen ungefähr 50% aus den entwickelten Schablonen formuliert werden.

Weiterhin sollen Anforderungen von höherer Qualität, betreffend der Qualitätskriterien eindeutig, atomar, prüfbar und verständlich, entstehen. Der entwickelte Prototyp sollte ebenfalls bestimmten Kriterien entsprechen (Bedienbarkeit, Eignung der Templates, Verbesserung der Qualität und die Möglichkeit eigene Templates zu definieren)

7.2 Vorbereitung

Zur Vorbereitung musste jeder Proband 100 Beispielanforderungen aus aktuellen Projekten zuschicken. Diese sollten Repräsentanten verschiedener Klassen[9] und unterschiedlichen Domänen sein. Die Anforderungen wurden danach anhand ihrer Möglichkeit sie durch Templates auszudrücken bewertet[10]. Außerdem erfolgte eine Einteilung in die entsprechenden Klassen, wodurch das Finden der notwendigen Templates bei der Evaluierung für die Probanden erleichtert wird.

Von den 100 zugeschickten wurden im nächsten Schritt 30 Repräsentanten guter und mittlerer Qualität ausgewählt. Wobei ca. 5 gute und 25 mittlere gewählt wurden, da die entwickelte Methodik vor allem auf diese Qualitätsklassen angewendet werden soll.

Für die Evaluierung des Prototypen wurde zusätzlich ein Fragebogen angefertigt. Dieser soll am Ende Auskunft über die Qualität des Prototypen geben, so wie sie im vorherigen Kapitel beschrieben wurden.

7.3 Auswertung der Reviews

Im ersten Schritt wurden 3 mal 30 Anforderungen der Probanden anhand der 4 gegeben Qualitätskriterien (atomar, verständlich, eindeutig, prüfbar) bewertet. Dabei traten bei 31 von 90 Anforderungen Mängel in diesen Kategorien auf.

Zum Beispiel war die folgende Anforderung nicht atomar:

„The ECU shall have a FCI excitation connector, that is designed in Poka Yoke to prevent a swapping of phases.“

Im nächsten Schritt erfolgte die Erstellung der Anforderungen mit Hilfe des Prototypen. Die Probanden konnten dabei in 70 Minuten 28 Anforderungen erstellen. Somit hat jeder ca. 9 von 30 Anforderungen geschrieben. Zu erwähnen ist noch, dass die Evaluation für alle Beteiligten die erste Benutzung des Prototypen war.

Anschließend erfolgte ein erneutes Review, aber diesmal der neu erstellten Anforderungen. Die Probanden probierten Anforderungen guter und mittlerer Qualität zu erstellen, dadurch wurden

9 Kapitel 4.1.4
10 Kapitel 4.1.3

lediglich 13 der 31 als vorher schlecht bewerteten Anforderungen durch den Prototypen nochmals erstellt. Die anderen 16 waren bereits gute Anforderungen, wo die Qualität durch das Erstellen mit Hilfe des Prototypen in den meisten Fällen zumindest gleich geblieben ist. Es gab Einzelfälle wobei Anforderungen schlechterer Qualität entstanden sind, diese sind jedoch nach Analyse auf Bugs des Prototypen und die Erstbenutzung des Prototypen zurückzuführen.

Von den 13 erstellten Anforderungen weißten 7 eine Steigerung der Qualität auf. Dies sind 54%. Bei den restlichen 46% blieb die Qualität in etwa gleich.

„The ECU shall have a FCI excitation connector.

The FCI excitation connector shall be designed in Poka Yoke to prevent a swapping of phases.“

Die Verbesserung ist an der Veränderung der Anforderung zum obigen Beispiel deutlich zu erkennen. Diese Anforderung ist nun atomar, stellt aber den selben Inhalt dar, weshalb sie letztendlich auch leichter zu verstehen ist.

7.4 Bewertung des Prototypen

Für die Bewertung des Prototypen wurden Fragebögen erstellt, die nach der Benutzung des Prototypen ausgefüllt wurden. Dabei sollten verschiedene Aspekte auf einer Skala von 1-5 bewertet werden. Diese konnten durch Kommentare erläutert werden. Es folgt eine Auflistung der Aspekte mit den einzelnen Bewertungen. Die Sätze resultieren aus den Kommentaren der Probanden.

Bedienbarkeit des Prototypen:

Die Bedienbarkeit wurde als mittel bis gut eingestuft, weil der Stil für die Variablen nicht gut genug war und zum Beispiel die Möglichkeit der optionalen Parts durch Darstellung von Drop Listen oder auch Checkboxen intuitiver gewesen wäre.

Empfundene Verbesserung der Qualität

Laut Aussage der Probanden war die Qualität der Anforderungen bei bereits guten Anforderungen wie erwartend gleich gut geblieben, aber bei mittleren Anforderung trat eine kleine Verbesserung auf, insbesondere bei der Vereinheitlichung und Kürzung von Anforderungen (eindeutig und atomar werden).

Eignung der Methodik zur Erstellung von Anforderungen

Als Resultat ist die Eignung der Methodik zur Erstellung von Anforderungen mittel bis gut. Bei einer deutlichen Verbesserung des Prototypen, sowie mehr Übung der Benutzer und bessere Templates ist dies noch steigerbar. Die Methode eignet sich bei einigen Klassen von Requirements deutlich besser, wie bei anderen. Diese wären vor allem die Diagnose Anforderungen, da diese meist sehr ähnlich sind und sich lediglich in den Variablen und ihren Werten unterscheiden.

Eignung der Methodik durch das Erstellen eigener Templates

Die Eignung der Methodik durch das Erstellen eigener Templates ist mittel, da sich eigene Templates zwar recht schnell erstellen lassen, nur bei kleinen Rechtschreibfehlern ist das Template nicht nutzbar, da die Variable nicht erkannt wird. Außerdem benötigt man etwas Übung beim Erstellen, da man beim Benutzen von neuen Variablen erst eine entsprechende Liste dafür anlegen muss. Durch eine interaktive Möglichkeit, das heißt kleine Variablenbausteine direkt beim Erstellen der Anforderung mit einbinden zu können, könnte diese Methode eventuell verbessert werden.

7.5 Zusammenfassung der Evaluation

Aus den 300 gegeben Anforderungen ließen sich mit Hilfe der Templates ca. 30% erstellen. Dies entspricht somit nicht den Vorgaben, was sich aber dadurch erklären lässt, dass es mit wenigen Templates wirklich schwer fällt die gesamte Breite eines Projektes abzudecken. Konzentriert man sich dabei nur auf gewissen Klassen von Requirements, wie zum Beispiel Diagnoseanforderungen erreicht man eine Abdeckung von ca. 60%. In diesem Bereich gibt es deutlich weniger spezielle Anforderungen.

Eine Verbesserung der Qualität erreichte man in 54% der Anforderungen, die Mängel in den vier Kategorien aufwiesen.

Der benötigte Mehraufwand ist schwer einzuschätzen, da es sich um die Erstbenutzung der Probanden handelt. Nach einer gewissen Eingewöhnungsphase und genug Erfahrung mit dem Prototypen ließen sich in der gleichen Zeit sicherlich deutlich mehr Anforderungen erstellen.

Der Prototyp wurde alles in allem als mittel bis gut eingestuft. Könnte jedoch durch entsprechende Programmierkenntnisse und viel Arbeitsaufwand verbessert werden.

Reviewbogen und Fragebogen befinden sich im Anhang. Die Beispielanforderungen und Templates befinden dich im nicht öffentlichen Anhang.

8. Fazit und Ausblick

Zum Abschluss der Arbeit folgt ein Fazit, wo ich nochmal auf die gesetzten Ziele eingehe und noch einmal kurz die Herangehensweise erläutere. Im zweiten Teil des Fazits folgt eine persönliche Meinung, was mir die Bachelorarbeit gebracht hat. Anschließend wird ein kleiner Ausblick gewährt, was basierend auf der Methodik eventuell verbessert oder noch entwickelt werden könnte.

8.1 Fazit

Das Ziel dieser Arbeit war das Entwickeln einer Methodik zur Verbesserung der Qualität von natürlichsprachlichen Anforderungen und die entwickelte Methodik anhand eines Prototypen zu testen. In diesem Rahmen wurden erst einmal allgemeine Qualitätskriterien an Anforderungen vorgestellt und im späteren Verlauf erläutert, in wie fern die entwickelte Methodik diese Qualitätskriterien beeinflussen kann. Im weiteren Verlauf erfolgten Erklärungen allgemeiner Grundlagen, die zur Heranführung an die Methodik notwendig war. Anschließend erfolgte die Erläuterung der Methodik der Variablen Templates und die use cases, die die Methodik erfüllen sollte. Darauf folgte die Vorstellung des Prototypen anhand von Erklärungen, in welcher Weise die vorher definierten Anforderungen umgesetzt wurde.

Wie man bereits an der Evaluation erkennen kann, stellt die in dieser Arbeit vorgestellte Methodik einen möglichen Ansatz dar, Anforderungen höherer Qualität zu erstellen. Die Methodik erscheint auf dem ersten Blick sehr zeitaufwändig für die einzelnen Requirements Engineers, da diese die benötigten Variablenlisten erstellen und befüllen müssen. Dieser Aufwand entsteht jedoch lediglich bei Erstverwendung der Methodik. Bei folgenden Benutzungen greift man auf die für dieses Projekt erstellten Listen zu und bei neuen Projekten auf die bereits erstellten globalen Listen. Es entsteht somit lediglich am Anfang eines Projektes ein scheinbarer Mehraufwand. Doch meiner Meinung nach ist dies kein Mehraufwand, da die Requirements Engineers sich durch diesen Vorgang bereits vor dem Erstellen der Anforderungen mit den einzelnen Variablen auseinander setzen müssen. Außerdem können dadurch Glossars erstellt werden, die zum Beispiel auftretende Abkürzungen genauer beschreiben. Die Glossars geraten in vielen Projekten zwecks Zeitmangel oft in Vergessenheit. Somit kann man abschließend sagen, dass die gesetzten Ziele erreicht wurden. Es wurde eine Methode entwickelt, die natürlichsprachliche Anforderungen besserer Qualität erstellen kann.

Nun zu meinem persönlichen Fazit. Ich hatte mir von der Bachelorarbeit zwei Sachen erhofft. Erstens weitere Erfahrungen im Bereich des Requirements Engineering zu sammeln, da ich großes Interesse daran hege. Zweitens wollte ich meine Programmierkenntnisse erweitern.

Beide persönlichen Ziele wurden dabei zu genüge erreicht. Ich hatte im Rahmen dieser Bachelorarbeit viel Kontakt mit echten Kundenanforderungen und den Personen, die diese begutachten müssen. Dadurch habe ich viele Erfahrungen gesammelt, welche Probleme dabei auftreten können. Sei es schlechte Anforderungen, mangelnde Kommunikation oder einfach das strikte Verbot diese zu ändern. Zusätzlich hatte ich viel Kontakt mit den dortigen Werkstudenten, die andere Programme von Berner&Mattner warten und mitentwickeln. Dadurch habe ich zusätzlich einiges über das Testen von Software und der Entwicklung von Testfällen gelernt.

Meine Programmierkenntnisse konnte ich durch die Entwicklung des Prototypen ebenfalls deutlich verbessern. Da innerhalb des Studiums wenig mit GUI Anwendungen hantiert wird, war dies eine lohnenswerte Erfahrung für mich. Dadurch habe ich viel Wissen im Bereich der Entwicklung einer GUI mit Hilfe von Swing erlangt. Am Ende war ich mit dem entwickelten Prototypen recht zufrieden, obwohl noch viel Arbeit erledigt werden muss bis dieser wirklich einsatzfähig werden würde.

8.2 Ausblick

Betrachtet man abschließend die Methodik und den Prototypen wird ersichtlich, dass es viele Möglichkeiten gibt diese zu erweitern.

Zum Beispiel lassen sich Anforderungen, welche Bedingungen enthalten, leicht in Testfälle umwandeln, so dass diese später ebenfalls vom Tool selber mit erzeugt werden könnten. Diese sind dann wiederum einheitlich, da einer Anforderungsart ein Template zu Grunde liegt.

Außerdem besteht die Möglichkeit, welches ebenfalls Herr Holtmann als Ziel hatte, aus den Anforderungen automatisch Modelle zu generieren. Diese Modelle können dadurch in den weiteren Entwicklungsschritten zu Hilfe genommen werden.

Durch die Verwendung eines Tools zum Erstellen der Anforderungen besteht ebenfalls die Möglichkeit die Vollständigkeit der Anforderungen besser zu gewährleisten. Wird zum Beispiel eine Anforderung erstellt die irgendein System einschaltet, so kann automatisch eine ähnliche Anforderung erstellt werden, die das selbe System wieder abschaltet. Lediglich die Bedingungen unter welchen Umständen das System ein- beziehungsweise ausgeschaltet werden soll unterscheiden sich. Da solche Beispiele recht häufig vorkommen liefert dies zusätzlich noch eine erhebliche Zeiteinsparung.

Wenn die Wirtschaft weiterhin fest auf DOORS zählt, kann diese Methodik auch als DOORS Plugin entwickelt werden, so wie es zum Beispiel DESIRe schon ist. Dort könnte es vor allem im Rahmen der Diagnostik- und Testanforderungen die Arbeit der Requirements Engineers erleichtern.

9. Abbildungsverzeichnis

10.Literaturverzeichnis

[Ebert 10] Christof Ebert, Requirements Engineering: Advanced Topics, Zugriff: 18.10.2010 , Quelle: *http://www.computer.org/portal/web/buildyourcareer/ts011*

[Sant] Dr.Santen,Prof. Hußmann, Softwaretechnologie 2, Technische Universität Dresden, Zugriff: 29.10.2010, Quelle: *http://www.mpipks-dresden.mpg.de/~mueller/download/old_uni/scripte_swt2/st2k2a-2.pdf*

[Rupp10] Christine Rupp, Requirements Templates – The Blueprint of your Requirements, Zugriff: 07.11.2010 Quelle: *http://www.softwarequalitaet.at/pages/texte/Fachtagung_2008/Article%20-%20Requirements%20Templates%20-%20The%20BluePrint%20of%20your%20Requirements.pdf*

[ITWissen] ITWissen, BNF(Backus-Naur-Form), Zugriff: 22.10.2010, Quelle: *http.://www.itwissen.info/definition/lexikon/Backus-Naur-Form-BNF.html*

[Rupp et al 07] Chris Rupp & die SOPHISTen, Requirements- Engineering und Management, Hanser Verlag, 4. Auflage, Deutschland, 2007, Seite 227 f.

[Broso et al 06] Sebastian Brosowski et. al., Der Systembegriff; Zugriff: 23.10.2010 Vom: xx.xx.2006, Quelle: *https://www.ph-ludwigsburg.de/fileadmin/subsites/2e-imix-t-01/user_files/Journal_NEI_-_PDFs_fuer_Webauftritt/Section_B/Volume_1_No_1_2006/NEI_Section_B_Vol._1_No._1_2006_p._01-10_-_Brosowski_Gierz_Spannagel_-_Systembegriff.pdf*

[Prech et East 10] Lutz Prechelt, Steve Easterbrook, Requirements Elicitation, Zugriff: 07.11.2010, Quelle: *http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2010/21_Anforderungserhebung.pdf*

[Rupp 2006] C. Rupp, Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis, 4th ed. Hanser Fachbuchverlag, 10 2006.

[Prech,Brueg,Dut] Lutz Prechelt, Bernd Bruegge, Allen H. Dutoit, Modeling with UML, Course Softwartechnik Chapter 2, Zugriff: 27.10.20010, Quelle: *http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2010/14_UML.pdf*

[Prech] Lutz Prechelt, Anwendungsfälle(use cases) , Course Softwaretechnik Chapter 4, Zugriff: 05.11.2010, Quelle:
http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2010/22_Usecases.pdf

[Holt10] Jörg Holtmann, Mit Satzmustern zu textuellen Anforderungen, 2010,
Zugriff: 10.12.2010, Quelle:
http://www.cs.uni-paderborn.de/uploads/tx_sibibtex/
holtmann_OS_RE_2010.pdf

[Toro et al 99] A. Durán Toro et. al. A Requirements Elicitation Approach Based in Templates and Patterns WER´99 Proceedings , 1999

[ISO26262] ISO26262 Baseline 17, Part 8, Chapter 6

[Coll99] Michael Collins, Formal Methods, erstellt: Frühling 1999, Zugriff: 15.12.2010
http://www.ece.cmu.edu/~koopman/des_s99/formal_methods/

11. Anhang

11.1 Öffentlicher Anhang

11.1.1 Reviewbogen

Sollte eine der folgenden Qualitätskriterien **nicht** zutreffen, tragen Sie bitte den entsprechenden Buchstaben in der Zelle ein.

atomar (A)

Ist die Anforderung in mehrere Anforderungen zerlegbar, ohne den Inhalt zu verfälschen? Ist sie so lang, dass das Verständnis oder die Implementierung / der Test dadurch erschwert werden?

verständlich (V)

Ist die Anforderung für eine Person, die technisches und methodisches Grundverständnis besitzt, nicht jedoch spezialisierte Kenntnisse aus dem konkreten Projektumfeld, verständlich? Könnten Sie die Anforderung in eigenen Worten jemandem erklären oder sie im Grundsatz implementieren / testen? Sind alle vorkommenden Begriffe entweder einem allgemein ausgebildeten Ingenieur/Informatiker klar, oder im Projektglossar definiert (in dieser Studie: Komponenten-/Parameter-/Signal-Tabellen)?

eindeutig (E)

Ermöglicht die Anforderung verschiedene Interpretationsmöglichkeiten, die alle plausible sein könnten?

prüfbar (P)

Ist die Anforderung test- beziehungsweise verifizierbar (bei nicht testbaren Anforderungen sollte zumindest ein Analyseverfahren oder eine Review-Frage benennbar sein)? Kann man die Anforderung auf irgendeiner Art und Weise nachmessen oder nachweisen?

11.1.2. Fragebogen zum Prototypen

Bedienbarkeit:	Sehr schlecht	schlecht	Mittel	Gut	Sehr gut
Kommentar:					
Empfundene Verbesserung der Qualität der Anforderungen:	Viel schlechter	Schlechter	Gleich gut	Etwas besser	Deutlich besser
Kommentar:					
Eignung der Methodik für Anforderungen	Sehr schlecht	schlecht	Mittel	Gut	Sehr gut
Kommentar:					
Eignung der mitgelieferten Templates	Sehr schlecht	schlecht	Mittel	Gut	Sehr gut
Kommantar:					
Eignung der Methodik durch das Erstellen eigener Templates	Sehr schlecht	schlecht	Mittel	Gut	Sehr gut
Kommentar:					

11.2 Nicht öffentlicher Anhang