



**Hochschule
Bonn-Rhein-Sieg**
University of Applied Sciences

Abschlussarbeit

im Studiengang Informatik

Konzipierung und Implementierung einer Web-Komponente zur Anforderungserhebung mit dem Sophist-Regelwerk

von

Hakimjon Turaboev

Erstprüfer: Prof. Dr. Manfred Kaul

Zweitprüfer: Prof. Dr. Sascha Alda

eingereicht am 02.09.2019

Inhaltsverzeichnis

Eidesstattliche Erklärung	iii
1. Einleitung.....	1
1.1. Problemstellung	1
1.2. Motivation.....	1
1.3. Stand der Forschung.....	2
1.4. Ziel der Arbeit.....	2
1.5. Aufbau und Vorgehensweise.....	2
2. Theoretische Grundlagen	3
2.1. Konzept DMS	3
2.1.1. Makerspace	3
2.1.2. Beispiele zum Makerspace	4
2.1.3. DMS	4
2.1.4. Funktion der DMS	5
2.2. Das SOPHIST-Regelwerk.....	6
2.2.1. Erklärung zum SOPHIST-Regelwerk.....	6
2.2.2. Prozesse und die damit verbundenen Regeln.....	9
2.2.3. Eigenschaften und damit verbundene Regeln.....	13
2.2.4. Mengen, Häufigkeit und damit verbundene Regeln.....	15
2.2.5. Mögliches und Unmögliches sowie die damit verbundenen Regeln	17
2.2.6. Ganzer Anforderungssatz und damit verbundene Regeln.....	18
2.2.7. Gesamtbild und damit verbundene Regeln	19
2.2.8. Anwendung aller SOPHIST-Regelwerk Regeln	22
2.3. MASTeR-Schablonen.....	23
2.3.1. FunktionsMASTeR ohne Bedingung und mit Bedingung.....	24
2.3.2. Detaillierter FunktionsMASTeR ohne Bedingung und mit Bedingung.....	27
2.3.3. EigenschaftsMASTeR.....	28
2.3.4. UmgebungsMASTeR	29
2.3.5. ProzessMASTeR	29

2.3.6. BedingungsMASTeR	30
3. Vorausgesetzte Technologien.....	31
3.1. Benötigte Sprachen.....	31
3.1.1. HTML.....	31
3.1.2. CSS	32
3.1.3. JavaScript.....	33
3.2. Webbrowser	34
3.3. CCM (Client-side Component Model)	34
3.4. Balsamiq Mockups	35
3.5. WebStorm	35
3.6. GitHub.....	36
4. Konzeption und Implementierung	37
4.1. Konzeption	37
4.1.1. Realisierungsüberlegungen	37
4.1.2. Anforderungen	41
4.1.3. Mockup	42
4.2. Implementierung.....	44
4.2.1. Vorbereitung der Implementierung.....	44
4.2.2. Implementierung der MASTeR-Schablonen.....	45
5. Umfrage.....	52
6. Zusammenfassung	54
Abkürzungsverzeichnis	55
Literaturverzeichnis.....	56
Abbildungsverzeichnis	60

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Name: Hakimjon Turaboev

Unterschrift: _____ Sankt Augustin, den 02.09.2019

1. Einleitung

Ausgangspunkt dieser Arbeit sind grundsätzliche Überlegungen und Vorarbeiten zu den Themen Sophist-Regelwerken und Schablonen, die hier für Konzipierung und Implementierung einer Web-Komponente zur Anforderungserhebung mit den MASTeR-Schablonen genutzt werden sollen.

Ziel der Arbeit ist es, eine Komponente zu entwickeln, die zukünftig ein Bestandteil des Digital Makerspace (DMS) sein soll. Im Folgenden werden die Problemstellung und die Motivation der Arbeit dargestellt und der aktuelle Forschungsstand erhoben. Anschließend wird das Vorgehen in dieser Arbeit genauer beschrieben.

1.1. Problemstellung

Im Projektmanagement gibt es die Phase „Requirements Management, welches für die Erstellung von Anforderungen zuständig ist. Die Anforderungen im Projektmanagement werden normalerweise in natürlicher Sprache verfasst. Da sprachliche Anforderungen meistens nicht präzise sind, wurden bestimmte Regeln für Anforderungen als Sophist-Regeln aufgestellt. Um diese Regeln zu veranschaulichen, haben einige Autoren sogenannte Schablonen erstellt und veröffentlicht. Diese Schablonen sind hilfreich und verständlich, weil eine effiziente Anwendung in der Praxis dadurch erleichtert wird. Als Komponente im Web wurden diese Schablonen noch nicht entwickelt. Aus diesem Grund beschäftigt sich die nachfolgende Arbeit mit dem Konzipieren und dem Implementieren von Schablonen auf der Basis von Sophist-Regeln.

1.2. Motivation

In der Bildung wurde seitens der Lehrkräfte bisher für die Gestaltung der elektronischen Lehrmittel viel Zeit investiert. Mit DMS können LernApps leichter erstellt und anderen zur Verfügung gestellt werden. Was für die Benutzer auch nützlich sein kann, ist, dass diese LernApps Open Source sind und von verschiedenen Interessenten implementiert und erweitert werden können.

Ziel dieser Arbeit ist es eine weitere Komponente auf der DMS-Plattform zu veröffentlichen. Darüber hinaus können noch zusätzlich die folgenden neuen Möglichkeiten angeboten werden:

- Erleichterte Erstellung von elektronischen Lehrmitteln zum Thema Sophist-Regelwerk-Schablonen.
- Weltweites Teilen und Nutzen der Sophist-Regelwerk-Schablonen als DMS-Link
- Zeitersparnisse während der Lehre und effizienteres Lernen des RE(Requirements Engineering) mithilfe von Sophist-Regelwerkschablonen.

1.3. Stand der Forschung

Der aktuelle Forschungsstand sieht so aus, dass andere Entwickler zur Anforderungserhebung mit den Sophist-Regelwerk-Schablonen bislang noch keine Komponenten erstellt haben. Verschiedene Veröffentlichungen (Rupp/Joppich 2014; Kluge 2016) beschäftigen sich jedoch mit der Nutzerfreundlichkeit der Schablonen und Zusammenfassung der Sophist-Regeln.

1.4. Ziel der Arbeit

Das Ziel dieser Arbeit ist es, eine Web-Komponente für die Anforderungserhebung mit den Sophist-Regelwerk-Schablonen zu erstellen. Dadurch soll die Erstellung von Anforderungen in Software Engineering(SE) erleichtert werden. Die Implementierung soll sich u.a. auf die Schablonen von Roland Kluge stützen. Nach der Fertigung der Web-Komponente soll das Ergebnis in der Lehre eingesetzt und evaluiert werden.

1.5. Aufbau und Vorgehensweise

Um sich dem Ziel der Arbeit schrittweise zu nähern, ist es in erster Linie wichtig die theoretischen Grundlagen zu erläutern, die in Kapitel 2 behandelt werden. Hier wird der Forschungsstand betrachtet und ein Überblick über die bisherigen Schablonen der Sophist-Regeln gegeben. Danach werden die vorausgesetzten Technologien, die während der Arbeit benötigt werden, vorgestellt.

Aufgrund dieser Vorarbeiten kann im darauffolgenden Kapitel ein Konzept für die praktische Umsetzung entwickelt werden. Diese Umsetzung erfolgt im nächsten Schritt und wird im Kapitel 4 beschrieben.

Nach der Implementierung wird eine Umfrage durchgeführt und die Usability der Komponente bewertet. Dazu werden mehrere Benutzer eingeladen, um die Komponente zu benutzen und auf die Fragen zu antworten. Diese Ergebnisse werden im Kapitel 5 dargestellt und diskutiert.

2. Theoretische Grundlagen

In diesem Kapitel werden wichtige theoretische Grundlagen erläutert, damit das Ergebnis der Arbeit und die spätere Implementierung der Komponente für den Leser verständlich werden. Zuerst gibt es einige Informationen über DMS und darüber wie das Projekt entstanden ist und welche Vorläufer existieren. Darüber hinaus wird die Hauptfunktion des Tools erläutert. Die danach kommenden Themen sind Sophist Regelwerk und die existierenden Schablonen, die ein Teil der Hauptbestandteil der Arbeit sind. Am Ende wird das Kapitel mit dem Thema Client-side Component Model (CCM) vervollständigt, welches den Kern der Implementierung dieser Arbeit darstellt.

2.1. Konzept DMS

Zunächst wird der Begriff physischer Makerspace anhand einiger Beispiele erläutert. Danach wird der DMS vorgestellt und seine Funktionen erläutert. Abschließend wird auf den aktuellen Forschungsstand des DMS eingegangen.

2.1.1. Makerspace

Makerspace ist ein Ort, an dem Menschen zusammenkommen und lernen, wie Materialien verwendet werden und kreative Projekte entwickelt werden können. Im Makerspace fördert leicht und spielerisch das Erwerben von Wissen. Ein Makerspace kann in einem Klassenzimmer, in einer Bibliothek oder sogar in einem eigenständigen Gebäude organisiert werden. Makerspace gibt es in unterschiedlichen Formen und Größen und sind meistens mit der Verwendung von Technologie und Entwürfen verbunden. Im Normalfall ermöglicht Makerspace:

- Spielen, Fördern, Erkundigen und Lernen
- Die Nutzerfreundlichkeit informeller Lernangebote und Verbindungen zwischen zu Hause, Schule und Gemeinschaft
- Lernen im Team, Kombination von Fähigkeiten und Kenntnisse von Pädagogen und Studenten.
- Steigerung der Kreativität im alltäglichen Leben.¹

Der Kerngedanke von Makerspace ist es neue Dinge zu entdecken, zu erschaffen oder bereits bestehende Ideen zu verbessern.

Das Sammeln, Basteln und andere künstlerische und handwerkliche Aktivitäten gab es schon vor dem 21. Jahrhundert. Am Anfang der 2000er begann aber die Bewegung

¹ www.makerspacesaustralia.weebly.com

„the maker movement“. Sie betonte die praktische, handgemachte Entdeckung in neuem Jahrhundert.²

2.1.2. Beispiele zum Makerspace

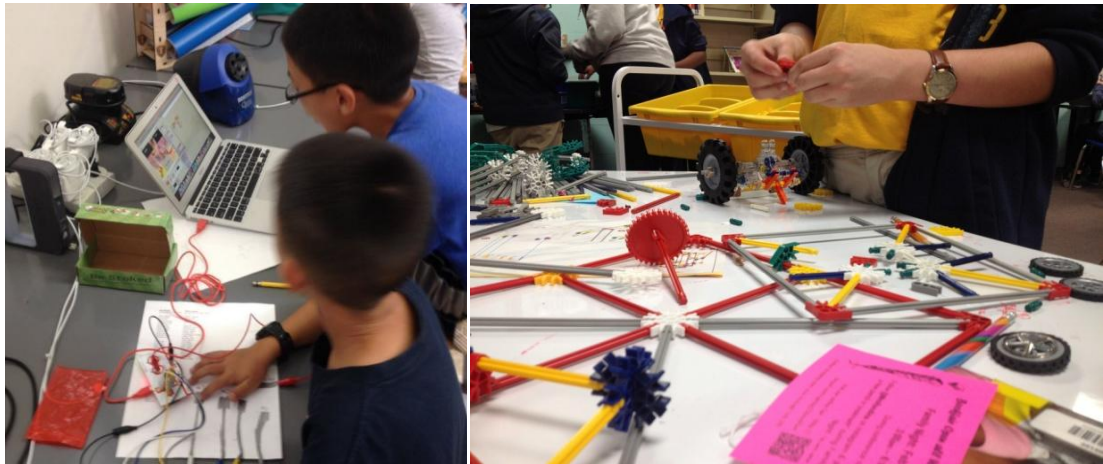


Abbildung 1. Links K-6 School Makerspace From @iolaniLSFabLab und rechts Stewart-Middle-School-Educational-Makerspace-@DianaLRendina

Makerspace erlaubt den Zugang zu günstiger Technologie, die die Verbesserungen in der Fertigung und Herstellung ermöglichen und die Fähigkeiten eines Studenten erweitern und neue Möglichkeiten für die Herstellung von Dingen schaffen. Digitale Fertigungsgeräte, wie 3D-Drucker oder Physical Computing wie Arduino und Raspberry Pi sind sehr gute Beispiele für den Einsatz von Technologie in Makerspace. In Abbildung 1 kann man sehen, wie in etwa ein Makerspace Klassenzimmer aufgebaut sein könnte.³

2.1.3. DMS

Der Begriff DMS wurde oft von verschiedenen Organisationen benutzt, doch der Begriff wurde nicht exakt für gleiche Zwecke verwendet. In der Universität North Carolina Wilmington ist DMS ein Ort für die Kooperation zwischen der Bibliothek Randall und des Technologie-Assistenzentrums. Das Ziel ist es, dass die neuen und modernen Technologien wie 3D-Druck und Virtual Reality an einem Ort für Studenten und Lehrkräfte zur Verfügung gestellt werden können.⁴

Das Team „Conversation X Labs“ hat eine online Projektplattform erstellt und hat diese DMS genannt. Ihr Zweck ist es, dass verschiedene Gemeinschaften um verschiedene

² <https://www.weareteachers.com>

³ <http://makerspacesaustralia.weebly.com>

⁴ <https://digitalmakerspace.uncw.edu>

Naturschutzprojekte zusammenkommen, damit sie die technikbasierten, innovativen Lösungen für Umweltprobleme entwickeln können.⁵

Bei dem DMS, welches in der Hochschule Bonn-Rhein-Sieg entwickelt wurde, sind alle Tätigkeiten online verfügbar. „Mit dem Konzept DMS wurde das populäre "Makerspace"-Konzept auf das Digitale übertragen.“⁶ Diese DMS bietet online Komponente (Produktionsmittel), die von Usern zu jeder Zeit und von jedem Ort benutzt werden können. Ihr Zweck ist es, dass Lehrende und Studierende die neuen Komponenten erstellen oder mit bereits erstellten Komponentendurch Konfiguration für Ihre Zwecke anpassen und weiterleiten können⁷. In der Abbildung 2 ist die Komponente des DMSs abgebildet.

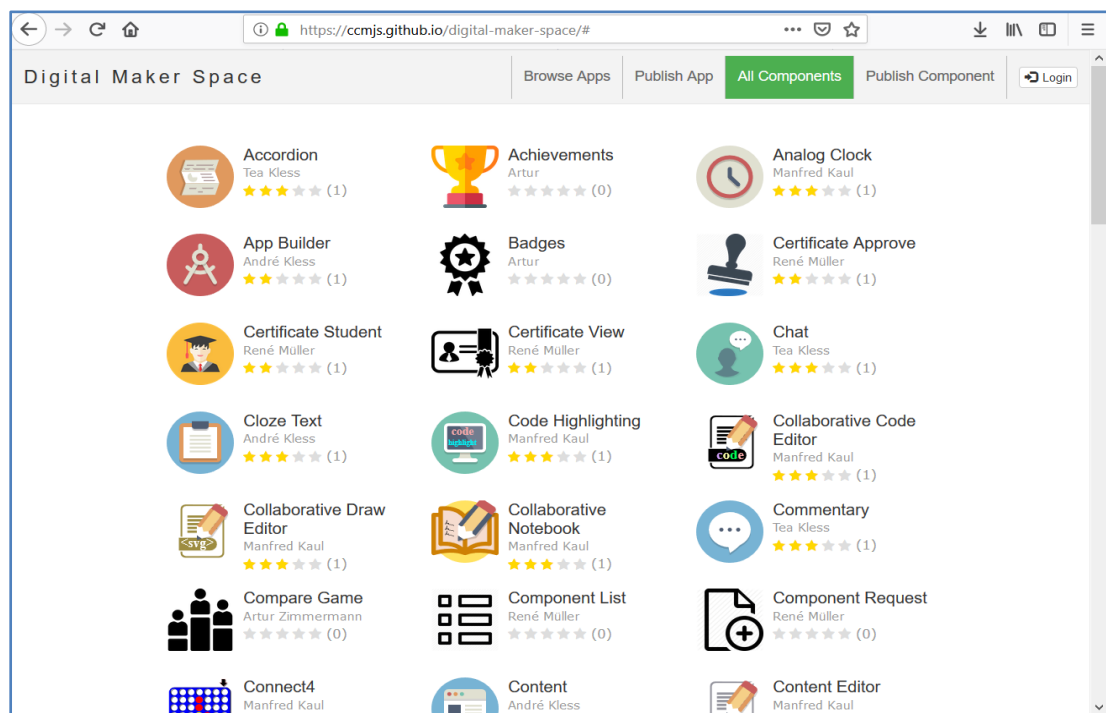


Abbildung 2: Screenshot von DMS⁸

2.1.4. Funktion der DMS

Die World Wide Web (WWW)-Komponenten werden in DMS als Grundsteine benutzt und sind einbettbar für verschiedene Webseiten. Diese DMS-Komponente können auf der Plattform ausgewählt, nach Wunsch bearbeitet, weiterentwickelt und in einer eigenständigen Seite eingebettet werden. Diese erfordert das Ausfüllen und Anpassen der Formulare. Tiefere Programmierkenntnisse sind dabei nicht erforderlich.⁹ Das ferti-

⁵ <https://conservationxlabs.com>

⁶ <https://kaul.inf.h-brs.de>

⁷ <https://kaul.inf.h-brs.de>

⁸ <https://CCMjs.github.io>

⁹ <https://www.stifterverband.org>

ge angepasste Produkt kann als Lehrmaterial („Content“) benutzt werden.

Da diese Plattform die Bildung unterstützt und ein fester Teil von WWW und Open Educational Resources (OER) ist (siehe Abbildung 3), wird das Ergebnis dieser Arbeit in DMS als eine neue Komponente veröffentlicht, die in der vorliegenden Untersuchung entwickelt worden sind.

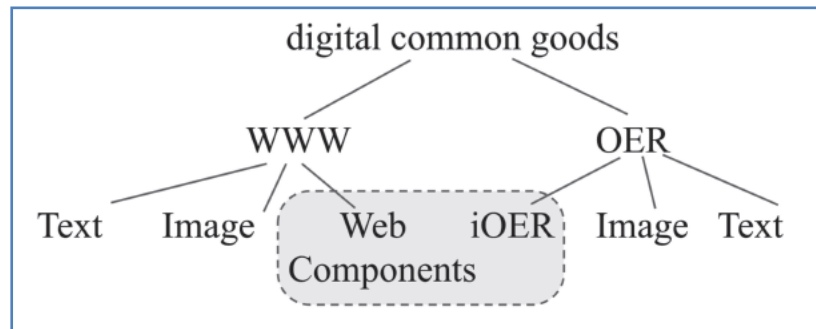


Abbildung 3. Bringing interactivity to digital common goods¹⁰

Die grundlegende Idee dieser Komponente basiert auf dem SOPHIST-Regelwerk, welches bei der Anforderungserhebung in RE (Requirement Engineering) hilft und im Folgenden erläutert wird.

2.2. Das SOPHIST-Regelwerk

SOPHIST-Regelwerk ist ein umfangreiches Thema, das in dieser Arbeit nicht vollständig und ausführlich besprochen werden kann. Deshalb wird dieses Thema in folgenden Unterkapiteln kurz und knapp erklärt.

2.2.1. Erklärung zum SOPHIST-Regelwerk

Anforderungen (User-Stories) werden von unterschiedlichen Auftraggebern erstellt. Das kann zu verschiedenen Problemen führen: Wichtige Informationen gehen verloren oder die Anforderungen sind nicht eindeutig und nur unvollständig formuliert. Aus diesem Grund muss beachtet werden, was der Stakeholder vom Projekthaben will. Die Wünsche und Ideen des Stakeholders müssen dementsprechend exakt dokumentiert werden. Da dieses Problem nicht einfach zu lösen ist, wurden von den SOPHISTen-Regeln entwickelt und aufgestellt. Die Sammlung dieser Regeln heißt „SOPHIST-Regelwerk“¹¹.

Bei diesem Regelwerk werden das linguistische-, psychologische-, psychotherapeuti-

¹⁰ Kaul, Student Activation in iOER Makerspaces

¹¹ Rupp, Chris & die SOPHISTen: „Requirements-Engineering und –Management, aus der Praxis von klassisch bis agil“. 2014, Seite 124

sche- und Informatik-Wissen zu einem Verfahren gebündelt.¹² So ist es möglich, die sprachlichen Defekte in Anforderungen zu beseitigen.

Die Vorteile des SOPHIST-REgelwerks sind u.a. folgende:

- Systematisches Vorgehen zur Analyse der Anforderungen.
- Qualitätsverbesserung der bereits dokumentierten sprachlichen Anforderungen.
- Überprüfen der Aussagen, die im Interview vom Stakeholder gesagt wurden.
Neue Fragestellung auf diese Aussagen.
- Nicht kommuniziertes Wissen aufdecken.
- Sowohl schriftliche als auch mündliche Anforderungen gleich, qualitativ, wesentlich verbessern.
- Lücken erkennen

„SOPHIST-REgelwerk ist eine Technik, die es Ihnen ermöglicht, Tilgungen, Generalisierungen und Verzerrungen in Anforderungen zu erkennen und somit fehlende und verzerrte Informationen aufzudecken“¹³

Um diese Aussage verständlicher zu machen, werden im Folgenden die Bedeutungen von Tilgungen, Generalisierung und Verzerrungen kurz erklärt:¹⁴

- Tilgung ist ein Prozess, in dem die Aussagen der Informationsgeber vom Informationsempfänger unvollständig angenommen werden.
- Generalisierung ist ein Prozess, in dem die gesamte Gruppe der verwandten Sachverhalte aufgrund der einmaligen persönlichen Erfahrung allgemein gleich dargestellt wird.
- Verzerrung ist ein Prozess, in dem die Informationen während der Übertragung an den Empfänger verfälscht oder umgestaltet werden. Der Empfänger stellt sich die Realität wegen der Verzerrung mit einem umgeänderten Bild vor.

Vor der Verwendung dieses Regelwerks muss man sich mit linguistischen Effekten, die die Anforderungsfehler verursachen können, fachlich auseinander setzen. Wie detailliert die Anforderungen dabei geschrieben werden, ist ein wichtiger Punkt, den es zu beachten gilt. In einem iterativen und inkrementellen Vorgehen werden die Anforderungen aber nicht so detailliert geschrieben. Ebenso wichtig sind die Kenntnisse, wie die

¹² Vgl. Ebd. S. 124.

¹³ Ebd. S. 133

¹⁴ Vgl. Ebd. S. 131 ff.

Prozesse in Projekten ablaufen. Um dabei das gewonnene Wissen exakt, vollkommen und korrekt zu dokumentieren, ist eine Sammlung von Regeln hilfreich.¹⁵

Dieses Regelwerk hilft bei der exakten, ordentlichen Formulierung der unordentlichen, ungenügend klar erklärten Äußerungen. Die Art der Wissensübergabe, ob diese beispielsweise mündlich oder schriftlich sind, spielt dabei keine Rolle.¹⁶

Mit folgenden drei Schritten kann jede Regel des SOPHIST-Regelwerks schnell und effektiv eingesetzt werden. Nach diesen Schritten werden zuerst die Signalwörter gesucht, um Mängel zu identifizieren. Anschließend werden zu den gefundenen Wörtern Fragen gestellt, um zu erfahren, ob die Informationen ausgefallen (Tilgung) oder verfälscht und umgeändert (Verzerrung, Generalisierung) wurden. Zum Schluss werden die sprachlichen Effekte und existierenden Fehler aussortiert, bereinigt und in einer neuen Form gestaltet (siehe Abb. 4).¹⁷

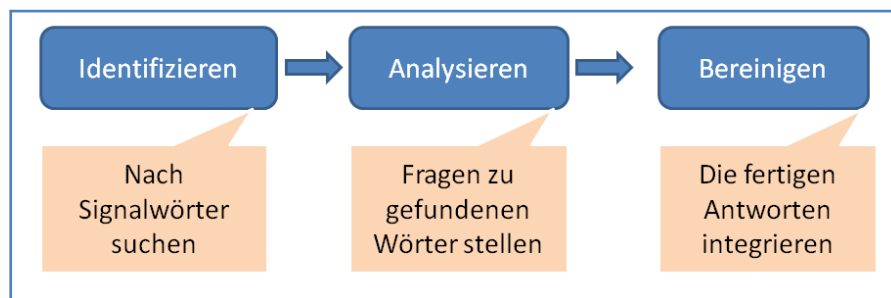


Abbildung 4. Vorgehen der Einsatz für jede Sophist-Regel (eigene Darstellung, nach Kluge)¹⁸

In dem nächsten Schritt wird festgelegt, wie der Satzbau nach Regeln formuliert werden kann. Dafür werden die Sätze in kleine Bestandteile geteilt, analysiert und wieder aufgebaut. Dieses ist in Abbildung 5 zu sehen.



Abbildung 5. Einsatzfolge des SOPHIST-Regelwerks, (eigene Darstellung, nach Kluge)¹⁹

¹⁵ Vgl. Ebd. S. 124

¹⁶ Ebd. S. 133

¹⁷ Vgl. Ebd.

¹⁸ Vgl. Ebd.

In Anlehnung an Kluge sollen die Anforderungen im ersten Schritt in kleine Satzbestandteile (1) geteilt und geprüft werden. In jedem Bestandteil muss ein Signalwort existieren, das beachtet werden muss. Da diese Wörter einen speziellen Effekt für die Anforderungen haben, wird die Regel nach diesem Effekt gewählt und verwendet. Danach wird geprüft, ob der ganze Satz (2) unnötige Informationen besitzt. Wenn der Anforderungssatz deswegen unübersichtlich ist, werden die unwichtigen Satzbestandteile ausgeklammert oder komplett ausgeschnitten. Zum Schluss wird geprüft, ob der komplette Anforderungssatz (3) zu dem gewünschten Gesamtbild passt. Wenn die gezielten Systemfunktionen oder -merkmale fehlen, kann das mithilfe der Signalwörter wieder vervollständigt werden.²⁰

Bei diesem Prozess und weiteren Anforderungsschreiben tauchen Schwierigkeiten auf, die zum Anfang jeder Arbeit auftreten können. Deshalb ist es wichtig, den roten Fadenaufrecht zu erhalten und die Grundregel zu beachten: „Zuerst die Anforderungen grob, aber vollständig aufzählen und dann vom Groben ins Feine abtauchen.“²¹

2.2.2. Prozesse und die damit verbundenen Regeln

Um die Bedeutungsvielfalt des Prozesswortes zu fixieren, ist es besser den Prozess mit „drucken“ oder „übertragen“ darzustellen. Diese Wörter geben die Erklärungen auf die W-Fragen (wer, wo, wann, worauf etc.). Dazu kommen noch die Regeln, die die Prozesse der Anforderungen normen:

„Regel 1: Formulieren Sie jede Anforderung im Aktiv.“²²

Diese Frage hilft nach Rupp Regel 1 anzuwenden: „Wer soll den Prozess ausführen bzw. über die Eigenschaft verfügen?“²³

Wenn der Satz im Passiv geschrieben wird, kann der Benutzer des Systems weggelassen werden. Das kann zum Verlust der Information führen, wer genau die Aktivität tätigt. Deshalb soll der Satz im Aktiv geschrieben werden. Dadurch ist es möglich, den Akteur einfach direkt zu identifizieren.²⁴

Um einen Passivsatz zu bestimmen, gibt es nach Günther gewisse Signalwörter. Diese sind „werden, wurden, sind“. Wenn eines dieser Wörter im Satz vorkommt, wird der Satz analysiert und soll ins Aktiv umgewandelt werden. Um den Satz zur Aktivform zu

¹⁹ Vgl. Ebd. S. 134

²⁰ Vgl. Ebd. S. 134 ff.

²¹ Vgl. Ebd. S. 135

²² Vgl. Ebd. S. 136

²³ Vgl. Ebd.

²⁴ Vgl. Ebd.

ändern, wird die oben genannte Frage gestellt. Mithilfe dieser Frage wird das Subjekt des Satzes gefunden und der Satz kann leicht ins Aktiv umgeformt werden. Hier ist dafür ein Beispiel: *Die Bezahlung muss berechnet werden.* „werden“ ist hier ein Signalwort für den Passivsatzstatus. Es wird gefragt, wer oder was den Prozess ausführen soll. Anschließend wird das Subjektwort „das System“ gefunden und die umgeformte Antwort ist: *Das System muss die Bezahlung berechnen.*²⁵

„Regel 2: Drücken Sie Prozesse durch „eindeutige/definierte“ Vollverben aus.“²⁶

Folgende Fragen helfen nach Rupp Regel 2 anzuwenden: „Welche konkrete Funktionalität wird gefordert? Wie lautet das VOLLVERB?“²⁷

Als Vollverb bezeichnet man die Verben, die alleine neben dem Subjekt in einem Satz stehen, das Prädikat des Satzes bilden und volle Taten ausdrücken. Zum Beispiel: „schreiben“ und „drucken“. Daneben gibt es noch einige andere Vollverben, die nicht verwendet werden dürfen, z. B. „verwalten“ und „prüfen“. Wenn der Prozess nicht im Vollverb ausgedrückt wird, ist es schwer, die linguistischen Effekte, die die Funktionalität der Anforderung verstecken, zu erkennen. Ohne Vollverb verstehen außerdem viele Projektteilnehmer die exakte, gezielte Bedeutung der Anforderung nicht. Es ist deshalb von hoher Bedeutung, Vollverben im Glossar einzutragen und gut zu definieren.²⁸

„Regel 3: Lösen Sie Nominalisierung auf.“²⁹

Diese Fragen helfen nach Rupp Regel 3 anzuwenden: „Ist das Substantiv eine Nominalisierung? Welcher konkrete Prozess steckt dahinter?“³⁰

Nominalisierung (Substantivierung) bezeichnet in der deutschen Sprache, dass die Wörter der anderen deutschen Wortarten die Eigenschaften der Substantive haben und genauso wie Substantive in den Sätzen verwendet werden.³¹ In RE entsteht eine Nominalisierung, wenn die Verben als Substantiv geschrieben werden. Durch die Nominalisierung werden die Informationen verzerrt, die für die Prozessbeschreibung wichtig sind. Da diese Art der Substantive auf Anforderungen sehr starken Einfluss hat, gibt es in RE eine Bauernregel: „Will der Stakeholder die Anforderungen verderben, macht

²⁵ Günther, Andreas: Das SOPHIST-REgelwerk - Andreas Günther Tipps Tricks und Grundlagen. 2019

²⁶ Vgl. Rupp, Ch. & die SOPHISTen: Requirements-Engineering und –Management, aus der Praxis von klassisch bis agil S. 137, 2014.

²⁷ Vgl. Ebd. S. 137

²⁸ Vgl. Ebd. S. 137 ff. und <https://www.schulminator.com/deutsch/vollverben-hilfsverben>

²⁹ Vgl. Ebd. S. 138

³⁰ Vgl. Ebd.

³¹ <https://www.cafe-lingua.de/deutsche-grammatik/substantivierung-von-wortarten.php>

er die Substantive aus den Verben.“³² Aus diesem Grund muss in jeder Anforderung die Nominalisierungen analysiert und geprüft werden. Falls sich dahinter ein Prozess verbirgt, soll in Vollverben umgewandelt werden. Wenn eine Anforderung nach dem Auflösen einer solchen Nominalisierung mehrere Prozessverben hat, wird für jedes Verb eine neue Anforderung geschrieben. Diese Verben müssen im Glossar eingetragen werden.³³

„Regel 4: Lösen Sie Funktionsverbgefüge auf.“³⁴

Diese Fragen helfen nach Rupp Regel 4 anzuwenden: „Wird der Prozess durch ein Funktionsverbgefüge ausgedrückt? Welcher konkrete Prozess steckt dahinter?“³⁵

Im Funktionsverbgefüge werden Funktionsverben und Nomen mit oder ohne Präposition kombiniert und bilden ein Prädikat im Satz. In dieser Kombination wird die Hauptbedeutung durch die Nomen getragen und als Vollverb wiedergegeben, z. B: einen Antrag stellen = beantragen.³⁶ Deshalb werden die Funktionsverbgefüge analysiert, geprüft, hinterfragt und die verschleierte Prozesse im System gefunden. Da manche Auftraggeber noch nicht genau wissen oder keine geplanten Vorstellungen davon haben, was exakt ein System machen muss, benutzen sie Funktionsverbgefüge. Infolgedessen wird nicht nur eine Anforderung geschrieben, sondern es entsteht unabsichtlich ein Epic oder Use-Case. In diesem Fall müssen diese Epics von RE-Fachleuten analysiert, geprüft und mithilfe von W-Fragen in einzelne Anforderungen übertragen werden.³⁷

„Regel 5: Schreiben Sie für jedes Prozesswort genau einen Anforderungssatz.“³⁸

Diese Fragen helfen nach Rupp Regel 5 anzuwenden: „Wie viele Prozesswörter(Verben) sind in der Anforderung enthalten?“³⁹

In anderen Regeln wurde versucht, die Funktionalitäten zu identifizieren und mit Vollverben zu schreiben. Diesbezüglich konnte eine Funktionalität mehrere Vollverben enthalten. Ein Vollverb ist ein Prozesswort in dieser Anforderung und soll mit anderen

³² Vgl. Ebd. S. 138

³³ Vgl. Ebd. S. 138 ff.

³⁴ Vgl. Ebd.

³⁵ Vgl. Ebd.

³⁶ <https://www.deutschplus.net/pages/Funktionsverben>

³⁷ Vgl. Rupp, Ch. & die SOPHISTen: Requirements-Engineering und –Management, aus der Praxis von klassisch bis agil S. 140 ff

³⁸ Vgl. Ebd. S. 141

³⁹ Vgl. Ebd. S. 142

Satzgliedern einen weiteren selbständigen Aufforderungssatz formulieren. Nach der Formulierung besteht eine Anforderung aus mehreren Anforderungssätzen. Nachfolgend wird ein Beispiel gezeigt:

- *„Das Kassensystem muss dem Verkäufer die Möglichkeit bieten, die Zahlungsdaten einzutippen und zu berechnen.“*

Jetzt kommen separate Anforderungssätze für jedes Vollverb:

- *„Das Kassensystem muss dem Verkäufer die Möglichkeit bieten, die Zahlungsdaten einzutippen.“*
- *„Das Kassensystem muss dem Verkäufer die Möglichkeit bieten, die Zahlungsdaten zu berechnen.“*

In diesem Fall ist es nützlich, dass die wahrscheinliche Verschleierung vermieden wurde. Es ist aber ungünstig, einen Satz in zwei Sätze mit aufwendiger Dokumentation umzuwandeln.⁴⁰

„Regel 6: Analysieren Sie fehlende Informationen zum Vollverb.“⁴¹

Diese Fragen helfen nach Rupp Regel 6 anzuwenden: „Wie oft /wie/wann/unter welchem Randbedingungen/an wem wird die Funktionalität ausgeführt?“⁴²

Es gibt Prozesswörter, die zum Informationsmangel in dem Anforderungssatz führen. In diesem Fall sollen die fehlenden Informationen gesucht und analysiert werden. Wenn die Tilgung der wichtigen Funktionalität entstanden ist, sollen die W-Fragen an das Prozesswort gestellt werden. Mithilfe dieser Fragen kann die fehlende Information gefunden und existierende Defekte gelöst werden. Folgende Fragewörter werden in den meisten Fällen benutzt: *„was, wem, wie, wann und wie oft“*. Der daraus folgende Vorteil ist die Verhinderung der Defekte. Jedoch ist die Umwandlung in eine standardisierte und defektlose Struktur aufwendig. Außerdem können so zusätzliche Anforderungssätze entstehen.⁴³

Als Zusammenfassung kann gesagt werden, dass alle Regeln der Prozesse, die bisher vorgestellt wurden, mit folgenden vier Schritten umgesetzt werden können:

⁴⁰ Vgl. Ebd. S. 141 ff.

⁴¹ Ebd. S. 142

⁴² Ebd. S. 143

⁴³ Vgl. Ebd. S. 143



Abbildung 6. Die vier Schritte zum Prüfen der Prozesse⁴⁴

Wie in Abbildung 6 zu sehen ist, wird im 1. Schritt die Anforderung im Aktiv formuliert, im 2. Schritt die konkreten Prozesse bestimmt: Dabei werden die Regeln 2 bis 4 umgesetzt. Der 3. Schritt ist Regel 5, und zwar wird jede Funktionalität in den Anforderungen mit einem einzelnen Satz geschrieben. Am Ende wird Regel 6 als 4. Schritt angewendet.⁴⁵

2.2.3. Eigenschaften und damit verbundene Regeln

Die Eigenschaften des Systems sollen auch wie andere Systemteile klar und komplett beschrieben werden. Diese Beschreibungen sind nicht funktional und werden normalerweise mit Adverbien und Adjektiven formuliert. Diese Wortarten reichen aber in der Praxis nicht aus, vollständige Systemeigenschaft auszudrücken. Diesbezüglich sollen die Signalwörter noch einmal identifiziert, geprüft und die fehlenden Informationen der Eigenschaft vervollständigt werden.

Das Prüfen der Eigenschaften wird in zwei Schritten angewendet. Dies ist in nachfolgender Abbildung zu sehen.



Abbildung 7. Schritte für Eigenschaftsprüfung⁴⁶

⁴⁴ Ebd. S. 144

⁴⁵ Vgl. Ebd.

⁴⁶ Ebd. S. 148

Im ersten Schritt werden die Sophisten-Regeln 7 und 8 verwendet. Nach der Regel 7 müssen zunächst fehlende Informationen gefunden werden.

„Regel 7: Hinterfragen Sie fehlende Informationen zu beschriebenen Eigenschaften.“⁴⁷

Diese Fragen helfen nach Rupp Regel 7 anzuwenden: „Für wen/für was/wann/unter welchen Randbedingung wird die Eigenschaft gefordert?“⁴⁸

Mit dem Hinterfragen der Adjektive und Adverbien kann die Tilgung erkannt und geprüft werden, ob die Prozesse mit Eigenschaftswörtern beschrieben sind. Dabei dürfen die nicht-funktionalen Aspekte nicht vergessen werden, die das Substantiv und die Verben ergänzen, z. B.:

„installiertes“ Programm, „ausgeschaltet“ transportieren.

Das System soll die unterschriebene Quittung erkennen. Nachdem der Kunde die Quittung unterschrieben hat, soll das System diese Quittung erkennen.

„Regel 8: Formulieren Sie Eigenschaftswörter mess- bzw. testbar.“⁴⁹

Diese Fragen helfen nach Rupp Regel 8 anzuwenden: „In Bezug bzw. Vergleich zu wem bzw. was? Wie können Erfüllung bzw. Abweichung gemessen werden?“⁵⁰

Um die Anforderung nach dieser Regel zu schreiben, soll zuerst der Bezugspunkt des Eigenschaftsworts gefunden und die fehlende Information vervollständigt werden. Dabei sollen auch die Steigerungs- und Vergleichsformen der Adjektive und Adverbien beachtet werden, weil diese Formen die Genauigkeit des Systems falsch zeigen. Damit kann die Anforderung nicht gemessen, geprüft oder getestet werden. Deshalb ist es immer zu empfehlen, die Genauigkeit vorher zu bestimmen, z. B. *Der Drucker soll die Papiere „schneller“ ausdrucken.* Mit „schneller“ ist in diesem Fall „10 Seite pro Minute“ gemeint. In diesem Beispiel konnte das Problem mit den Messwerten gelöst werden. Aber es gibt auch Wörter, die nicht gemessen werden können, wie beispielweise das Wort „benutzerfreundlich“. Dieses Wort kann je nach Kunde und Personengruppe unterschiedlich verstanden werden. Aus diesem Grund ist es sinnvoll, Stakeholder detailliert zu fragen, wie das Wort „benutzerfreundlich“ zu definieren ist.

⁴⁷ Ebd. S. 145

⁴⁸ Ebd. S. 148

⁴⁹ Ebd. S. 145

⁵⁰ Ebd. S. 148

Eine hilfreiche Frage ist dabei „Was ist nicht benutzerfreundlich?“⁵¹

Wie oben informiert wird jetzt Schritt 2 umgesetzt. Dabei ist Regel 9 relevant, die folgendes aussagt.

„Regel 9: Formulieren Sie eigene Anforderungen für nicht-funktionale Anforderungen.“⁵²

Diese Fragen helfen nach Rupp Regel 9 anzuwenden: „Ist die Eigenschaft eigenständig tastbar? Wird sie global gefordert?“⁵³

Nach dieser Regel werden die Aspekte in zwei Gruppen (funktionale und nicht-funktionale) geteilt, wenn die nicht-funktionalen Aspekte getestet werden sollen oder mehrere Funktionalitäten verlangt wurden. Da je nach Kunden diese Aspekte unterschiedlich gefordert werden können, können diese entweder mit Anforderungen oder mit einem Akzeptanzkriterium zusammengeschrieben werden.⁵⁴

2.2.4. Mengen, Häufigkeit und damit verbundene Regeln

Um Redundanzen zu vermeiden, wird für einen Sachverhalt nur eine allgemeine Beschreibung geschrieben. In dieser Beschreibung werden oft die Zahl- und Mengenwörter als Signalwörter benutzt. Dabei kommen die Regeln 10, 11 und 12 zum Einsatz.

„Regel 10: Hinterfragen Sie verwendete Zahl- und Mengenwörter.“⁵⁵

Folgende Fragen helfen nach Rupp Regel 10 anzuwenden: „Gilt das Verhalten/die Eigenschaft für wirklich alle Objekte der Menge? Oder gibt es auch Ausnahmen?“⁵⁶

Diese Regel sagt, dass die Zahl- und Mengenwörter geprüft werden sollen. Wenn festgestellt wird, dass die Eigenschaft nicht für alle hinterfragten Objekte gilt, soll die Spezifikation der Eigenschaft in der Anforderung eingetragen werden, z. B.:

Das Kassensystem soll dem Verkäufer die Möglichkeit bieten, alle Produkte zu messen.

Das Wort „alle“ muss in diesem Kontext genau hinterfragt werden. Des Weiteren muss herausgestellt werden, ob Ausnahmen zugelassen werden können. Das oben beschriebene Beispiel hat Ausnahmen und die umgeänderte Version lautet wie folgt:

⁵¹ Vgl. Ebd. S. 147

⁵² Ebd.

⁵³ Ebd.

⁵⁴ Ebd. S. 147 ff.

⁵⁵ Ebd. S. 148 ff.

⁵⁶ Ebd. S. 149

Das Kassensystem soll dem Verkäufer die Möglichkeit bieten, die „messbar“ gekennzeichneten Produkte zu wiegen.“

„Regel 11: Klären Sie fehlende Zahl- und Mengenwörter.“⁵⁷

Diese Fragen helfen nach Rupp Regel 11 anzuwenden: „Für welche Objekte genau soll das geforderte Verhalten bzw. die Eigenschaft gelten?“⁵⁸

Nach dieser Regel soll die Menge der Objekte geklärt, geprüft und definiert werden. Wenn festgelegt wird, dass die Mengenwörter fehlen, sollen die passenden Wörter hinzugefügt werden: *Das Kassensystem muss dem Verkäufer die Möglichkeit bieten, die Verkaufsdaten zu speichern.* Um diese Anforderung zu hinterfragen, werden die Signalwörter „immer“, „jeder“, „alle“, „mehrfach“ und „parallel“ benutzt. Die normierte Form des Beispiels lautet:

*Das Kassensystem muss jedem Verkäufer jederzeit die Möglichkeit bieten, alle Verkaufsdaten zu speichern.*⁵⁹

„Regel 12: Hinterfragen Sie schwammige Substantive.“⁶⁰

Diese Fragen helfen nach Rupp Regel 12 anzuwenden: „Wer/Was...genau? Welcher Teil der genannten Menge?“⁶¹

Nach dieser Regel sollen die Substantive, die die Objekte ungenau beschreiben, hinterfragt werden. Die daraus ermittelten Informationen sollen in der Anforderung hinzugefügt bzw. geändert werden, z. B.: *Das System soll jedem Anwender die Möglichkeit bieten, die Daten zu speichern.* Die nach der Regel umgeänderte Form lautet: *Das Kassensystem soll jedem Verkäufer die Möglichkeit bieten, die von Kunden angenommenen Bezahlungsdaten zu speichern.* Nachfolgend wird zusammengefasst, wie die sprachlichen Defekte schrittweise identifiziert und behandelt werden. Das ist in der Abbildung 8 zu sehen.

⁵⁷ Ebd. S. 150

⁵⁸ Ebd.

⁵⁹ Vgl. Ebd.

⁶⁰ Ebd. S. 151

⁶¹ Ebd.



Abbildung 8. Die zwei Schritte zum Prüfen von Mengen und Häufigkeiten⁶²

2.2.5. Mögliches und Unmögliches sowie die damit verbundenen Regeln

Es reicht nicht immer aus, nur die Anforderungen zu dokumentieren, sondern es muss auch gefragt werden, wie diese Anforderungen umgesetzt werden oder funktionieren sollen. Diesen fachlichen Ablauf bezeichnet man als „Geschäftslogik“ (Business Logic). Um eine Geschäftslogik zu erkennen, helfen Signalwörter wie „kann“, „darf“, „es ist (nicht) möglich“ oder „er/sie/es ist außerstande“. Wichtig ist zu verstehen, dass die Geschäftslogik nicht die fachliche Implementierungsbeschreibung ist, wie z. B.: *Ein Framework wird verwendet*. Wenn das Verhalten der Anforderung mit „möglich“ oder „unmöglich“ beschrieben wird, geschieht die Tilgung der Geschäftslogik. Deshalb sagt das SOPHIST-Regelwerk 13:

„Regel 13: Klären Sie Mögliches und Unmögliches.“⁶³

Diese Fragen helfen nach Rupp Regel 13 anzuwenden: „Was macht das Verhalten möglich/unmöglich? Welche fachliche Logik steckt dahinter? Welcher Teil der genannten Menge?“⁶⁴

Nach dieser Regel muss geprüft werden, ob die Vorbedingungen gültig und die fachliche Logik stimmig sind und was das System möglich oder unmöglich macht. Manchmal wird auch ein Teil der Anforderungen getilgt und in diesem Fall soll kontrolliert werden, ob die wichtigen Vorbedingungen und Voraussetzungen dokumentiert wurden z. B. *Es muss unmöglich sein, dass ein Kunde, der eine ausländische Bankkarte besitzt, bezahlen kann*. Wie das erfüllt wird, weiß der Stakeholder genau. Für diese Aussage wäre eine Maßnahme:

⁶² Ebd. S. 152

⁶³ Ebd. S. 152

⁶⁴ Ebd. agil S. 153

Wenn eine Karte eine unbekannte Bankleitzahl hat, darf das System die Bezahlung nicht akzeptieren.

Als Zusammenfassung werden die sprachlichen Defekte um das Thema der Möglichkeiten und Unmöglichkeiten wie in nachfolgender Abbildung gezeigt, gelöst.



Abbildung 9. Ein Schritt zum Prüfen von Möglichem und Unmöglichem⁶⁵

2.2.6. Ganzer Anforderungssatz und damit verbundene Regeln

Wie schon oben ausgeführt wurde, sollen außer Satzbestandteilen auch die kompletten Anforderungssätze geprüft werden. Da einige Informationen im Satz nicht wichtig sind, sind diese aus der Anforderung zu entfernen. Mithilfe folgender Signalwörter können solche Nebensätze bestimmt werden: „weil“, „damit“, „um“, „deshalb“, „so dass“ etc.

„Regel 14: Extrahieren Sie Nebensätze, die für die Anforderung nicht notwendige Informationen enthalten.“⁶⁶

Diese Frage hilft nach Rupp Regel 14 anzuwenden: „Welche Relevanz für die Anforderung hat die Information im Nebensatz?“⁶⁷

Nach dieser Regel sollen die Nebensätze nachgefragt werden. Wenn sie aus unnötigen Informationen bestehen, werden diese Nebensätze herausgelöst, aber die zeitlichen und logischen Nebensätze und Kommentare sind erwünscht und können daher unberührt bleiben z. B. *Um dem Verkäufer die Tagesstatistik zu verdeutlichen, muss das Kassensystem die tägliche Summe von Null anfangen.* Dieses Beispiel zeigt, dass der Nebensatz mit „um...zu...“ als Kommentar herausgezogen und allein geschrieben werden muss.

⁶⁵ Ebd. S. 154

⁶⁶ Ebd. S. 154

⁶⁷ Ebd. S. 154

Das Kassensystem muss dem Verkäufer die Möglichkeit bieten, die tägliche Summe von Null anzufangen. „Kommentar: Dadurch wird die tägliche Summe vom Verkäufer deutlich erkannt“⁶⁸

„Regel 15: Vermeiden Sie redundante Informationen und Floskeln in der Anforderung.“⁶⁹

Diese Frage hilft nach Rupp Regel 15 anzuwenden: „Wird etwas doppelt oder mit Floskeln ausgedrückt?“⁷⁰ Nach dieser Regel werden die redundanten Wörter und Satzteile, komplizierte Redewendungen gestrichen und umformuliert. Dabei soll keine Information verloren gehen z. B. *Da der Verkäufer beliebig viele falsche Produkte zu beliebig verschiedenen Zeiten beliebig oft scannen kann, muss das System das Scannen beliebig zurücksetzen können.*

In diesem Fall werden einige Wörter gestrichen und der Satz folgenderweise formuliert: *Solange der Verkäufer die Produkte falsch scannt, muss das System das Scannen zurücksetzen.*⁷¹

Nach allen Schritten sieht unser Prozessbild für den ganzen Anforderungssatz folgendermaßen aus.



Abbildung 10. Schritte zum Prüfen des Satzes⁷²

2.2.7. Gesamtbild und damit verbundene Regeln

Schließlich kann das komplette Gesamtbild und zwar die Gesamtspezifikation betrachtet werden. Dabei müssen die Anforderungen das gesamte Verhalten des Systems beschreiben. Deshalb müssen beim Erstellen der Anforderungen alle Ausnahmefälle, Abweichungen und Randbedingungen, die das System beeinflussen, hinterfragt wer-

⁶⁸ Vgl. Ebd. S. 154 ff.

⁶⁹ Ebd. S. 155

⁷⁰ Ebd.

⁷¹ Ebd. S. 155 ff.

⁷² Ebd. S. 156

den. Diese Einflüsse kann man anhand von Signalwörter wie: „müssen“, „sollen“, „es ist notwendig“ etc. erkennen.⁷³

„Regel 16: Klären Sie Abweichungen vom Normalverhalten.“⁷⁴

Diese Fragen helfen nach Rupp Regel 16 anzuwenden: „Kann das System das geforderte Verhalten immer sicherstellen? Was passiert, wenn nicht? Welche Ausnahmen sind möglich?“⁷⁵

Nach dieser Regel werden die nicht normalen Verhaltensfälle, die vom System nicht erwartet werden und später eintreten können, geklärt, z. B. *Das Kassensystem soll die täglichen Daten in externen Medien speichern.* Für diese Anforderung soll eine Maßnahme geschrieben werden, wenn dieses Verhalten nicht erfüllt wird, z. B.: *Wenn das Kassensystem die täglichen Daten auf den Medien nicht speichern kann, muss das System die Fehlermeldung „Speichern ist nicht möglich“ anzeigen.*⁷⁶ Im nächsten Schritt wird Regel 17 verwendet.

„Regel 17: Analysieren Sie unvollständige Bedingungsstrukturen.“⁷⁷

Folgende Fragen helfen nach Rupp Regel 17 anzuwenden: „Wie soll sich das System verhalten, wenn die Bedingung nicht eintritt? Gibt es noch weitere Fälle?“⁷⁸

Nach dieser Regel werden die unerfüllten Fälle der Anforderungsbedingung betrachtet. Um diese Bedingung zu erkennen, helfen folgende Signalwörter „wenn ...“, „dann“, „falls ...“, „dann“, „im Falle von“ und „abhängig von“ z. B. *Wenn die Kundenkarte nicht gesperrt ist, dann muss das Kassensystem dem Kunden die Bezahlung ermöglichen.*

Es wurde aber hier nicht angegeben, was gemacht werden muss, wenn die Kundenkarte gesperrt ist. In solchen Fällen können die Anforderungen ergänzt oder die neuen Anforderungen geschrieben werden. In diesem Beispiel wird für den Fall, dass die Bedingung nicht erfüllt wurde, eine zusätzliche Anforderung geschrieben:

Wenn die Kundenkarte gesperrt ist, muss das Kassensystem dem Kunden die Karte zurückgeben.

Es kann aber auch mehrere Fälle für das Nichteintreten einer Anforderung geben. Auch ist es möglich, dass ein Problem mithilfe einer Fallunterscheidung nicht gelöst werden

⁷³ Ebd. S. 156 ff.

⁷⁴ Ebd. S. 157

⁷⁵ Ebd.

⁷⁶ Ebd.

⁷⁷ Ebd. S. 158

⁷⁸ Ebd.

kann. In diesem Fall soll eine Entscheidungstabelle als Darstellungsmittel benutzt werden.⁷⁹

Als letzter Schritt wird die implizite Annahme, die getilgte Aussagen sind, analysiert. Manchmal denken die Stakeholder, dass einige Sachverhalte, Aussagen selbstverständlich sind und sagen in der Beschreibung nicht Sätze wie z. B. Ich will Wasser haben. Der Stakeholder denkt, dass das Wort „kalt“ von jedem ergänzt werden kann und aus diesem Grund nicht ausdrücklich gesagt werden muss. In diesem Fall werden die Anforderungen getilgt. Wenn die Annahme des Stakeholders in der Beschreibung nicht wahr ist, werden die anderen Anforderungen keinen Sinn geben. Deshalb wird auf dieses Problem während der ganzen Dokumentation geachtet. Infolgedessen haben alle bisherigen 17 Regeln teilweise implizite Annahmen aufgedeckt. Regel 18 beschäftigt sich damit intensiv.⁸⁰

„Regel 18: Analysieren Sie implizite Annahmen.“⁸¹

Diese Fragen helfen nach Rupp Regel 18 anzuwenden: „Steckt in der Anforderung eine implizite Annahme? Ist diese Aussage bereits irgendwo in der Spezifikation beschrieben?“⁸²

Es gibt auch für diese Regel Signalwörter, auf die geachtet werden soll. Beispiele hierfür sind Wörter wie: „Falls“, „Nachdem“, „Sobald“, „Wenn“, „eingegebene Benutzerdaten“, „angezeigter View“, „berechneter Kapitalwert“ etc. Das Prüfen der Anforderung funktioniert nach dieser Regel folgenderweise: Identifizieren des Signalworts → Prüfen implizite Annahme mit diesem Signalwort → Prüfen der implizite Annahme in anderen Beschreibungen → explizite Bestimmung der nicht beschriebenen, impliziten Annahme. Z. B. *Nachdem das Kassensystem die registrierte Paybackkarte gezeigt hat, soll das System die Kundenpunkte mitberechnen.* In diesem Beispiel ist „registrierte Paybackkarte“ ein Bezugswort.

Nachfolgend wird eine Abbildung gezeigt, in der Zusammenfassend die Reihenfolge erläutert wird, in der das Gesamtbild geprüft werden soll.⁸³

⁷⁹ Vgl. Ebd. S. 158 ff.

⁸⁰ Vgl. Ebd. S. 159

⁸¹ Ebd. S. 160

⁸² Ebd. S. 160

⁸³ Vgl. Ebd. S. 160 ff.

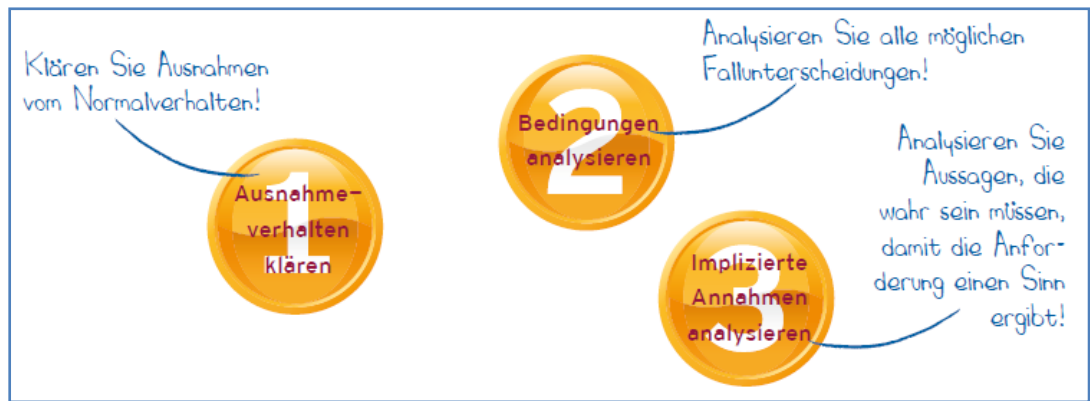


Abbildung 11. Schritte zum Prüfen des Gesamtbildes⁸⁴

2.2.8. Anwendung aller SOPHIST-Regelwerk Regeln

Wie oben behandelt wurde, sind die Regeln effektiv. Zum Einsetzen der Regeln bedarf es jedoch Übung. Geübte Stakeholder können selbst versuchen, während der Kommunikation Defekte zu vermeiden. Das spart Zeit und Aufwand. Es gibt mehrere Prioritäten bei den Regeln, wie in der folgenden Abbildung zu sehen ist. Wenn die Zeit für den Einsatz aller Regeln nicht ausreichend ist oder die Benutzer nicht mit allen Regeln vertraut sind, können nur ein paar Regeln der hohen Priorität angewendet werden.

Prioritäten	NIEDRIG	MITTEL	HOCH
SOPHIST-Regelwerk	4, 9, 13, 14, 15	1, 5, 7, 8, 10, 11, 16	3, 6, 12, 17, 18

Abbildung 12. Prioritäten von SOPHIST-Regelwerk nach Rupp und eigene Darstellung⁸⁵

Dafür gibt es eine feste Regel: „Ist die Zeit dir weggerannt, nimm nur ein paar Regeln in die Hand.“⁸⁶Eine weitere Hilfe für Anfänger sind die MASTeR-Schablonen. Diese Schablonen werden als klare Vorgaben benutzt und die Anforderungen in SE nach diesen Mustern formuliert. Infolgedessen können einige Regeln automatisch umgesetzt werden.

Ausführliche Informationen über diese Schablonen werden im nächsten Kapitel gegeben.

⁸⁴ Ebd. S. 161

⁸⁵ Vgl. Ebd. S. 163

⁸⁶ Ebd. S. 163

2.3. MASTeR-Schablonen

Wie oben erklärt, verlangt die Anwendung des SOPHIST-REgelwerks von dem Anfänger viel Zeit und Erfahrung. Deshalb wurde ständig eine Formulierungsmöglichkeit des Anforderungssatzes gesucht, die für alle RE-Kunden einfach und übertragbar ist. Wegen dieses Bedarfs und aufgrund Ihrer Interesse hat das SOPHISTen-Team während der Beratungstätigkeiten bei den unterschiedlichen Unternehmen versucht, die Anforderungssätze nach Schablonen zu formulieren. Diese Formulierungen sind die Anforderungsschablonen. Rupp beschreibt diese Schablone wie folgt: „Eine Anforderungsschablone ist ein Bauplan, der die Struktur eines einzelnen Anforderungssatzes festlegt.“⁸⁷

Folglich wurde eine erste Schablone, die „MASTeR“⁸⁸ Schablone, für die funktionalen Anforderungen entwickelt.

Jede Anforderungsschablone wurde so formuliert, dass damit möglichst minimale Kosten und maximaler Nutzen erreicht werden kann. Mit dem Versuch zur Verbesserung dieser Anforderungsschablone hat das Team in Projekten weitere Schablonen entwickelt⁸⁹. Mit der Berücksichtigung und Unterscheidung der zeitlichen, logischen Bedingungen in MASTeR-Schablonen können sogar die funktionalen und nicht-funktionalen Anforderungen getrennt geschrieben werden.⁹⁰ Nachdem die MASTeR-Schablonen vom SOPHISTen-Team jahrelang bearbeitet und verbessert wurden, existieren heute folgende Schablonen:

- FunktionsMASTeR
- EigenschaftsMASTeR
- UmgebungsMASTeR
- ProzessMASTeR
- BedingungsMASTeR

Im Folgenden wird erklärt, wie diese Schablonen ausgefüllt werden sollen.

Nach Kluge werden optimierte Formulierungsmöglichkeiten auf folgende Fragen ständig gesucht:

- a) Wie formuliert man Objekt und Prozesswort am besten?
- b) Was ist der Hauptsinn des Satzes?
- c) Was ist das Ergebnis von a und b und wie kann dieses die Schablonen beeinflussen?
- d) Wie können Bedingungen und diesbezüglich Konjunktionen konstruiert wer-

⁸⁷ Ebd. S. 7.

⁸⁸ Kluge, R., „Schablonen für alle Fälle“ 2016, S. 6

⁸⁹ Vgl. ebd.

⁹⁰ Vgl. ebd.

den?

2.3.1. FunktionsMASTeR ohne Bedingung und mit Bedingung

Je nach Antwort auf diese oben gestellten vier Fragen können die Anforderungen trotz des gleichen Satzbaus unterschiedliche Bedeutung ausdrücken. Von daher ergibt es keinen Sinn jede Idee in Schablonen einzutragen. In der Abbildung 13 wird detailliert erklärt, wie genau „FunktionsMASTeR ohne Bedingung“ ausgefüllt werden soll.

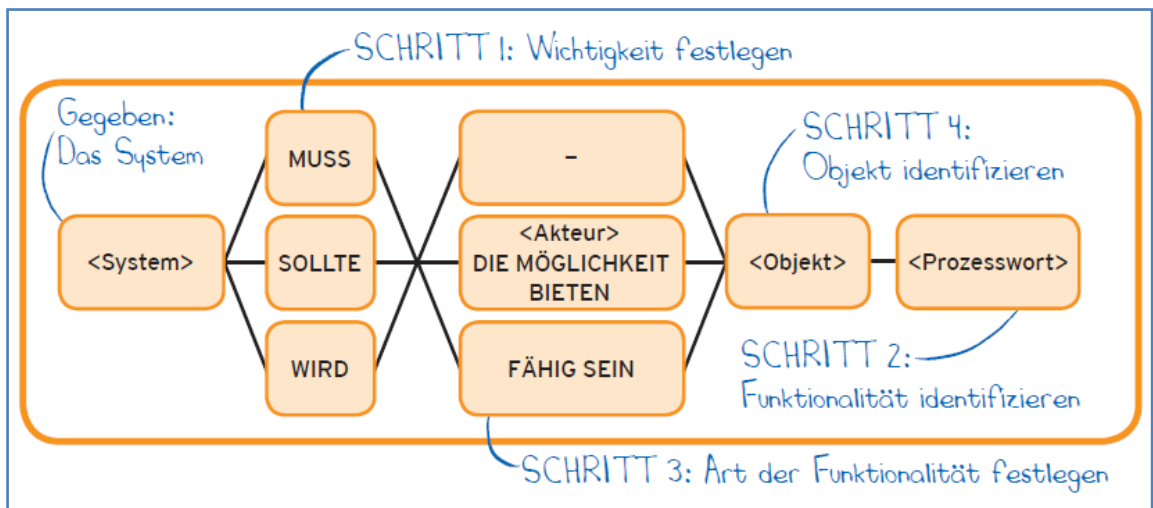


Abbildung 13. FunktionsMASTeR ohne Bedingung⁹¹

Wie in der Abbildung 13 zu sehen ist, gibt es vier Schritte für das richtige Ausfüllen der diese Arte der Schablonen, die nach der Vervollständigung eine gute Anforderung ergeben:

System: es ist selbstverständlich in jedem IT-Projekt gegeben. Wenn das System mit seinem konkreten Namen in die Schablone eingetragen wird, ist es konkret und verständlicher.

Zum Beispiel: Der **Kassenautomat** muss die Bezahlung berechnen.

Schritt 1: Zuerst soll der Wichtigkeitsgrad der Anforderungen bestimmt werden. Das wird zwischen Stakeholdern geklärt und vereinbart. Wenn eine wichtige Anforderung nach der Vereinbarung nicht umgesetzt wird, kann das gerichtliche Konsequenzen haben.

Es gibt drei Grade der Wichtigkeit, die mit vier Modalverben (MUSS, SOLLTE, WIRD, KANN) beschrieben werden.⁹²

⁹¹ Ebd. S. 11

⁹² Kluge, Roland: „Schablonen für alle Fälle“ 2016 Seite 12

- **MUSS:** Das bedeutet es ist Pflicht. Wenn die Muss-Anforderungen je nach Situation nicht erfüllt werden können, darf nicht MUSS geschrieben werden.
- **SOLLTE:** Diese Anforderungen werden vom Auftraggeber gewünscht, aber müssen nicht umgesetzt werden. Wenn sie umgesetzt werden, hat das positive Wirkung auf den Auftraggeber.
- **WIRD:** Dieses Wort drückt die Pläne der Stakeholder aus. Diese Pläne müssen bei der Entwicklung des Systems unbedingt beachtet werden, weil diese Anforderungen das zukünftig geplante Update oder die Veränderungen unterstützen. Ein Testen des Ergebnisses ist aber nicht vorgesehen.
- **KANN:** Die Anforderungen mit dieser Verbindlichkeit sind nur eine Lösungsvorstellung, wie die Funktionalität umgesetzt werden kann. Das ist aber nur ein Vorschlag. Aus diesem Grund helfen solche Anforderungen für die Einarbeitung und Verständnis der Entwickler.⁹³

Z. B: Kassenautomat **MUSS | WIRD | SOLLTE | KANN** die Bezahlung berechnen.

Schritt 2: Als Prozesswort wird die Grundform, d. h. der Infinitiv des Verbs, benutzt und das drückt die Hauptbedeutung der Anforderung aus. Da die falsche Auswahl des Prozessworts die zukünftige Implementierung negativ beeinflussen kann, muss darauf stark geachtet werden. Um die Unachtsamkeit zu vermeiden, sollen alle möglichen, projektbezogenen Wörter in einem Tool oder Dokument aufgeschrieben und mit den Stakeholdern besprochen werden.⁹⁴

Zum Beispiel: *Der Kassenautomat muss die Bezahlung **berechnen**.*

Schritt 3: Der Typ der Funktionalität soll bestimmt werden. In Anlehnung an Kluge können die Funktionen in folgende Arten differenziert werden und diese Arten werden mithilfe des folgenden Satzes betrachtet: Der Kassenautomat muss<Funktionalität>die Bezahlung berechnen.

- a) „Selbsttätige Systemaktivität“: Wenn eine Funktion ohne Hilfe des Benutzers gestartet wird, endet sie selbständig. Anstelle <Funktionalität> wird hier nichts geschrieben. Ergebnis: Der Kassenautomat muss<_____> die Bezahlung berechnen.
- b) „Benutzerinteraktion“: Wenn die Funktion nur mit Hilfe des Benutzers vollständig läuft oder sie zusammeninteragieren. Anstelle der <Funktionalität> wird hier <Akteur>**DIE MÖGLICHKEIT BIETEN + ZU** vor dem Prozesswort⁹⁵. Wenn der

⁹³ <https://www.youtube.com/watch?v=vr8s6jluq6I>

⁹⁴ Kluge, Roland: „Schablonen für alle Fälle“. 2016 Seite 13

⁹⁵ Ebd.

<Akteur> eindeutig ist, ist es besser den eindeutigen Namen zu schreiben. Z. B: Verkäufer, der Arzt usw.⁹⁶ Ergebnis:

Der Kassenautomat muss <<dem Verkäufer> MÖGLICHKEIT BIETEN> Bezahlung <ZU> berechnen.

- c) „Schnittstellenanforderung“: Falls die Funktion nur mit Hilfe von externen Systemen funktioniert. Anstelle <Funktionalität> wird hier <<FÄHIG SEIN> +ZU vor dem Prozesswort>> verwendet.

Ergebnis: *Der Kassenautomat muss<FÄHIG SEIN> Bezahlungsdaten <ZU> übertragen.*

Schritt 4: Objekte werden identifiziert. Wenn es ein Oberbegriff ist, müssen dafür ausführliche Anleitungen geschrieben werden. Objekte sind semantisch gesehen die Substantive und mit dem Prozesswort stark verbunden.

*Z. B: Der Kassenautomat muss dem Verkäufer die Möglichkeit bieten, **die Bezahlung** zu berechnen.*

Die Bezahlung ist hier das Objekt und das System musste dem Verkäufer eine Berechnungsfunktion anbieten.

FunktionsMASTeR ohne Bedingung wird wie oben beschrieben ausgefüllt. Um die Qualität von FunktionsMASTeR zu verbessern, werden diese Anforderungen auch mit Bedingungen angewendet. Diese Funktions-MASTeR ist in der nachfolgenden Abbildung dargestellt. Dazu ein Beispiel:

Sobald das Kassensystem den Kassenbestand gespeichert hat, sollte das Kassensystem dem Verkäufer die Möglichkeit bieten, den Tagesabschluss auszudrucken.

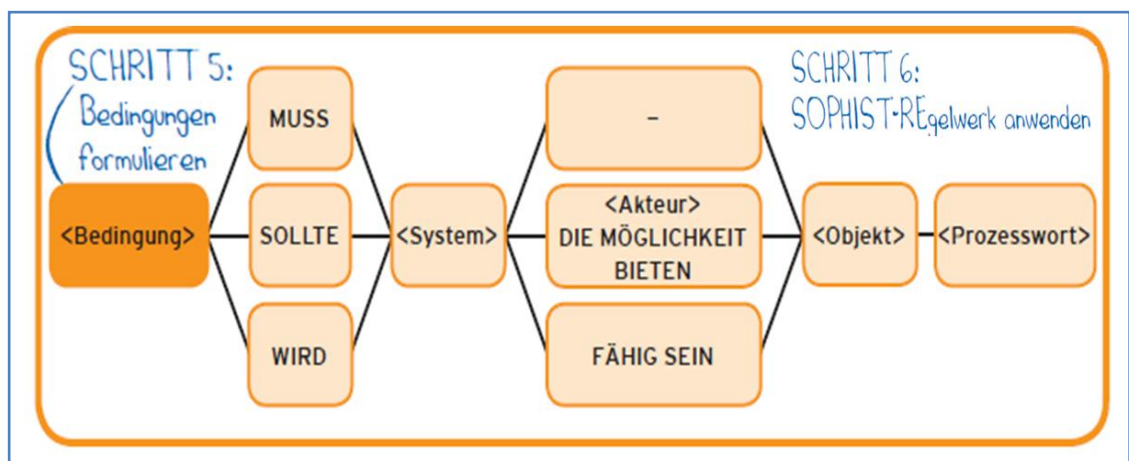


Abbildung 14. FunktionsMASTeR mit Bedingung⁹⁷

⁹⁶ Vgl. Rupp, Chris & die SOPHISTen: „Requirements-Engineeering und –Management, aus der Praxis von klassisch bis agil“. 2014, Seite 222

⁹⁷ Ebd. S. 224

Als fünfter Schritt wird gefragt, unter welcher Bedingung der Prozess ausgeführt wird und diese Bedingung wird im Satz vorgezogen. Obwohl nach dem fünften Schritt „**Bedingung formulieren**“ die funktionale Anforderung komplett und gut strukturiert ist, kann es einige semantische Fehler und sprachliche Effekte geben. Deshalb wird als nächster und letzter Schritt das SOPHIST-REgelwerk angewendet. Infolgedessen wird die Anforderungsqualität weiter verbessert und der „FunktionsMASTeR mit Bedingung“ ist fertiggestellt.

2.3.2. Detaillierter FunktionsMASTeR ohne Bedingung und mit Bedingung

Die einfachen FunktionsMASTeR mit und ohne Bedingung sind für die Benutzer, die neu mit der Anforderungserhebung anfangen, hilfreich. Für die Fortgeschrittenen, die die Anforderungen semantisch exakter und vollständiger erstellen wollen, gibt es „Detaillierter FunktionsMASTeR ohne Bedingung“ und „Detaillierter FunktionsMASTeR mit Bedingung“. Dabei werden nur die hinteren zwei Bestandteile der Anforderung detailliert analysiert.

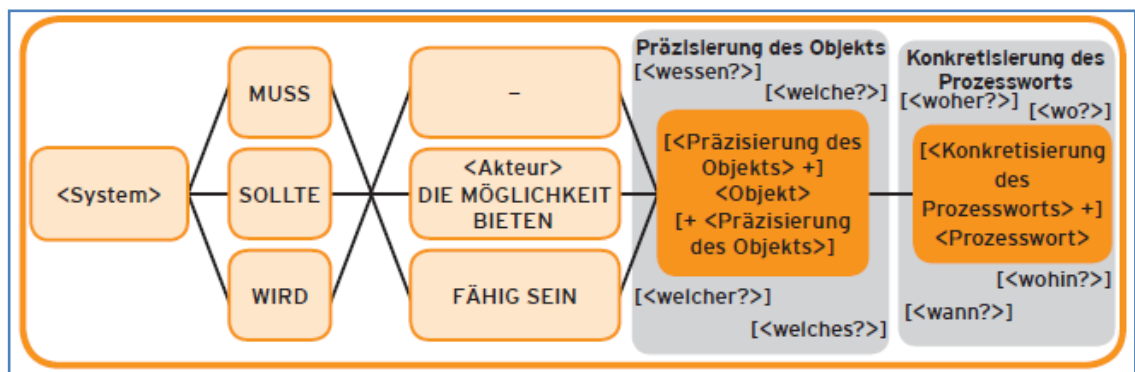


Abbildung 15. Detaillierter FunktionsMASTeR ohne Bedingung⁹⁸

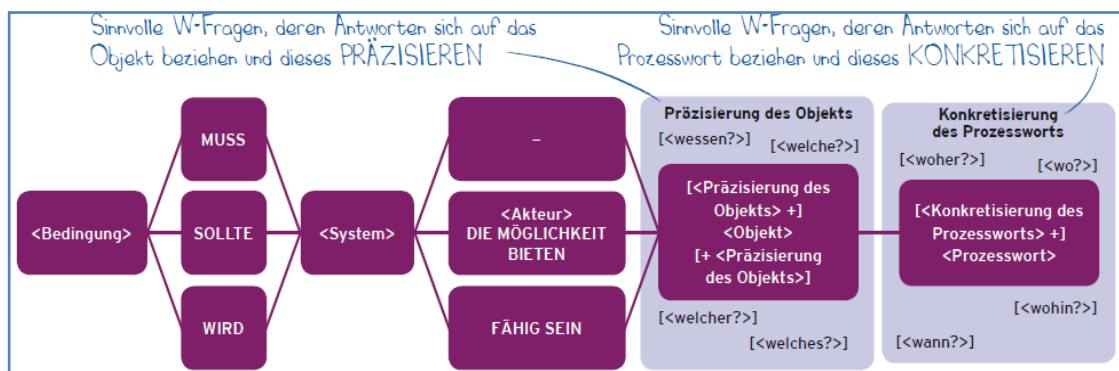


Abbildung 16. Detaillierter FunktionsMASTeR mit Bedingung⁹⁹

⁹⁸ Kluge, Roland: „Schablonen für alle Fälle“. 2016 S. 17

2.3.3. EigenschaftsMASTeR

Die EigenschaftsMASTeR ist die Schablone für die Erhebung von nicht funktionalen Anforderungen. Mit dieser Schablone können die Anforderungen für die Qualität, Technologie, die Benutzeroberfläche, die sonstigen Lieferbestandteile und die rechtlichen und vertraglichen Aktivitäten erhoben werden.

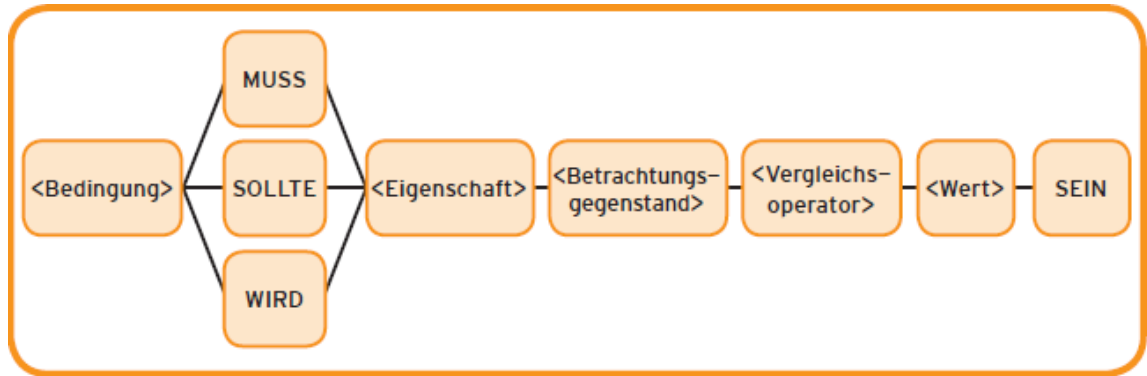


Abbildung 17. EigenschaftsMASTeR¹⁰⁰

Die Satzbestandteile <Bedingung> und <Verbindlichkeit> wurden schon oben erklärt. Aus diesem Grund gibt es nachfolgend einige kurze Erklärungen, was die folgenden Bestandteile bedeuten:

<Eigenschaft> ist ein Merkmal der Gegenstände und Funktionen wie z. B.: *Gewicht, Farbe, Größe eines Gegenstands, Geschwindigkeit, Dauer eines (Teil-)Systems*.

<Betrachtungsgegenstand> kann je nach dem Kontext und der fachlichen Bedeutung verschiedene Formen haben z. B.: ein Teilsystem oder eine Funktion. <Betrachtungsgegenstand> und <Eigenschaft> beziehen sich aufeinander und werden in dieser Art der Schablonen zusammengeschrieben.

<Vergleichsoperator> sind die Vergleichswörter wie z. B.: „wie“ oder „gleich“. Diese Wörter können nach Wunsch und Konstruktion weggelassen werden.

<Wert> ist ein bestimmter Zahlenwert mit verschiedenen Maßeinheiten oder standardisierten Wörtern oder Wortgruppen für Maße wie z. B.: 100 Gramm, weiß.

⁹⁹ Rupp, Chris & die SOPHISTen: „Requirements-Engineering und –Management, aus der Praxis von klassisch bis agil“. 2014, Seite 232

¹⁰⁰ Kluge, Roland: „Schablonen für alle Fälle“. 2016 S. 27

<sein> ist ein Vollverb, das am Ende des Satzes kommt. **Vorsicht:** statt „**sein**“ wird kein anderes Prozesswort geschrieben, da diese Schablone keine funktionalen Anforderungen bildet.

Hier ist ein Beispiel für diese MASTeR: *Die Breite des Eingabefensters muss gleich groß sein.*

Zusammengefasst können die mit dieser Schablone erstellten Anforderungen gut getestet und geprüft werden, weil in diesen Anforderungen die expliziten Werte benutzt werden.¹⁰¹

2.3.4. UmgebungsMASTeR

Diese Schablone ist geeignet zur Bildung von technologischen Anforderungen. Wie in der nachfolgenden Abbildung zu sehen ist, ist diese Schablone ähnlich zur EigenschaftsMASTeR.

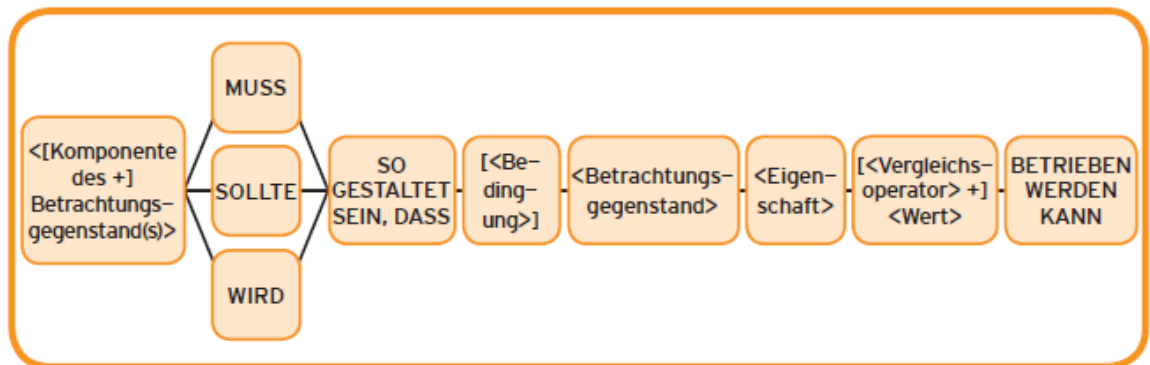


Abbildung 18. UmgebungsMASTeR¹⁰²

<[Komponente des +] Betrachtungsgegenstand(s)> ist eine dem **<Betrachtungsgegenstand>** zusätzlich hinzugefügte Formulierung, die die Anforderung noch exakter macht z. B.: *das Lasergerät des Kassensystems*. **[Komponente des +]** kann hingegen in einigen Sätzen weggelassen werden. Der Vorteil der Schablone ist, dass die Anforderungen wie die EigenschaftsMASTeR testbar und prüfbar werden.¹⁰³

2.3.5. ProzessMASTeR

ProzessMASTeR ist ebenfalls eine Schablone, die für nicht funktionale Anforderungen angewendet werden kann. Mit dieser Schablone werden die Anforderungen für „die durchzuführenden Tätigkeiten“ und die rechtlich-vertraglichen Anforderungen erstellt.

¹⁰¹ Rupp, Chris & die SOPHISTen: „Requirements-Engineering und –Management, aus der Praxis von klassisch bis agil“. 2014, Seite 235 ff.

¹⁰² Kluge, Roland: „Schablonen für alle Fälle“. 2016 S. 30

¹⁰³ Rupp, Chris & die SOPHISTen: „Requirements-Engineering und –Management, aus der Praxis von klassisch bis agil“. 2014, Seite 237 ff.

Dabei werden nicht die Systemfunktionalitäten, sondern die Gesetze und die Standards, was für die Entwicklung wichtig ist, festgelegt. In Abbildung 19 ist die ProzessMASTeR zu sehen.

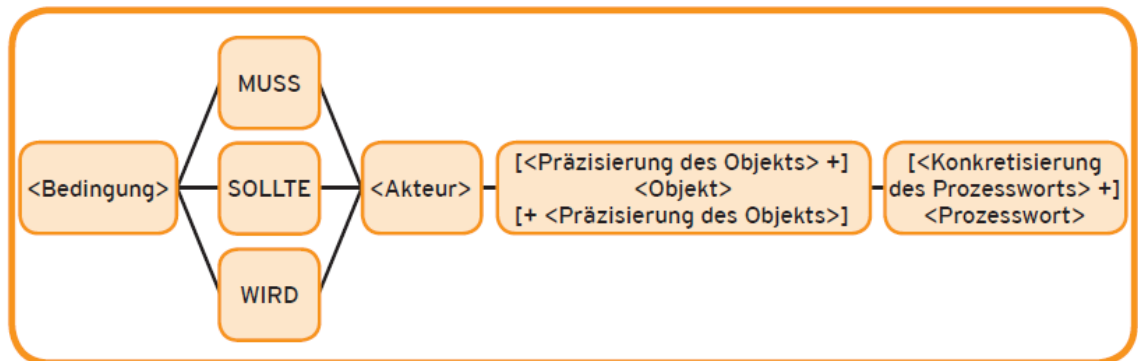


Abbildung 19. ProzessMASTeR¹⁰⁴

2.3.6. BedingungsMASTeR

Obwohl in diesem MASTeR nur einen Teil des Anforderungssatzes betrachtet wird, hilft es, den Nebensatz (die Bedingungen) richtig zu schreiben. In diesem Nebensatz werden die Wörter „falls“ für die logische Bedingung, sowie „sobald“ und „solange“ für zeitliche Bedingungen die Schlüsselwörter benutzt. Im Folgenden wird die BedingungsMASTeR-Schablone abgebildet.

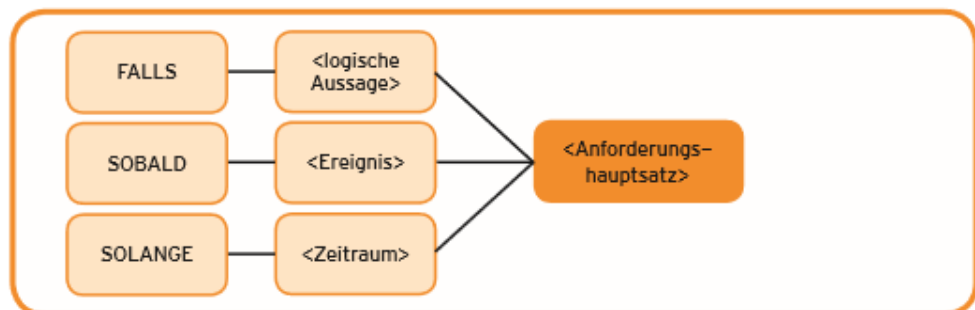


Abbildung 20. BedingungsMASTeR¹⁰⁵

¹⁰⁴ Kluge, Roland: „Schablonen für alle Fälle“. 2016 S. 34

¹⁰⁵ Ebd.

3. Vorausgesetzte Technologien

In diesem Kapitel werden die Technologien, die während dieser Arbeit benutzt werden, beschrieben. Diese Technologien helfen nicht nur bei der Implementierung, sondern auch beim Dokumentieren der Vorarbeit des aktuellen Stands und des Ergebnisses der Implementierungen.

3.1. Benötigte Sprachen

Folglich werden die für die Implementierung notwendigen Sprachen vorgestellt. Die drei wichtigen Sprachen HTML, CSS und JavaScript, die auf anderen Webseiten benutzt werden¹⁰⁶, sind auch die Implementierungstechnologien dieser Arbeit. Siehe die folgende Abbildung:



Abbildung 21. Frontend Hahn 2017, S. 30¹⁰⁷

3.1.1. HTML

HTML (Hypertext Markup Language) ist keine Programmiersprache, sondern eine Auszeichnungssprache, mit der die Struktur und Gestaltung von Webseiten ausgezeichnet werden können. Die Gestaltung einer Webseite wird durch HTML-Tags definiert. Jeder HTML-Tag hat seine Funktion und zeichnet einen Inhalt aus.¹⁰⁸

HTML-Dokumente bestehen aus zwei Teilen: Kopf und Körper. Im Kopf (head) werden die detaillierten Informationen der Seite zum Browser und Stile der Seite mitgeteilt. Beispiele hierfür sind Attribute wie: Titel, CSS Code. Im Körper werden die gesamten Inhalte des Dokuments mit anderen Tags geschrieben.¹⁰⁹

¹⁰⁶ Günster, Kai; „Schrödinger lernt HTML5, CSS3 & JavaScript“ 2017, S. 20

¹⁰⁷ Hahn, Martin; „Webdesign. Das Handbuch zur Webgestaltung“, 2017. S. 30

¹⁰⁸ Lubkowitz, Mark; „Webseiten programmieren und gestalten“, 2007, S. 109

¹⁰⁹ Robson, Elisabeth; Freeman, Eric; „HTML und CSS von Kopf bis Fuß“ 2012 S. 23 ff.

```

<!DOCTYPE html>
<html>
  <head>
    <title> Titel der Seite </title>
  </head>

  <body>
    <p> Inhalt der Seite </p>
  </body>
</html>

```

Abbildung 22. Grundgerüst einer HTML-Seite, eigene Darstellung nach Wiley & Sons, John „HTML & CSS Design and build Websites“ 2011, Indianapolis, Indiana, Seite 20

Wie in Abbildung 22 zu sehen ist, hat das Dokument Tags, die mit spitzen Klammern eingeschlossen sind. Mithilfe dieser Tags werden Inhalt und Gestaltung der Seite von Browsern gelesen. Nach dem Öffnen dieses Dokuments mit dem Browser Internet Explorer erscheint auf dem Bildschirm folgender Titel und Inhalt.

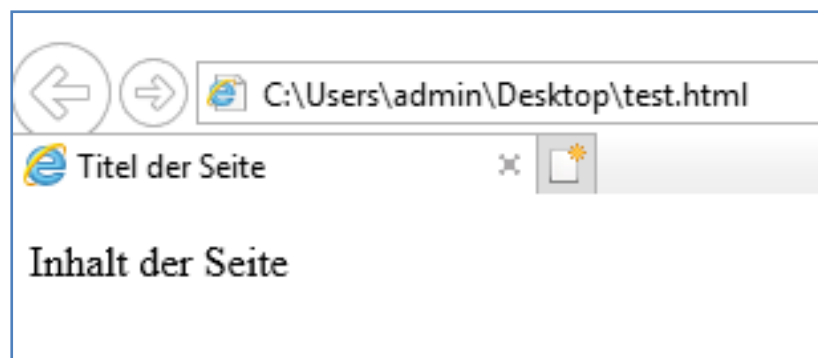


Abbildung 23. Anzeige der „test.html“ HTML-Seite im Browser, Ergebnis eigener Darstellung

3.1.2. CSS

CSS (Cascading Style Sheets) ist eine Stylesheet-Sprache, mit der ein oder mehrere Dokumente gestylt und präsentiert werden können. CSS beschreibt, wie Dokumentelemente auf dem Display dargestellt werden sollen.¹¹⁰

CSS bietet folgende Möglichkeiten an:

- mehr Flexibilität im Design, ohne grundlegende HTML-Konzepte zu verlieren.
- getrennte Gestaltung und Struktur als Grundlage für barrierefreie Websites.
- umfangreiche Formatierung der HTML-Elemente.
- Erzeugung von Animationen

¹¹⁰ Ebd.

- Einfluss auf die Eigenschaft von HTML wie z. B. Platzformatierung oder Schriftformatierung ¹¹¹

3.1.3. JavaScript

JavaScript ist die am häufigsten benutzte, flexible webbasierte Programmiersprache. Mit dieser Sprache können Back-End, Front-End und sogar Spiele entwickelt und noch andere Ziele in der Entwicklung erreicht werden. ¹¹²

Java und JavaScript dürfen nicht verwechselt werden, weil die Namen ähnlich aussehen, aber die Sprachen unterschiedlich sind. „Mit der Programmiersprache *Java* hat *JavaScript* nur einen Teil des Namens gemeinsam. Es sind aber völlig unterschiedliche Sprache.“ ¹¹³

JavaScript ist eine leicht erlernbare Sprache und wurde extra für das Web entwickelt, deshalb können die Elemente der Seiten und Nutzereingaben bearbeitet werden. Mithilfe von JavaScript sind die heutigen Webseiten so weit elaboriert, dass einige von ihnen funktionsfähiger und leistungsfähiger, als die auf dem PC installierte Software sind. ¹¹⁴ Noch ein Vorteil dieser Sprache ist, dass sie Funktionalitäten JSON (JavaScript Objekt Notation) und AJAX (Asynchronous JavaScript und XML) hat. ¹¹⁵

3.1.3.1. AJAX

Mit AJAX kann nicht die komplette Seite aufgerufen werden, sondern nur ein Teil oder eine Funktionalität der Seite. AJAX optimiert Request und Response zwischen Browser und Server. Damit entstehen folgende Vorteile:

- Die Ladegeschwindigkeit der Seite wird verbessert.
- Die Benutzereingaben werden nicht automatisch gelöscht.
- Request und Response zwischen Browser und Server sind für den Benutzer unauffällig. ¹¹⁶

¹¹¹ Laborenz, Kai; „CSS-das umfassende Handbuch“, 2011, S. 20.

¹¹² Vgl. <https://www.pluralsight.com/paths/javascript>

¹¹³ Jacobsen, Jens; Matthias Gidda; „Webseiten erstellen für Einsteiger“, 2016, S. 243

¹¹⁴ Ebd. S. 243

¹¹⁵ Günster, Kai; „Schrödinger lernt HTML5, CSS3 & JavaScript“ 2017, S. 708 ff.

¹¹⁶ Ebd. S. 708

3.1.3.2. JSON

JSON ist ein Format, welches für den Datenaustausch dient und einfach gelesen und geschrieben werden kann. Ein Rechner kann es einfach analysieren und generieren. Aus diesem Grund ist JSON eine benutzerfreundliche Datenaustauschsprache. JSON kommt in zwei unterschiedlichen Formen vor:

- Eine Sammlung der Objekte, des Datensatzes, der Struktur, des Dictionary, der Hash-Tabelle.
- Eine geordnete Liste der Werte von Array, des Vektors, der Liste oder Sequenz.

Ein Objekt als un-/geordnetes Element, Name, Variable werden zwischen zwei geschweiften Klammern „{ }“ geschrieben. Jedem Element folgt ein Doppelpunkt. Danach werden die dazugehörenden Werte geschrieben. Das nächste Element wird nach dem Komma einer neuen Zeile zugewiesen.¹¹⁷

Ein Beispiel für JSON ist in der nachfolgenden Abbildung dargestellt.

```
{ "employees": [  
  { "firstName": "John", "lastName": "Doe" },  
  { "firstName": "Anna", "lastName": "Smith" },  
  { "firstName": "Peter", "lastName": "Jones" }  
]}
```

Abbildung 24. Beispiel für JSON-Datei¹¹⁸

3.2. Webbrowser

Im Endeffekt soll das Ergebnis der Arbeit im Webbrowser aufgerufen und gesehen werden können. Um den aktuellen Status zu sehen, werden die Implementierungen während der Arbeit immer mit Browsern getestet. Ein Webbrowser ist eine Software, mit der die Webseiten in WWW richtig angezeigt werden können. Da es aber viele verschiedene Browser gibt, versteht jeder Browser den gleichen Code anders. Dabei kann es sein, dass die Seiten anders angezeigt werden.¹¹⁹ Um diesen Fall zu verhindern, werden in dieser Arbeit nur wichtige, populäre Webbrowser wie „Google Chrome“, „Mozilla Firefox“, „Microsoft Internet Explorer“, „Safari“, oder „Microsoft Edge“ benutzt.

3.3. Client-side Component Model (CCM)

CCM ist ein Modell, mit dem die Webkomponenten im Internet veröffentlicht werden

¹¹⁷ <https://www.json.org/>

¹¹⁸ https://www.w3schools.com/js/js_json_xml.asp

¹¹⁹ https://praxistipps.chip.de/was-ist-ein-browser-einfach-erklart_41369

können. CCM hat zwei Hauptbestandteile: CCM-Framework und CCM-Komponenten und noch dazu zwei Standardmerkmale der W3C-Komponenten: Shadow DOM und Custom Elements.

Der erste Bestandteil von CCM ist ein CCM-Framework. Das ist eine freie Software und nötig für die Einbettung von Komponenten in beliebig vielen, verschiedenen Webseiten und Datenverwaltung dieser Seiten. Obwohl sie mit diesen zwei Diensten große Leistung hat, besteht CCM-Framework nur aus einer kleinen JavaScript-Datei und unter der MIT-Lizenz im Web herunterladbar.

Der zweite Bestandteil von CCM, CCM-Komponente ist einer freie Software, der mit HTML, CSS, JavaScript entwickelt und als JavaScript-Datei veröffentlicht wurde. Eine Komponente besteht aus einem oder mehreren CCM-Komponenten und kann in jeder Webseite integriert werden. Diese Komponente kann entweder mit CCM-Framework oder mit anderen JavaScript-Frameworks funktionieren.¹²⁰

3.4. Balsamiq Mockups

Wireframing hilft einen Webseitendienst strukturiert zu gestalten und Inhalte und Funktionalität auf einer Seite darzustellen.¹²¹ Es gibt viele Tools zur Bildung von Wireframes, wie Justinmind, Wireframe.cc, Balsamiq Mockups.¹²² In dieser Arbeit wird Balsamiq benutzt, da es ein schnelles Wireframing-Tool ist. Das Tool steht unter folgendem Link zum kostenlosen Herunterladen bereit:

<https://balsamiq.com/wireframes/desktop/#>. Bei der Erstellung des Mockups (Modell) einer Webseite wird die existierende Skizzierung der Webseite von den Notizen oder Tafeln auf einen Computer übertragen. Während der Übertragung sollte der Fokus auf dem Aufbau und dem Inhalt der Seite liegen. Als Mangel ist jedoch zu nennen, dass die Werkzeuge des Tools für Farben- und Details-Formatierung noch nicht verfügbar sind.¹²³

3.5. WebStorm

In dieser Arbeit wird eine IDE (integrated development environment - integrierte Entwicklungsumgebung) „WebStorm“ verwendet, die von der Firma JetBrains entwickelt wurde. Diese IDE bietet die Möglichkeit, mit JavaScript, Node.js, HTML und CSS smart zu arbeiten. Ihre wichtigen, hilfreichen Eigenschaften für die Entwickler sind u. a. Codevervollständigung, leistungsstarke Navigationsfunktionen, sowie rasche Fehlerer-

¹²⁰ <https://github.com/CCMjs/CCM>

¹²¹ <https://www.experienceux.co.uk/faqs/what-is-wireframing/>

¹²² <https://www.creativebloq.com/wireframes/top-wireframing-tools-11121302>.

¹²³ <https://balsamiq.com/wireframes/>

kennung.¹²⁴

3.6. GitHub

GitHub ist ein Onlinedienst zur Speicherung bzw. Entwicklung mehrerer Software-Projekte, die auf den Servern bereitgestellt werden. Dadurch haben Entwickler die Möglichkeit, gleichzeitig sowie separat an den Projekten zuarbeiten. GitHub ist auch für beliebige Open-Source-Projekte geeignet, sodass die Software für jeden Entwickler frei zugänglich ist. Dabei werden die Eingaben immer auf Kompatibilität geprüft, um eventuelle Probleme bzw. Fehler zu vermeiden. Die grafische Darstellung bietet die Möglichkeit, die Aktionen mit wenigen Klicks auszuführen. Das erleichtert den Einstieg ins Projekt und spart somit den Entwicklern viel Zeit.¹²⁵

Da DMS und die anderen Komponenten auf GitHub zur Verfügung stehen und dieser Onlinedienst die oben erwähnten Vorteile hat, wird GitHub in dieser Arbeit verwendet, um die fertigen Endcodes zu veröffentlichen. Die bisher vorgestellten Technologien werden in folgendem Kapitel benutzt, damit die Komponente konzipiert und implementiert wird.

¹²⁴ <https://www.jetbrains.com/webstorm/features/>

¹²⁵ <https://www.dev-insider.de/was-ist-github-a-645831/>

4. Konzeption und Implementierung

Dieses Kapitel unterteilt sich in zwei Sinnabschnitte. Im ersten Teil wird geplant, wie die Komponente in dieser Arbeit aufgebaut wird und aussehen soll. Im zweiten Teil werden beschrieben, wie der Komponente implementiert wird.

4.1. Konzeption

Das Hauptziel der Arbeit ist die Anforderungserhebungskomponente mit der Theorie der Sophisten GmbH zu implementieren. Um diese Implementierung durchzuführen, wird zunächst Konzept erarbeitet. Im Folgenden werden die Konzeption für die Schablonen in verschiedenen Sprachen und die Anforderungen, die während der Implementierung nötig sind, vorgestellt.

4.1.1. Realisierungsüberlegungen

Durch das Ergebnis der Arbeit soll die Erstellung der Anforderungen erleichtert werden. Dafür werden die zusammenfassenden Informationen der Schablone, die im Kapitel 2 vorgestellt und analysiert wurden, als Erklärung für die Komponente veröffentlicht. Da es fachlich nicht richtig ist, die Software ohne Anforderungen zu entwickeln, werden zuerst die Anforderungen, die für die Entwicklung dieser Komponente wichtig sind, festgelegt. Die schriftlichen Anforderungen können aber das Design des Projekts nicht beschreiben. Aus diesem Grund wird das Aussehen der Komponente mit einem Mockup beschrieben. Nachdem das Mockup und die Anforderungen vorbereitet sind, wird die Komponente in drei Sprachen Deutsch, Englisch und Usbekisch mit den im Kapitel 3 vorgestellten Technologien entwickelt.

4.1.1.1. Schablonen in deutscher Sprache

In deutscher Sprache werden die Schablonen, die in Kapitel 2.2 und 2.3 vorgestellt und erklärt wurden, entwickelt. Einige MASTeR-Schablonen wurden von den Autoren erwähnt, aber ihre Abbildung oder strukturelle Darstellung kann in der Öffentlichkeit nicht gesehen werden. Deshalb werden die Schablonen folgenderweise strukturiert:

- FunktionsMASTeR ohne Bedingung: System + Verbindlichkeit + Funktionalität + Objekt + Prozesswort. Siehe Kapitel 2.
- FunktionsMASTeR mit Bedingung: Bedingung + Verbindlichkeit + System + Funktionalität + Objekt + Prozesswort. Siehe Kapitel 2.

In diesem Fall soll für jeden Satzbestandteil ein Text-Eingabefeld oder Selectfeld implementiert werden.

- Detaillierter FunktionsMASTeR ohne Bedingung: In Kapitel 2 wurde dieser so dargestellt: System + Verbindlichkeit + Funktionalität + (Präzisierung des Ob-

jekts + Objekt + Präzisierung des Objekts) + (Konkretisierung des Prozessworts + Prozesswort). In diesem Fall entsteht das Problem, dass die Eingabefelder beim Schreiben unübersichtlich werden. Es kann vorgestellt werden, dass manche User ohne Präzisierung oder ohne Konkretisierungen schreiben werden. In diesem Fall können die Namen der Satzbestandteile nach der Erstellung der Sätze nicht erkennbar sein. Deshalb werden die Details eine getrennte Struktur haben. System + Verbindlichkeit + Funktionalität + Präzisierung 1 des Objekts + Objekt + Präzisierung 2 des Objekts + Konkretisierung des Prozessworts + Prozesswort.

- Detaillierter FunktionsMASTeR mit Bedingung: Die Struktur wird auch wie bei „Detaillierter FunktionsMASTeR ohne Bedingung“ gebastelt. Bedingung + Verbindlichkeit + System + Funktionalität + Präzisierung 1 des Objekts + Objekt + Präzisierung 2 des Objekts + Konkretisierung des Prozessworts + Prozesswort.
- EigenschaftsMASTeR ohne Bedingung: Die Struktur-Abbildung der Schablone gibt es in der Öffentlichkeit nicht, aber Kluge hat einige Sätze dafür vorgestellt¹²⁶. Nach diesen Sätzen sieht die Struktur der Schablone wie folgt aus: Eigenschaft + Betrachtungsgegenstand + Verbindlichkeit + Vergleichsoperator + Wert + SEIN.
- EigenschaftsMASTeR mit Bedingung: Bedingung + Verbindlichkeit + Eigenschaft + Betrachtungsgegenstand + Vergleichsoperator + Wert + SEIN.
- UmgebungsMASTeR: Nach der Schablonen-Abbildung, die im Kapitel 2 vorgestellt wurde, kann der Satz nicht richtig gebaut werden, weil der Bedingungsnebensatz nach der deutschen Grammatik falsch geordnet ist.¹²⁷ Es musste so ((Komponente des) Betrachtungsgegenstand(s)) + Verbindlichkeit + (SO GESTALTET SEIN, DASS) + Bedingung + Betrachtungsgegenstand + Eigenschaft + ((Vergleichsoperator) + Wert) + (BETRIEBEN WERDEN KANN) gestaltet sein. Es ist aber besser und nutzerfreundlicher, wenn statt des Bedingungsnebensatzes eine Bedingung in einer nominalisierten Form mit dem „Betrachtungsgegenstand“ zusammen geschrieben werden z. B.: ((Bedingung)+ Betrachtungsgegenstand).
- ProzessMASTeR ohne Bedingung: Akteur + Verbindlichkeit + Präzisierung 1 des Objekts + Objekt + Präzisierung 2 des Objekts + Konkretisierung des Prozessworts + Prozesswort.
- ProzessMASTeR mit Bedingung: Bedingung + Verbindlichkeit + Akteur + Präzisierung 1 des Objekts + Objekt + Präzisierung 2 des Objekts + Konkretisierung

¹²⁶ Kluge, Roland: „Schablonen für alle Fälle“. 2016 S. 29

¹²⁷ <https://mein-deutschbuch.de/nebensaetze.html>

des Prozessworts +Prozesswort.

BedingungsMASTeR: Wie die Bedingungen geschrieben werden, ist nicht streng nach Struktur begrenzt, aber die Signalwörter sollen benutzt und inhaltlich korrekt sein. Für jeden BedingungsMASTeR wird eine Schablone erstellt.

- Logische Aussage: FALLS + logische Aussage, + Anforderungshauptsatz.
- Ereignis: SOBALD + Ereignis, + Anforderungshauptsatz.
- Zeitraum: SOLANGE + Zeitraum, + Anforderungshauptsatz.

In diesen Schablonen gehören nur die ersten zwei Fenster zum BedingungsMASTeR. Das dritte Fenster ist dafür da, damit die User sich orientieren können.

4.1.1.2. Schablonen in englischer Sprache

In englischer Sprache wurden die MASTeR-Schablonen veröffentlicht, aber Satzreihen einiger Schablonen passen nicht zu der englischen Grammatik¹²⁸, deshalb werden einige Veränderungen während der Entwicklung vorgenommen.

Der bisherige detaillierter FunktionsMASTeR:

System + SHALL + provide (actor) withtheabilityto + processverb (details on processverb) +(details on object) object (details on object).¹²⁹

Die Wörter, die in Klammer sind, können nicht benutzt werden. Falls sie benutzt werden, soll diese Reihe festgehalten werden. Um die Bedeutung der Schablone zu verbessern und die englische Grammatik nicht zu verletzen, werden folgende Satzbestandteile verändert.

- Wenn Akteur benutzt wird, soll so „provide (actorwith) theabilityto“ geschrieben werden.
- Es gibt im Englischen das Wort „directobject“, welches sofort nach dem Verb geschrieben wird und „indirectobject“, dass nach dem „directobject“ geschrieben wird.„(details on processverb) + (details on object) object (details on object)“ werden durch die direkten und indirekten Objekte ersetzt.

Das Ergebnis lautet wie folgt:

System + SHALL + provide (actorwith) theabilityto + processverb + directobject + indirectobject.

4.1.1.3. Schablonen in usbekischer Sprache

In usbekischer Sprache gibt es im Moment überhaupt keine Schablone, die für die Er-

¹²⁸ James Chamberlain, „Word order in English“ 2019 and Murphy R. English Grammar in Use. 2012

¹²⁹ Requirements-Engineering und –Management, aus der Praxis von klassisch bis agil S. 234

hebung der Anforderungen in SE benutzt werden können. Deshalb wird versucht, einige Schablonen in usbekischer Sprache zu konzipieren. Diese werden sich an den MASTeR-Schablonen orientieren, obwohl die Satzbestandteile in usbekischer Sprache anders als in deutscher Sprache sind¹³⁰. Im Folgenden wird ein deutscher Beispielsatz, der mit der MASTeR-Schablone erstellt wurde, gegeben und mit der usbekischen Satz-Struktur verglichen:

Deutsch: Das System muss dem Verkäufer die Möglichkeit bieten, die Bezahlung zu berechnen. Usbekisch: Tizim sotuvchiga hisobni hisoblash imkonini berishi kerak. Dieses Beispiel zeigt: In usbekischen Satz wird kein Komma benutzt, wenn ein Satz mehrere Vollverben enthält. Es muss als nächstes die syntaktischen Möglichkeiten geprüft, ob dieser Satz auch ins Usbekische umgewandelt wird und als usbekische Schablone benutzt werden kann.

Usbekischer Satz: Tizim + sotuvchiga + hisobni + hisoblash + imkonini berishi + kerak. Erklärung zum usbekischen Aufbau des Satzes: Das System + der Akteur + Objekt + Prozesswort + die Möglichkeit bieten + Verbindlichkeit. Das zeigt, dass die Satzbestandteile der MASTeR-Schablone in Usbekisch zum Teil benutzt werden können, wenn Verbindlichkeit „MUSS“ und „SOLLTE“ benutzt wird.

Die Verbindlichkeitsarten „KANN“ und „WIRD“ können aber nicht als ein Wort ersetzt werden. Dafür wurden keine passenden Wörter in der usbekischen Sprache¹³¹ gefunden. Stattdessen gibt es Suffixe, die an jeweiligen Wörtern angehängt werden und ihre Bedeutung im Satzzusammenhang erhalten, z.B.: *Das System kann dem Verkäufer die Möglichkeit bieten, die Bezahlung zu berechnen.*

Tizim + sotuvchiga + hisobni + hisoblash + imkonini ber(aoladi).

Das System + der Akteur + Objekt + Prozesswort + die Möglichkeit bieten(KANN).
und

Tizim + sotuvchiga + hisobni + hisoblash + imkonini ber(moqchi).

Das System + der Akteur + Objekt + Prozesswort + die Möglichkeit bieten(WIRD).

Um dieses Problem zu lösen, wäre es eine Möglichkeit, ein benachbartes Wort zu finden und (KANN und WIRD) zu verallgemeinern. Die alten Versionen der MASTeR-Schablonen hatten nur drei Verbindlichkeiten und konnten trotzdem in der Praxis gut verwendet werden. Deshalb kann ein Wort diese beiden Wörter auf Usbekisch verallgemeinern. Dieses Wort ist „MUMKIN“ z. B.:

Das System kann dem Verkäufer die Möglichkeit bieten, die Bezahlung zu berechnen.

Das System wird dem Verkäufer die Möglichkeit bieten, die Bezahlung zu berechnen.

¹³⁰ A. Nurimonov, N. Mahmudov, A. Ahmedov, S. Solixujayeva „O'zbektiliningmazmuniysintaksisi“ S. 50 ff. 1992

¹³¹ A. Madvaliyev, „O'zbektiliningizohlilug'ati“ 2006, Tashkent

Tizim + sotuvchiga + hisobni + hisoblash + imkonini berishi + MUMKIN.

Damit ist die erste usbekische Schablone mit drei Verbindlichkeiten fertig.

4.1.2. Anforderungen

Hier sind die Anforderungen, die umgesetzt werden sollen:

ID: REQ-F-01
Die Sophist-Komponente muss dem Benutzer die Möglichkeit bieten, die Erklärungen der Schablonen zu lesen.
Akzeptanzkriterien: Die App enthält genügend Informationen, die die Schablonen erklären.

Die Anforderung REQ-F-01 ist für den Anfang sehr wichtig, damit die Benutzer als Erstes das Wissen darüber haben, was das Sophist-Regelwerk und Schablonen sind. Dann können die Schablonen besser benutzt werden.

ID: REQ-F-02
Die Schablone muss dem Benutzer die Möglichkeit bieten, die Anforderungen zu schreiben.
Akzeptanzkriterien: Die Schablonen haben Eingabefelder, in die die Bestandteile der Anforderungen eingegeben werden können.

Die Anforderung REQ-F-02 besagt, dass die Schablonenbestimmte Eingabefenster haben und diese Fenster sollen dem User erleichtern, die Anforderungen schneller als bisher und nach Schablonen zu schreiben.

ID: REQ-F-03
Falls die User die Satzbestandteile fehlerhaft schreiben, muss die Komponente dem User die Möglichkeit bieten, die Fehler zu korrigieren.
Akzeptanzkriterien: Die fehlerhaft geschriebenen Anforderungen müssen bearbeitet werden können.

Es ist zu erwarten, dass die User beim Schreiben Fehler machen und diese korrigieren wollen. Mithilfe des „edit“-Button können die gewünschten Anforderungen bearbeitet und wieder mit dem „add“-Button gespeichert werden.

ID: REQ-F-04
Falls die User die Anforderungen redundant schreiben, muss die Komponente dem User die Möglichkeit bieten, die redundanten Anforderungen zu löschen.
Akzeptanzkriterien: Ein „delete“-Button muss zur Verfügung stehen. Beim Drücken des Buttons soll die gewünschte Anforderung gelöscht werden.

Diese Anforderung hilft dabei, dass die User beliebig viele oder alle Anforderungen löschen können. Wenn das komplette Dokument falsch geschrieben ist, kann das Dokument entleert und erneut angefangen werden.

ID: REQ-F-05
Falls User die Komponente auf einer anderen Website benutzen möchten, muss die Komponente dem User die Möglichkeit bieten, einen Einbettungslink zu erhalten.
Akzeptanzkriterien: Der Einbettungslink der Komponente wird erstellt und der erstellte Link kann in der anderen Website eingebettet werden.

Diese Anforderung hilft dem User von den beliebigen gewünschten MASTeR-Schablonen Einbettungscodes zu erstellen und sie mit anderen zu teilen oder auf einer anderen Website einzubetten.

4.1.3. Mockup

Im Folgenden werden die Mockupdarstellungen, die mit dem Tool Balsamiq und Snipping Tool von Windows erstellt wurden, vorgestellt.

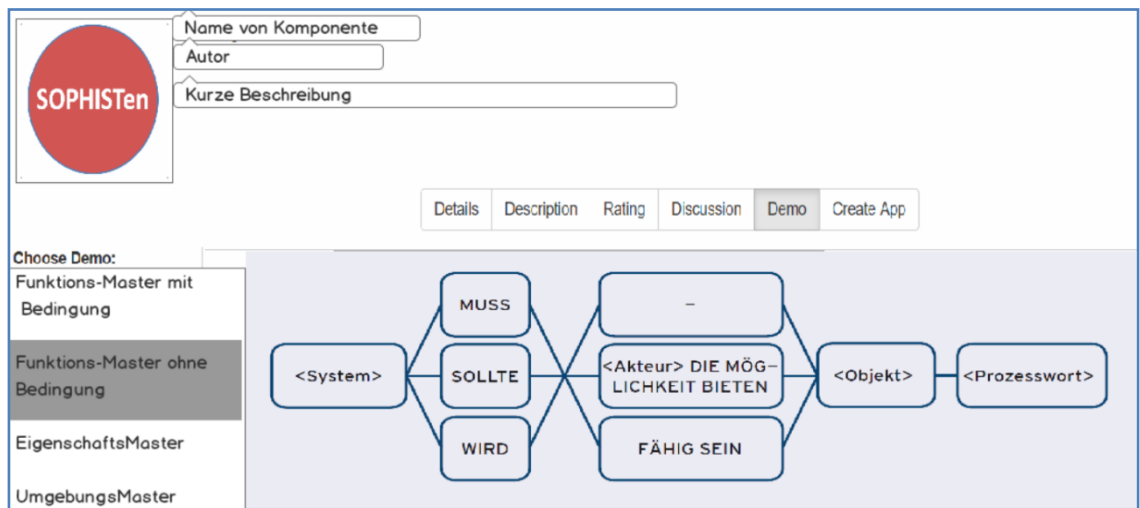


Abbildung 25. Mockup für Demofenster der Komponente, eigene Darstellung

Die Abbildung 25 zeigt die allgemeine Demoseite für alle Schablonen. Auf dieser Seite können bestimmte Arten der Schablonen ausgewählt und betrachtet werden. Anschließend soll die Seite, die nach dem Drücken des Buttons „Create App“ erscheint, dargestellt werden (s. Abb. 26).

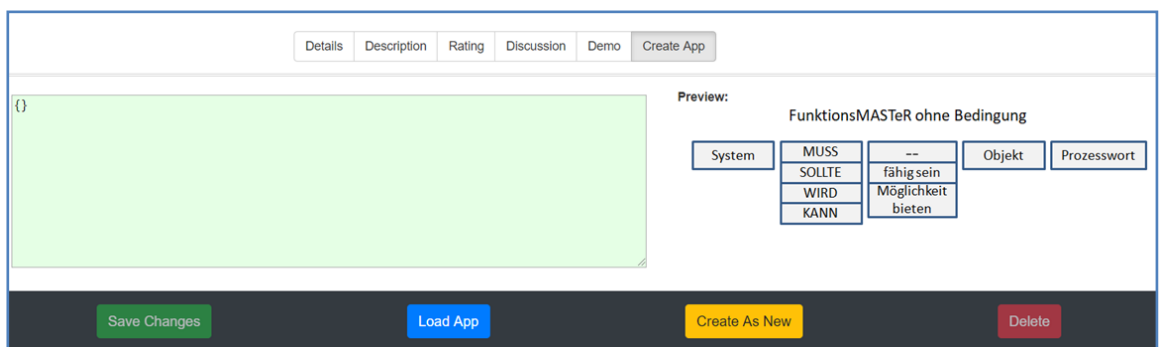


Abbildung 26. Mockup für die "Create APP", eigene Darstellung

Nach der Auswahl einer Schablone und „create APP“ werden für diese Art der Schablonen die verschiedenen Funktionalitäten gezeigt. Je nach Wunsch und Situation können für diese Schablonen die einbettbaren Links erstellt, gelöscht, geändert oder geladen werden. Das ist in der Abbildung 26 zu sehen.

So werden alle anderen Schablonen erstellt. Wie diese Schablonen aussehen sollen, wird im Folgenden gezeigt:

FunktionsMASTeR ohne Bedingung					
Systemname	Verbindlichkeit	Funktionalität	Objekt	Prozesswort	Buttons
System	MUSS	fähig sein	die Daten	zu speichern	<input type="button" value="edit"/> <input type="button" value="save"/> <input type="button" value="delete"/>
<input type="text" value="System"/>	<input type="text" value="MUSS"/> <input type="text" value="SOLLTE"/> <input type="text" value="WIRD"/> <input type="text" value="KANN"/>	<input type="text" value="--"/> <input type="text" value="fähig sein"/> <input type="text" value="Möglichkeit bieten"/>	<input type="text" value="Objekt"/>	<input type="text" value="Prozesswort"/>	<input type="button" value="add"/>

Abbildung 27. Mockup für FunktionsMASTeR ohne Bedingung, eigene Darstellung.

In der Abbildung 27 ist ein Mockup von der Schablone „FunktionsMASTER ohne Bedingung“ zu sehen. Wenn jede Schablone im WWW aufgerufen wird, erscheint oben der Name, damit den Usern verständlich wird, welche Schablone sie aktuell benutzen und welche Art der Anforderungen sie aktuell schreiben. In der nächsten Reihe stehen die Satzbestandteile, aus denen die aktuelle Schablone besteht. Am Ende der Reihe sind Buttons, die für den Satz einige Funktionalitäten liefern. Die Schrift ist fett und nicht änderbar festgelegt. Diese Einstellung wurde so vorgenommen, damit sich die User besser orientieren können. Anschließend wird eine Musteranforderung, die nach der Schablone erstellt wurde, vorgestellt. Sie dient als erleichterte Anweisung für die User, die noch keine Schablone benutzt haben. Dieser Satz kann mit dem „delete“ Button gelöscht oder mit dem „edit“ Button bearbeitet werden. Jede erstellte Anforderung hat diese drei Buttons, die mit dem Löschen der Anforderung gleichzeitig gelöscht werden. Ganz unten sind Eingabefelder und Selectfelder, die für die Erstellung der Schablone nötig sind. Nach dem Ausfüllen dieser Felder wird der „add“ Button gedrückt und damit wird die neue Anforderung mit den Hilfsbuttons „edit“, „delete“ und „save“ hinzugefügt. Als Zweck für die Erleichterung und Zeitersparnis werden die Felder „System“, „Verbindlichkeit“ und „Funktionalität“ für die Eingabe der nächsten Anforderung nicht entleert, weil diese Satzteile nicht so viele Optionen haben und mit großer Wahrscheinlichkeit wieder benutzt werden.

4.2. Implementierung

In diesem Kapitel wird zunächst die Implementierung vorbereitet und die verschiedenen Schablonen mithilfe von den oben genannten Technologien implementiert.

4.2.1. Vorbereitung der Implementierung

Als Vorbereitung für die Implementierung wurde die IDE WebStorm von der Anbieterseite <https://www.jetbrains.com/webstorm/> heruntergeladen und installiert. Da die Implementierung die Werkzeuge Java Script, CSS, CCM Framework Wissen und Erfahrung verlangt, mussten zuerst die Werkzeuge eingearbeitet und gelernt werden. In die-

sem Fall soll erwähnt werden, dass CCM Framework nicht zeit- und platzumstündlich war.

4.2.2. Implementierung der MASTeR-Schablonen

Die Schablonen werden Schritt für Schritt implementiert. Dabei wird den bereits erstellten Anforderungen und Mockups gefolgt. Die erste Schablone dient als Muster für die nächsten Schablonen. Dafür wurde die einfachste Schablone „FunktionsMASTeR ohne Bedingung“ zur Entwicklung ausgesucht. Die Entwicklung wird am Anfang mit HTML-Code umgesetzt. Dabei werden die Eingabefenster mit `<input>` und `<select>` Elementen entwickelt. Die Abbildung 28 zeigt, wie ein Text-Eingabefenster mit dem placeholder `<System>` und id „new_system“ in HTML geschrieben wird.

```
<input type="text" id="new_system" placeholder="System">
```

Abbildung 28. Inputfeld in HTML, eigene Darstellung

Abbildung 29 zeigt, was der Tag von dem Bild 28 im Browser ausgibt und darstellt. Die Darstellung ist schon mit CSS gestylt.

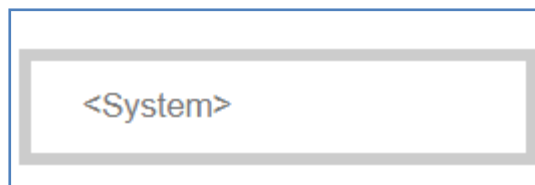


Abbildung 29. Inputfeld im Browser, eigene Darstellung

Da die Schablonen die erleichterte Bedienung und Zeitersparnis erbringen sollen, sollen einige Fenster mit dem `<select>` Tag geschrieben werden. In diesem Beispiel stehen nur vier Wortmöglichkeiten zur Auswahl. Diese sind die Arten der Verbindlichkeit. Wenn das fünfte Wort geschrieben wird, ist es falsch. Deshalb wurde der Vorteil des Selectfeldes benutzt, um das Schreiben der anderen Wortmöglichkeiten zu begrenzen und nur vier Verbindlichkeitsarten festzulegen.

```
<select id="new_modal">
  <option>MUSS</option>
  <option>SOLLTE</option>
  <option>WIRD</option>
  <option>KANN</option>
</select>
```

Abbildung 30. Optionen für die Verbindlichkeit, eigene Darstellung

Im Folgenden ist zu sehen wie das mit CSS gestylte Selectfeld im Browser angezeigt wird.

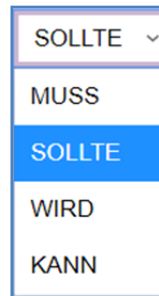


Abbildung 31. Darstellen der Verbindlichkeit, eigene Darstellung

Insgesamt werden fünf Eingabefenster der Schablone „FunktionsMASTeR ohne Bedingung“ entwickelt. Jedes Eingabefeld ist ein Bestandteil des Satzes und nach der Eingabe der Texte in Eingabefelder soll der fertige Satz bestätigt und unter den anderen Sätzen gelistet werden. Dafür wird ein anklickbarer Button, der eine Funktion auslösen kann, erzeugt. Es gibt in HTML ein Button-Element, das in der nachstehenden Abbildung dargestellt ist.

```
<button class="submit" onclick="myFunction()"> Submit</button>
```

Abbildung 32. Button-Element

„Submit“ ist die Beschriftung des Buttons, die auf dem Desktop angezeigt wird. Der Class-Name des Buttons ist „submit“. Er wird beim CSS-Design gebraucht. Damit ist die Struktur der ersten Schablone fertig (s.Abb.33).

Funktionsmaster ohne Bedingung

<System>

MUSS

die Möglichkeit bieten

<Objekt>

<Prozesswort>

Submit

Abbildung 33. Darstellen einer Schablone, eigene Darstellung

Wenn die Eingabefelder ausgefüllt sind und der „Submit“ Button angeklickt wird, wird die Funktion „myFunktion()“ aufgerufen und ein Schablonen-Satz erstellt. Man kann den Vorgang beliebig oft wiederholen und beliebig viele Sätze erstellen.

Diese Schablone muss als Ergebnis der Arbeit in DMS veröffentlicht und benutzt werden. Dafür muss die HTML Struktur in JSON umgewandelt und einige technische Eigenschaften so angepasst werden, dass dadurch die Komponente konfigurationsfähig wird. Eine in DMS funktionsfähige Komponente (in diesem Fall „SOPHISTen“) besteht aus folgenden Dateien, die im WebStorm erstellt wurden, siehe Abbildung 34.

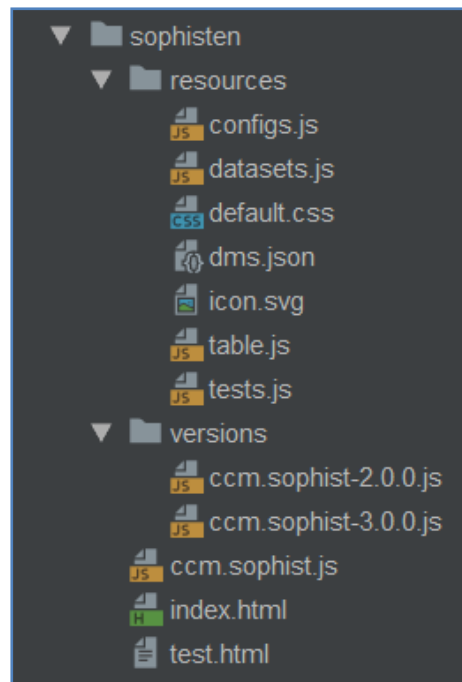


Abbildung 34. die Liste der Sophist-Komponenten-Dateien, eigene Darstellung

Im Folgenden werden diese Dateien erklärt und vorgestellt.

config.js – in dieser Datei werden statische Konfigurationen erstellt. `CCM.files['configs.js'] = {}` stellt die file „configs.js“ und die darin enthaltenen Konfigurationen dar. Hier wurde eine Konfiguration „FunktionsMASTeR mit Bedingung“ geschrieben.

```
ccm.files[ 'configs.js' ] = {
  //FunktionsMASTeR mit Bedingung
  "FunktionsMASTeR_mB": {
    "html.main.inner.0.inner": "FunktionsMASTeR mit Bedingung",
    key: "FunktionsMASTeR_mB",
    headers: [ "Bedingung", "Verbindlichkeit", "Systemname", "Funktionalität", "Objekt", "Prozesswort", "Buttons" ],
    columns: [ "bedingung", "modal", "system", "func", "object", "process" ],
    initial_values: {
      bedingung: "Bedingung",
      system: "das System",
      object: " das Objekt",
      process: " das Prozesswort"
    },
    data: {
      "store": [ "ccm.store", 'https://turaboev.github.io/sophiste/resources/datasets.js' ],
      "key": "FunktionsMASTeR_mB"
    }
  },
},
```

Abbildung 35. Ausschnitt von config.js, eigene Darstellung

datasets.js - in dieser Datei werden die Daten, die während des Aufrufs dieser Art der Schablone angezeigt werden, für die Schablonen gesetzt. In der nachfolgenden Abbildung werden die Daten für die Schablone „Demo“ gesetzt.

```

ccm.files[ 'datasets.js' ] = {
  //Demo oder FunktionsMASTeR ohne Bedingung
  "demo": {
    "key": "demo",
    "rows": [
      {
        system: "Das System",
        modal: "MUSS",
        func: "die Möglichkeit bieten",
        object: "die Bezahlung",
        process: "zu berechnen"
      },
      {
        system: "Das Kassensystem",
        modal: "SOLLTE",
        func: "fähig sein",
        object: "die Daten",
        process: "zu speichern"
      }
    ]
  }
},

```

Abbildung 36. Ausschnitt von datasets.js, eigene Darstellung

default.css - diese Datei beschreibt, wie die Schablonen aussehen sollen. Die Farbe und Struktur wurden ähnlich der typischen MASTeR-Schablonen implementiert, damit die User, die schon die Schablonen gesehen oder gelesen haben, diese schnell erkennen können.

dms.json- ist eine Brücke, mit der die Schablonen mithilfe von DMS veröffentlicht werden können.

Am Anfang der Datei werden die Informationen, die in DMS die Komponente beschreiben, gegeben. Danach werden die Demos der Schablonen wie in der Abbildung 37 zu sehen, geschrieben.

```

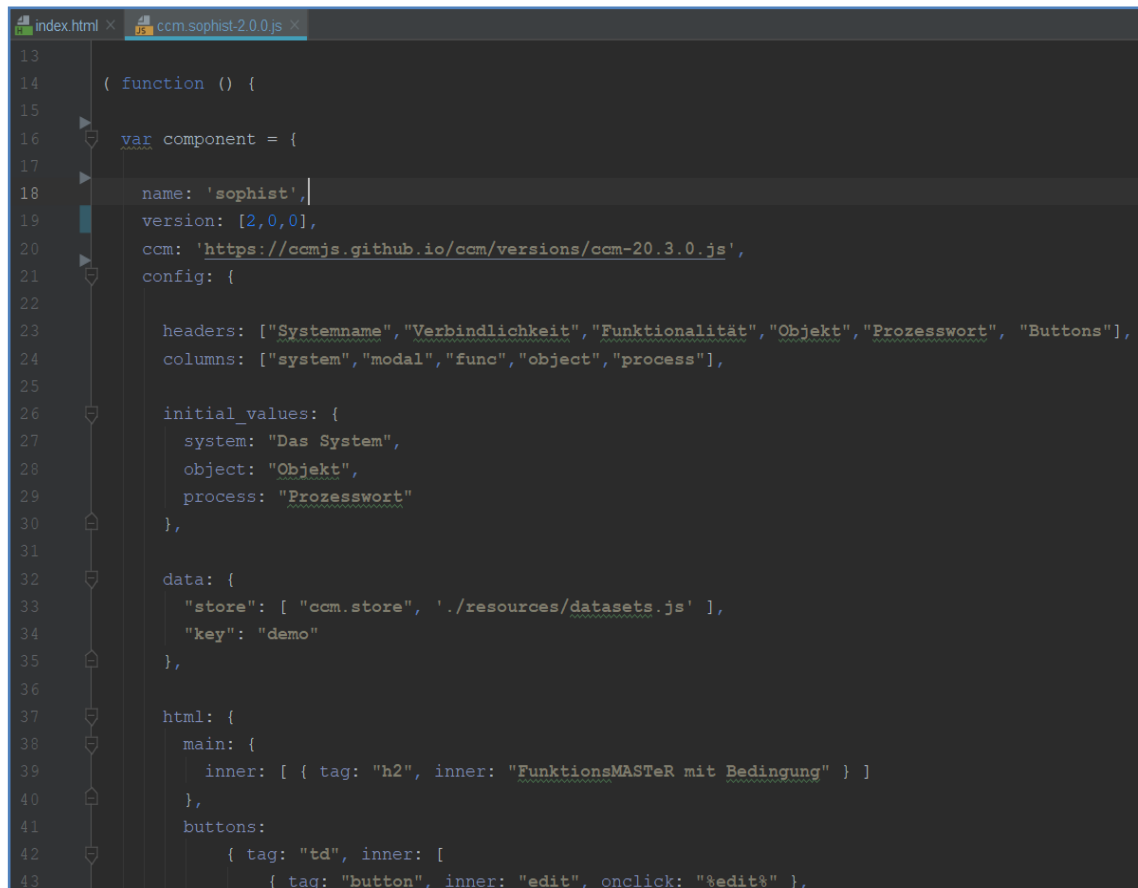
1 {
2   "id": "sophisten",
3   "title": "Sophisten",
4   "icon": "https://turaboev.github.io/sophiste/resources/icon.svg",
5   "abstract": "Anforderungen schreiben",
6   "description": "Eine Anforderungsschablone ist ein Bauplan, der die Struktur eines einzelnen Anforderungssatzes festlegt",
7   "url": "https://turaboev.github.io/sophiste/versions/ccm.sophist-3.0.0.js",
8   "version": "3.0.0",
9   "website": "https://turaboev.github.io/sophiste/",
10  "developer": "Hakimjon Turaboev",
11  "license": "MIT License",
12  "ignore": {
13    "demos": [
14      {
15        "title": "demo",
16        "config": [
17          "ccm.get",
18          "https://turaboev.github.io/sophiste/resources/configs.js",
19          "demo"
20        ]
21      }
22    ]
23  }
24 }

```

Abbildung 37. Ausschnitt von dms.js, eigene Darstellung

ccm.sophist.js - enthält den Code der Sophist-Komponente. Der Code wurde zuerst,

wie andere CCM-Komponenten, in einer namenlosen Funktion geschrieben. In dieser Funktion wurde eine Variable mit dem Namen „component“ erstellt. In dieser „component“ wurden Komponententname, -version, eine Version der ccm-Framework und config geschrieben, die die Schablone getrennt oder zusammen wiederverwendbar macht.



```
13
14 ( function () {
15
16   var component = {
17
18     name: 'sophist',
19     version: [2,0,0],
20     ccm: 'https://ccmjs.github.io/ccm/versions/ccm-20.3.0.js',
21     config: {
22
23       headers: ["Systemname", "Verbindlichkeit", "Funktionalität", "Objekt", "Prozesswort", "Buttons"],
24       columns: ["system", "modal", "func", "object", "process"],
25
26       initial_values: {
27         system: "Das System",
28         object: "Objekt",
29         process: "Prozesswort"
30       },
31
32       data: {
33         "store": [ "ccm.store", './resources/datasets.js' ],
34         "key": "demo"
35       },
36
37       html: {
38         main: {
39           inner: [ { tag: "h2", inner: "FunktionsMASTeR mit Bedingung" } ]
40         },
41         buttons:
42           { tag: "td", inner: [
43             { tag: "button", inner: "edit", onclick: "%edit%" },
```

Abbildung 38. Ausschnitt von ccm.sophist.js, eigene Darstellung

index.html hilft, während der Entwicklung die Komponente im Browser aufzurufen und den Status der entwickelten Komponente zu prüfen. Diese Datei ist in nachfolgender Abbildung zu sehen.



```
js x index.html x
<!DOCTYPE html>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="author" content="Hakimjon Turaboev">
<meta name="licence" content="The MIT License (MIT)">
<title>Sophist</title>

<ccm-sophist key=["ccm.get","resources/configs.js","demo"]></ccm-sophist>

<ccm-sophist key=["ccm.get","/resources/configs.js","FunktionenMASTeR_mB"]></ccm-sophist>

<ccm-sophist key=["ccm.get","https://turaboev.github.io/sophiste/resources/configs.js","condition_time"]></ccm-sophist>
```

Abbildung 39. ein Ausschnitt von index.html, eigene Darstellung

In index.html Datei wurden einige Befehle geschrieben, die Konfigurationen der CCM-Komponente aufrufen z.B.:

```
<CCM-sophist key=["CCM.get","resources/configs.js","demo"]></CCM-sophist>
```

ruft die „demo“ Konfiguration der Komponente in Datei local auf und
Muharram

```
<CCM-sophistkey=["CCM.get",  
"https://turaboev.github.io/sophiste/resources/configs.js",  
"FunktionenMASTeR_mB"]></CCM-sophist>
```

ruft die FunktionenMASTeR_mB Konfiguration der Komponente in Datei config.js online auf.



Abbildung 40. Icon, eigene Darstellung

Das Icon wurde als .svg Datei geschrieben, damit die Größe der Komponente kleiner

wird. Dieses Icon ist 1 KB groß. Das folgende Bild beschreibt, wie das Icon implementiert wurde.

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" viewBox="0 0 64 64" xmlns="http://www.w3.org/2000/svg">
  <svg version="1.1" viewBox="0 0 64 64"
    xmlns="http://www.w3.org/2000/svg" xmlns:bx="https://boxy-svg.com">
    <g id="Layer_1">
      <g>
        <circle cx="32" cy="32" fill="#E0995E" r="32"/>
      </g>
    </g>
    <g id="Layer_2"/>
    <text style="fill: rgb(255, 255, 255);
font-family: Times New Roman; font-size: 8px;
font-weight: 900; white-space: pre;" x="11.9"
      y="18.56">SOPHISTen</text>
    <text style="fill: rgb(255, 255, 255);
font-family: Times New Roman; font-size: 12px;
font-weight: 900; white-space: pre;" x="5.9"
      y="34.56">MASTeRs </text>
  </svg>
</svg>
```

Abbildung 41. Ausschnitt von icon.svg, eigene Darstellung

Die default.css Datei ergibt, wie die Komponente aussehen soll. Die Farbe und Struktur wurde ähnlich der typischen MASTeR-Schablonen erstellt, damit die User, die schon die Schablonen gesehen oder gelesen haben, sie schnell erkennen können.

5. Umfrage

Um die Usability der implementierten Schablonen zu prüfen, wurde eine Umfrage bei mehreren Nutzern durchgeführt. Die Umfrage für die Usability wird nach dem System Usability Scale (SUS) Fragebogens durchgeführt. Abbildung 42 zeigt den Fragebogen auf Englisch, es wurde jedoch für die deutschsprachigen Benutzer die in Deutsch umgeschriebene Version benutzt, welche sich im Anhang befindet.

	Strongly disagree					Strongly agree	
	1	2	3	4	5		
1. I think that I would like to use this system frequently.					✓		4
2. I found the system unnecessarily complex.				✓			1
3. I thought the system was easy to use.		✓					1
4. I think I would need the support of a technical person to be able to use this system.	✓						4
5. I found the various functions in this system were well integrated.		✓					1
6. I thought this system was too inconsistent.			✓				2
7. I would imagine that most people would learn to use this system very quickly.		✓					1
8. I found the system very cumbersome to use.				✓			1
9. I felt very confident using the system.					✓		4
10. I needed to learn a lot of things before I could get going with this system.		✓					3
Total = 22	SUS Score = 22 × 2.5 = 55						

Abbildung 42.SUS-Fragebogen¹³²

Der SUS-Fragebogen berechnet die SUS-Score, bei dem die Werte zwischen 0 und 100 (am positivsten) sind. Das Ergebnis wird wie folgt ausgewertet: Zuerst wird jede Zeile mit der Zahl von 1 bis 5 bewertet. Wenn die Aussage positiv formuliert ist, ist der Wert „Bewertung der Zeile“-„1“. Wenn die Aussage negativ formuliert ist, ist der Wert „5“-„Bewertung der Zeile“. So werden die von jeder Zeile gewonnenen Zahlen addiert.

¹³²Barnum, C.: „Usability Testing Essentials“ 2011 und <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>

Die Summe muss zwischen 0 und 40 liegen. Die Summe wird mit 2,5 multipliziert und ergibt das Ergebnis des Fragebogens. Wenn die Werte unter 60% sind, gibt es Usability-Probleme.

Die von User ausgefüllten Fragebogen ist im Anhang zusehen. In nachfolgender Tabelle ist das Ergebnis der Umfragen für die implementierten, fertigen Schablonen, das nach SUS-Fragebogen gerechnet wurde. Das gesamte durchschnittliche Ergebnis ist 83,61.

User1	User2	User3	User4	User5	User6	User7	User8	User9	gesamt
95	92,5	75	92,5	92,5	82,5	72,5	70	80	83,611111

Abbildung 43. Ergebnis der Umfragen, eigene Darstellung

6. Zusammenfassung

Das Ziel dieser Arbeit war, eine Web-Komponente für die Anforderungserhebung mit den Sophist-Regelwerk-Schablonen zu erstellen. Dafür wurde zunächst der Umgang mit den Satzbestandteilen und dem Sophist Regelwerk erläutert und von diesem Ansatz ausgegangen. Mit dieser Theorie als Grundlage erbringt das Schreiben der Anforderungssätze mit den vorgefertigten Schablonen dem User Zeitersparnis und es lassen sich korrekte Satzstrukturen erbringen. Als Muster wurden die MASTeR-Schablonen genommen. Einige von diesen Schablonen wurden sprachlich analysiert und vor der Implementierung angepasst. Darüber hinaus wurden neue Schablonen in usbekischer Sprache entwickelt. Bei der Entwicklung ergab sich, dass, je mehr die Sprachstruktur von der deutschen Sprachstruktur abweicht, desto mehr Satzänderungen und Umstrukturierungen vorgenommen werden müssen. Nach der Fertigstellung der Schablonen für die Implementierung wurde die Implementierung mithilfe von Framework „CCM“ durchgeführt. Da die mit der CCM erstellten Komponenten und Komponentenkfigurationen wiederverwendbar sind, konnte die Zeit und Arbeit für die Codierung gespart werden. Die Komponenten wurden erfolgreich erstellt und waren funktionsfähig. Abschließend wurde eine kleine SUS-Umfrage unter den Elektrotechnik-, Wirtschafts-, und Informatik Studenten und Sprachencenter-Mitarbeitern der Hochschule durchgeführt. Das Ergebnis in der folgenden Tabelle zeigt, dass die Komponenten mit 83,6 Punkte als nutzerfreundlich einzustufen sind.

Insgesamt kann festgestellt werden, dass viel Zeit eingespart wird und es nutzerfreundlich ist, wenn die Komponenten als elektronische Lehrmittel benutzt und mit den Schablonen Anforderungen erstellt werden. Da die Komponenten weltweit als Link geteilt und benutzt werden können, ist es sehr empfehlenswert, die Schablonen auch in anderen Sprachen zu entwickeln.

Abkürzungsverzeichnis

DMS	Digital Makerspace
CCM	Client-side Component Model
IREB e.V.	International Requirements Engineering Board
MASTeR	Muster Anforderungen-die SOPHIST Templates für Requirements
OER	Open Educational Resources
RE	Requirement Engineering
SE	Software Engineering
SUS	System Usability Scale

Literaturverzeichnis

Ackermann, Philip. *"JavaScript : das umfassende Handbuch"*. Bonn: Rheinwerk Verlag GmbH, 2016.

Balsamiq Studios LLC. *"Wireframes"*. 2008-2019. <https://balsamiq.com/wireframes/> (Zugriff am 26. 08 2019).

Bensch, Mauricio. *"Nebensätze"*. 2018. <https://mein-deutschbuch.de/nebensaetze.html> (Zugriff am 26. 08 2019).

CHIP Digital GmbH. *"Was ist ein Browser einfach erklärt"*. 2019. https://praxistipps.chip.de/was-ist-ein-browser-einfach-erklart_41369 (Zugriff am 26. 08 2019).

Claire, Hana, Jared, Krystal, und Fiona. *"What is a Makerspace"*. Herausgeber: Queensland University of Technology. 01. 05 2019. <http://makerspacesaustralia.weebly.com/what-is-a-makerspace.html> (Zugriff am 26. 08 2019).

Conservation X Labs. *"Digital Makerspace"*. <https://conservationxlabs.com/digital-makerspace> (Zugriff am 26. 08 2019).

Crockford, Douglas. *www.json.org*. 2019. <https://www.json.org/> (Zugriff am 26. 08 2019).

deutschplus-Team, Das. *"Funktionsverben"*. 2012. <https://www.deutschplus.net/pages/Funktionsverben> (Zugriff am 26. 08 2019).

Experience UX. *"What is Wireframing"*. 2019. <https://www.experienceux.co.uk/faqs/what-is-wireframing/> (Zugriff am 26. 08 2019).

Foster, Nathan. *"Top Wireframing Tools"*. 2019. <https://www.creativebloq.com/wireframes/top-wireframing-tools-11121302> (Zugriff am 26. 08 2019).

GmbH, Vogel IT-Medien. *"Was ist Github?"*. 2019. <https://www.dev-insider.de/was-ist-github-a-645831/> (Zugriff am 26. 08 2019).

Hahn, Martin. *"Webdesign. Das Handbuch zur Webgestaltung"*. 2. aktualisierte Auflage. Bonn: Rheinwerk GmbH, 2017.

Jacobsen, Jens, und Matthias Gidda. *"Webseiten erstellen für Einsteiger"*. 2. Auflage. Bonn: Rheinwerk Verlag, 2016.

JetBrains s. r. o. *"Features"*. 2000-2019. <https://www.jetbrains.com/webstorm/features/> (Zugriff am 26. 08 2019).

Joppich, Rainer, und SOPHIST GmbH. *"Die SOPHIST-Satzschablone Rainer Joppich"*. 25. 01 2019. <https://www.youtube.com/watch?v=vr8s6jluq6I> (Zugriff am 26. 08 2019).

Kapori, Bernd. *"Substantivierung von Wortarten"*. 2019. <https://www.cafe-lingua.de/deutsche-grammatik/substantivierung-von-wortarten.php> (Zugriff am 26. 08 2019).

Kaul, Manfred. *"DMS Endbericht"*. Hochschule Bonn-Rhein-Sieg. 2019. https://kaul.inf.h-brs.de/download/DMS_Endbericht.html (Zugriff am 26. 08 2019).

—. *www.stifterverband.org*. 2018. <https://www.stifterverband.org/file/5203/download?token=lay3iM9c > (Zugriff am 11. 04 2019).

Kaul, Manfred, und Andre Kless. *Digital Makerspace*. 2019. <https://ccmjs.github.io/digital-maker-space/#1562446314531X9878992170446217=apps> (Zugriff am 26. 08 2019).

Khoshmashrab, Mehrdad. *"Anforderungsschablonen"*. <https://docplayer.org/10853434-Anforderungsschablonen.html> (Zugriff am 26. 08 2019).

Kless, Andre. „ccm-Vortrag für Entwickler“. 24. 12 2018. <https://www.youtube.com/watch?v=2fUwoH-fRMs> (Zugriff am 26. 08 2019).

Kless, Andre. *"Eingebettetes kollaboratives E-Learning"*. Masterthesis, Sankt Augustin: Hochschule-BonnRhein-Sieg, 2015.

Kless, Andre (akless). *"CCM"*. <https://github.com/ccmjs/ccm> (Zugriff am 26. 08 2019).

Kluge, Roland. *"Die kleine RE-Fibel"*. 3. Auflage. Nürnberg: SOPHIST GmbH, 2016.

—. „www.sophist.de.“ 2016.

https://www.sophist.de/fileadmin/user_upload/Bilder_zu_Seiten/Publikationen/Wissen_for_free/MASTeR_Broschuere_3-Auflage_interaktiv.pdf (Zugriff am 26. 08 2019).

Kluge, Roland, und SOPHIST GmbH. *"Schablonen für alle Fälle"*. 3. Auflage. Nürnberg: SOPHIST GmbH, 2016.

Laborenz, Kai. *"CSS-das umfassende Handbuch"*. Bonn: Galileo Press.

Lubkowitz, Mark. *"Webseiten programmieren und gestalten"*. Bonn: Galileo Press, 2007.

Madvaliyev, A. „*O`zbek tilining izohli lugati*“. Tashkent, 2006.

Nurimonov, A., N. Mahmudov, A. Ahmedov, und S. Solixujaeva. „*O'zbek tilining mazmuniy sintaksisi*“. Taschkent, 1992.

Pluralsight LLC. *"Javascript"*. 2004 - 2019. <https://www.pluralsight.com/paths/javascript> (Zugriff am 26. 08 2019).

Pomaska, Günter. *"Webseiten-Programmierung: Sprachen, Werkzeuge, Entwicklung"*. Springer-Verlag, 2012.

Robson, Elisabeth, und Eric Freeman. *"HTML und CSS von Kopf bis Fuß"*. 2. Auflage. Köln: O'Reilly Verlag GmbH & Co. KG, 2012.

Rupp, Chris, und Rainer Joppich. *"Anforderungsschablonen- der MASTER-Plan für gute Anforderungen, in Rupp, Chris & die SOPHISTen: Requirements-Engineering und – Management, Aus der Praxis von klassisch bis agil"*. München: Carl Hanser, 2014.

The University of North Carolina Wilmington. *"About"*. 04. 01 2011. <https://digitalmakerspace.uncw.edu/About> (Zugriff am 26. 08 2019).

U.S. Department of Health and Human Services. *www.usability.gov*. 2019. <https://www.usability.gov/about-us/index.html> (Zugriff am 26. 08 2019).

W3Schools. *JSON vs XML*. 1999-2019. https://www.w3schools.com/js/js_json_xml.asp (Zugriff am 30. 08 2019).

WeAreTeachers. "*What is a Makerspace*". 27. 06 2017.

<https://www.weareteachers.com/what-is-a-makerspace/> (Zugriff am 26. 08 2019).

Yildirim, Mahir. "*Vollverben & Hilfsverben*". 2019.

<https://www.schulminator.com/deutsch/vollverben-hilfsverben> (Zugriff am 26. 08 2019).

Abbildungsverzeichnis

Abbildung 1. Links K-6 School Makerspace From @iolaniLSFabLab und rechts Stewart-Middle-School-Educational-Makerspace-@DianaLRendina.....	4
Abbildung 2: Screenshot von DMS	5
Abbildung 3. Bringing interactivity to digital common goods.....	6
Abbildung 4. Vorgehen der Einsatz für jede Sophist-Regel (eigene Darstellung, nach Kluge).....	8
Abbildung 5. Einsatzfolge des SOPHIST-Regelwerks, (eigene Darstellung, nach Kluge)	8
Abbildung 6. Die vier Schritte zum Prüfen der Prozesse	13
Abbildung 7. SchrittfürEigenschaftsprüfung	13
Abbildung 8. Die zwei Schritte zum Prüfen von Mengen und Häufigkeiten.....	17
Abbildung 9. Ein Schritt zum Prüfen von Möglichem und Unmöglichem	18
Abbildung 10. Schritte zum Prüfen des Satzes	19
Abbildung 11. Schritte zum Prüfen des Gesamtbildes	22
Abbildung 12. Prioritäten von SOPHIST-REgelwerk nach Rupp und eigene Darstellung	22
Abbildung 13. FunktionsMASTeR ohne Bedingung	24
Abbildung 14. FunktionsMASTeR mit Bedingung.....	26
Abbildung 15. Detaillierter FunktionsMASTeR ohne Bedingung.....	27
Abbildung 16. Detaillierter FunktionsMASTeR mit Bedingung.....	27
Abbildung 17. EigenschaftsMASTeR	28
Abbildung 18. UmgebungsMASTeR	29
Abbildung 19. ProzessMASTeR.....	30
Abbildung 20. BedingungsMASTeR.....	30
Abbildung 21. Frontend Hahn 2017, S.30	31
Abbildung 22. Grundgerüst einer HTML-Seite, eigene Darstellung nach Wiley & Sons, John „HTML & CSS Design and build Websites“ 2011, Indianapolis, Indiana, Seite 20	32
Abbildung 23. Anzeige der „test“ HTML-Seite im Browser, Ergebnis eigener Darstellung	32
Abbildung 24. https://www.w3schools.com/js/js_json_xml.asp	34
Abbildung 25. Mockup für Demofenster der Komponente, eigene Darstellung.....	43
Abbildung 26. Mockup für die "Create APP", eigene Darstellung	43
Abbildung 27. Mockup für FunktionsMASTeR ohne Bedingung, eigene Darstellung...	44
Abbildung 28. Inputfeld in HTML.....	45
Abbildung 29. Inputfeld im Browser	45

Abbildung 30. Optionen für die Verbindlichkeit.....	45
Abbildung 31. Darstellen der Verbindlichkeit.....	46
Abbildung 32. Darstellen einer Schablone.	46
Abbildung 33. die Liste der Sophist-Komponenten-Dateien	47
Abbildung 34. Ausschnitt von config.js.....	47
Abbildung 35. Ausschnitt von datasets.js.....	48
Abbildung 36. Ausschnitt von dms.js.....	48
Abbildung 37. Ausschnitt von CCM.sophist.js	49
Abbildung 38. ein Ausschnitt von index.html	50
Abbildung 39. Icon	50
Abbildung 40. Ausschnitt von icon.svg.....	51
Abbildung 41. SUS-Fragebogen	52