Masterarbeit

AUTOMATISIERTE ERSTELLUNG DER ANLAGENDOKUMENTATION

ausgeführt am

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
Dieter Pichler, BSc
1710322008

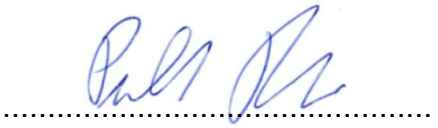
betreut und begutachtet von
Dipl.-Ing. Johannes Fritz, BSc

Graz, im Dezember 2018

.....
Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.



.....

Unterschrift

DANKSAGUNG

An dieser Stelle möchte ich mich bei allen bedanken, die mich im Laufe meines Studiums und vor allem während der Erstellung dieser Masterarbeit unterstützt haben. Ein ganz besonderer Dank gilt Herrn DI Johannes Fritz, BSc, der meine Masterarbeit betreut und begutachtet hat. Die Unterstützung bei der Themenfindung, der Gliederung, bis hin zur Empfehlung unterschiedlicher Literaturquellen kam meiner Arbeit sehr zu Gute. Weiters möchte ich mich bei meinem Betreuer im Unternehmen ANDRITZ Herrn Ing. Alexander Rostek für die Ermöglichung dieser Masterarbeit und die tatkräftige Unterstützung bedanken. Ebenso ist es mir ein Anliegen, meiner Familie und meinen Freunden zu danken. Speziell für das Korrekturlesen, die erhaltenen Vorschläge für inhaltliche Themen und die moralische Unterstützung möchte ich mich herzlichst bedanken.

KURZFASSUNG

Kunden der Maschinen- und Anlagenbauindustrie erwarten sich immer kürzere Lieferzeiten in Kombination mit qualitativ hochwertigen Produkten, wie auch eine vollständige und korrekte Dokumentation. ANDRITZ geht bei nahezu jeder verkauften Maschine- oder Anlage auf die Bedürfnisse und Wünsche der Kunden ein. Aus diesem Grund entsteht eine sehr flexible Produktpalette. Um den Wunsch nach kürzeren Lieferzeiten in Kombination mit der flexiblen Produktpalette erfüllen zu können, muss der Planungsprozess adaptiert werden. Das Ziel dieser Masterarbeit ist die Entwicklung einer Software zur Automatisierung des Planungsprozesses von modularisierten Anlagen und die daraus resultierende Verkürzung der Lieferzeit. Der erste Teil der praktischen Arbeit befasst sich mit der systematischen Entwicklung und Testung dieser Software. Um garantieren zu können, dass die neue Software *Metris Engineering Configurator* den Wünschen der Interessensvertreter entspricht, werden unter anderem modellbasierte Vorgehensweisen angewandt. Im zweiten Teil wird eine Filterpressenanlage vollständig standardisiert, modularisiert und für den Einsatz des *Metris Engineering Configurators* vorbereitet. Im letzten Teil werden die Vorteile der neuen Software an der automatisierten Erstellung der prozess- und automatisierungstechnischen Dokumentation einer Filterpressenanlage aufgezeigt. Durch den Einsatz des *Metris Engineering Configurators* kann die Durchlaufzeit und somit auch die Lieferzeit einer Anlage signifikant verringert werden kann. Folglich können auch die Kosten für die Planung reduziert werden. Aufgrund des automatisierten Planungsprozesses kann zudem eine qualitativ hochwertige Anlagendokumentation garantiert werden. Die Software wurde im Unternehmen eingeführt und kann von allen Mitarbeitern verwendet werden. Die Tests haben gezeigt, dass der *Metris Engineering Configurator* in Zukunft für unterschiedliche Anlagen und Ausrüstungen zum Einsatz kommen kann. Zwei Updates für die Erweiterung der Software werden in dieser Masterarbeit bereits behandelt.

ABSTRACT

In the business of machinery and plant engineering customers expect increasingly shorter delivery times as well as high-quality products and their documentation. ANDRITZ has a very flexible product range due to the customization of almost all sold machines and plants. When these two aspects are combined, it poses a challenge, which can be met by adapting the process of engineering. The aim of this master thesis is to develop a software to fully automate the engineering of modularized plants in order to shorten the delivery times. Model-based methods are used to raise requirements, to develop and test the software and introduce it to the company. The advantages of the newly developed software *Metris Engineering Configurator* are demonstrated by creating the technical documentation related to the process and automation of a concrete plant. For this purpose, a filter press is standardized, modularized and prepared for the *Metris Engineering Configurator*. The deployment of the application for automatizing the engineering of the filter press demonstrates that the processing time can be decreased significantly and correspondingly the costs for the engineering can be equally reduced. Due to the automated engineering process, a high-quality plant and documentation can be guaranteed. The software has been introduced to the company and is ready for use. In future the *Metris Engineering Configurator* can be enhanced to automatize the engineering process of different plants and equipment delivered by ANDRITZ. Two updates for the developed software are already considered.

INHALTSVERZEICHNIS

1 EINLEITUNG

Diese Masterarbeit behandelt das Thema *automatisierte Erstellung der Anlagendokumentation*. Um international wettbewerbsfähig zu bleiben, muss das Engineering vor allem für Maschinen und kleinere Anlagen automatisiert werden. Um diese Herausforderung zu erfüllen, muss eine Lösung zur wissenschaftlichen Fragestellung – wie kann das Engineering und daraus folgend die Erstellung der Anlagendokumentation automatisiert werden – ermittelt werden.

In dieser Einleitung wird als erstes die Motivation zur Untersuchung dieses Themas genauer beschrieben. Im zweiten Unterkapitel wird auf die Ziele der Arbeit eingegangen. Die Vorgehensweise bei dieser Masterarbeit und deren Gliederung werden im dritten Unterkapitel näher erläutert. Im letzten Unterkapitel werden die Begriffe Metris und Comos beschrieben.

1.1 Motivation

Kunden erwarten sich immer kürzere Lieferzeiten für Maschinen und Anlagen. Speziell bei standardisierten Maschinen und kleineren Anlagen planen Kunden mit einer schnellen Lieferung und einem frühestmöglichen Produktionsstart. Je schneller eine Anlage in Betrieb geht, desto schneller kann damit Wertschöpfung generiert werden. Als Lieferant von Anlagen und Ausrüstungen für unterschiedliche Industrien verkauft ANDRITZ keine Massenware, sondern geht bei jeder verkauften Anlage oder Maschine auf die Wünsche der Kunden ein. Somit wird nahezu keine Anlage mehrmals identisch geliefert. Die daraus resultierende individuelle Projektabwicklung erfordert die Beteiligung mehrerer Mitarbeiter und Abteilungen. Die Erstellung der verfahrens-, elektro- und automatisierungstechnischen Anlagendokumentation wird bei ANDRITZ größtenteils im Engineering Programm Comos, welches im Unterkapitel 1.4 erläutert wird, durchgeführt. Die Erstellung dieser Dokumentation erfolgt großteils sequenziell. Beispielsweise dient das P&ID (Piping & Instrumentation Diagram, Rohrleitungs- und Instrumentenfließschema) als Grundlage für die Auslegung der Motoren und auch die Erstellung der Stromlaufpläne. Die Funktionspläne, die als Vorlage für die Erstellung der Automatisierungssoftware verwendet werden, können ebenfalls erst erstellt werden, wenn die Funktion einer Anlage mittels dem P&ID definiert ist. Des Weiteren stellen die Schnittstellen, die entstehen, wenn mehrere Abteilungen an einem Projekt beteiligt sind, ein Problem dar. Hier entsteht die Gefahr eines Informationsverlustes. Es kann passieren, dass eine Abteilung für sie relevante Informationen nicht erhält, da sie von einer anderen Abteilung nicht weitergegeben wurden. Um diese Probleme lösen zu können, muss der Planungsprozess adaptiert werden. Wie die konkreten Änderungen aussehen sollen, wird im nächsten Absatz erläutert.

1.2 Ziele

Die vorliegende Arbeit beschäftigt sich mit der Automatisierung des beschriebenen Planungsprozesses. Dieses Ziel wird mit der Entwicklung und Einführung einer Software erreicht, welche die im vorherigen Unterkapitel beschriebenen technischen Dokumente im Engineering Programm Comos automatisiert erstellt. Das Engineering Programm Comos wird im Unterkapitel 1.4 näher erläutert. Darüber hinaus können neben diesen Dokumenten, unterstützende Dokumente, wie z.B. Bestelllisten automatisiert generiert

werden. Um die Funktion der neuen Software, welche als *Metris Engineering Configurator* bezeichnet wird, testen und deren Vorteile aufzeigen zu können, wird die technische Dokumentation einer Filterpressenanlage vollautomatisch erstellt. Die Herkunft des Namens *Metris Engineering Configurator* wird ebenfalls im Unterkapitel 1.4 erläutert. Damit die neue Software die Dokumente der Filterpressenanlage erstellen kann, muss die Filterpressenanlage standardisiert und modularisiert und für die Verwendung der neuen Software vorbereitet werden. Weiters soll die neue Software skalierbar ausgeführt sein. Das bedeutet, dass sie zukünftig für andere Maschinen oder Anlagen zum Einsatz kommen kann.

1.3 Gliederung der Arbeit

Für die Entwicklung des *Metris Engineering Configurators* werden systematische Vorgehensmodelle verwendet. Diese sollen eine qualitativ hochwertige Software unter Einhaltung der geplanten Entwicklungskosten garantieren. Somit orientiert sich die gesamte Arbeit am V-Modell, welches im nächsten Kapitel erläutert wird. Im Kapitel danach wird die Theorie zur Erhebung und Dokumentation der Anforderungen beschrieben. Im Kapitel 4 wird auf die theoretischen Grundlagen zur Testung der Software eingegangen. Das Kapitel 5 stellt das erste praktische Kapitel dar. In diesem werden die konkreten Anforderungen an den *Metris Engineering Configurator* aufgezeigt. Die Modularisierung der Filterpressenanlage und auch die generelle Funktionsweise dieser werden im Kapitel 6 behandelt. Im siebenten Kapitel werden die Prüfungen, die an der Software durchgeführt wurden, beschrieben. Das achte Kapitel beschreibt die Erstellung der Anlagendokumentation der Filterpressenanlage mit und ohne die neue Software. Im letzten Kapitel wird das Fazit über die gesamte Masterarbeit gezogen. Das betrifft zum einen das konkrete Ergebnis, im Zuge dessen die Vor- und Nachteile des *Metris Engineering Configurators* erläutert werden. Zum anderen wird im Kapitel 9 die Vorgehensweise bei der Entwicklung der neuen Software reflektiert. Mit dem Ausblick wird diese Arbeit abgeschlossen.

Da das Vokabular in den behandelten Themenfeldern auch im deutschsprachigen Raum von englischen Begriffen dominiert wird, werden in dieser Masterarbeit die Originalbegriffe großteils beibehalten.

1.4 Beschreibung Metris / Comos

Mit der Technologiemarke *Metris* werden die IoT (Internet of Things) Lösungen des Unternehmens ANDRITZ zusammengefasst. [1] Der Begriff *Metris* ist eine Kombination aus den Wörtern Metis und Matrix. Metis steht in der griechischen Mythologie für praktisches, komplexes und implizites Wissen. Der Terminus Matrix wurde aufgrund der Verwendung numerischer Daten zur Regelung und Optimierung von Maschinen gewählt. [2]

Generell beruht *Metris* auf drei Säulen, Smart Sensors, Big Data und Augmented Reality. Neben dem Einsatz intelligenter Sensoren werden von ANDRITZ unter dem Begriff Smart Sensors auch Mikro- und Drahtlossensoren angeboten. Big Data steht generell für die Verarbeitung von großen Datenmengen. Bei ANDRITZ wird dieser Begriff für die Optimierung von Anlagen zur Verringerung der Stillstandszeiten und

[1] Vgl. ANDRITZ (2017a), Online-Quelle [27.November.2018].

[2] Vgl. ANDRITZ (2018a), Online-Quelle [27.November.2018].

Reduzierung von Verbrauchsstoffen mithilfe von Simulationsprozessen verwendet. Der Begriff Augmented Reality (computergestützte Erweiterung der Realitätswahrnehmung) steht bei ANDRITZ für die Anzeige von Informationen direkt an der Anlage. [3]

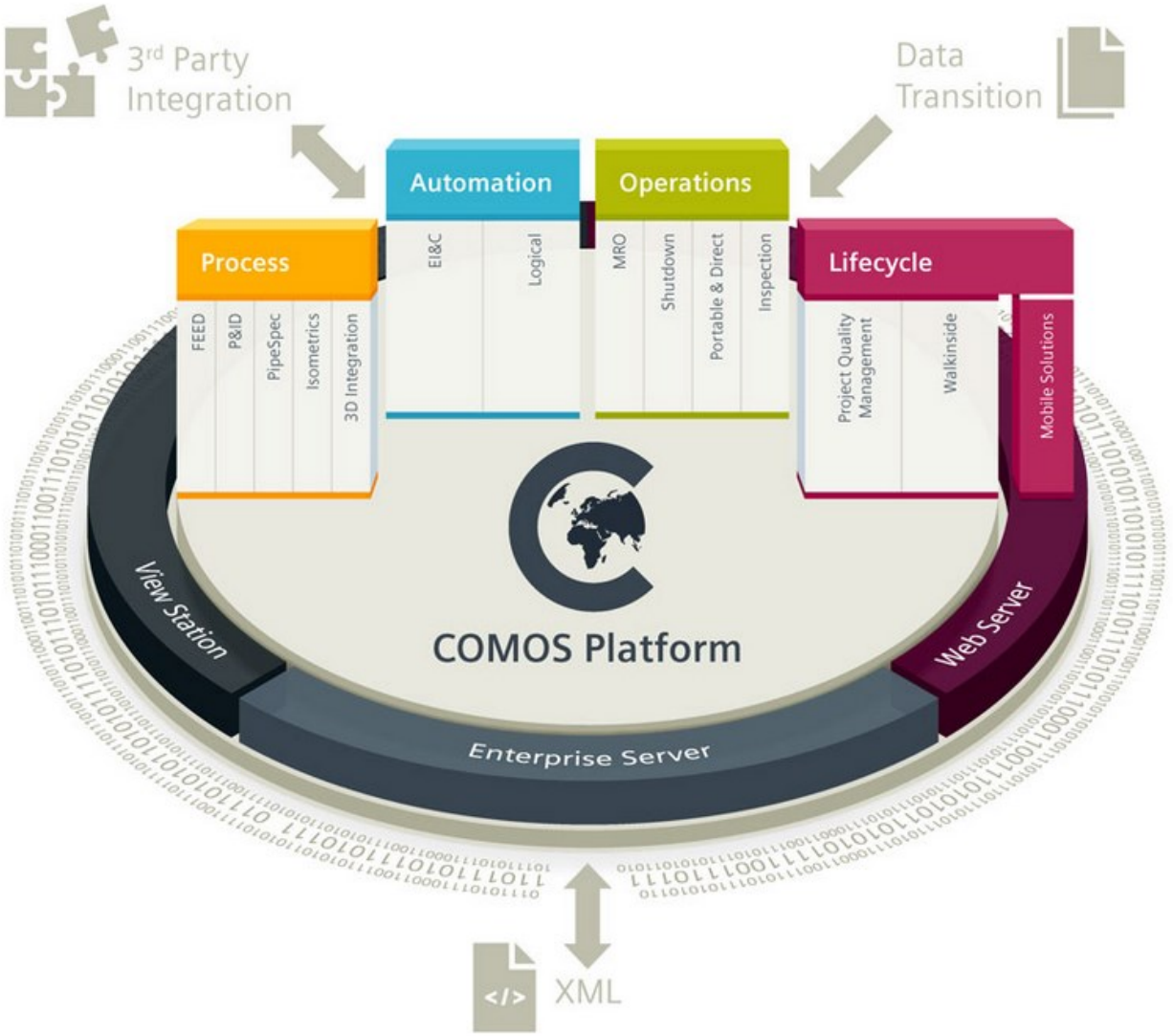


Abbildung 1: Comos, Quelle: Siemens AG (2018a), Online-Quelle [27.November.2018].

Comos ist eine Plant Engineering Software des Unternehmens Siemens AG. Wie in Abbildung 1 ersichtlich, ermöglicht das datenbankbasierte Programm das Anlagenmanagement von der Planung bis zur Betriebsphase einer Anlage. Comos ist in fünf Bereiche gegliedert. Die *Comos Platform* liefert mit der Datenbank die Basis für die weiteren Module. Um eine Anlage verfahrenstechnisch abbilden zu können, wird das Modul *Comos Process* verwendet. Mit diesem können z.B. P&IDs und auch Rohrleitungsisometrien erstellt werden. Das zweite Modul *Comos Automation* ist für die Planung einer Anlage aus Sicht der Automatisierung zuständig. Es umfasst sowohl die Planung des elektrischen-, hydraulischen- und pneumatischen Energie- und Informationsflusses, wie auch die Erstellung von Funktionsplänen. Das dritte Modul *Comos Operations* und das vierte Modul *Comos Lifecycle* betreffen hauptsächlich den Betrieb und die Wartung

[3] Vgl. ANDRITZ (2017a), Online-Quelle [27.November.2018].

einer Anlage. Beispielsweise können die Planungsdaten in der Betriebsphase weiterhin genutzt und auch bearbeitet werden. Des Weiteren können z.B. Inspektions- und Wartungsaufgaben geplant werden. *Comos Lifecycle* ermöglicht das Informationsmanagement in den unterschiedlichen Lebenszyklusphasen einer Anlage. [4]

Das Engineering Programm Comos kommt bei ANDRITZ hauptsächlich für die verfahrens- und automatisierungstechnische Planung von Anlagen zum Einsatz. Da mit dem *Metris Engineering Configurator* Anlagen unter anderem über eine Schnittstelle von einem beliebigen Ort aus konfiguriert werden können und diese Software somit als Teil der IoT-Lösungen gesehen wird, erhielt sie den Technologiemarkennamen *Metris*. Der zweite Teil *Engineering Configurator* kommt zum einen daher, dass User der neuen Software eine Anlage nicht mehr selbst planen, sondern konfigurieren. Zum anderen wird aufgrund der Flexibilität der neuen Software der Begriff *Engineering Configurator* anstatt beispielsweise Plant Configurator verwendet. Sie soll in Zukunft nicht nur für Anlagen, sondern z.B. auch für RIO-Schränke (Remote-I/O-Schrank) zum Einsatz kommen.

[4] Vgl. Siemens AG (2018b), Online-Quelle [27.November.2018].

2 SYSTEMATISCHE VORGEHENSMODELLE

Unter einem Vorgehensmodell wird jede modellhafte, abstrahierende Beschreibung von Vorgehensweisen, Richtlinien, Empfehlungen oder Prozessen, die für einen abgegrenzten Problembereich gilt und in einer möglichst großen Anzahl von Einzelfällen anwendbar ist, verstanden. [5] Mit der Verwendung von Vorgehensmodellen wird das Ziel einer hierarchischen Gliederung der Gesamtaufgabe verfolgt. [6] Der Zweck dieser Gliederung wird im ersten Unterkapitel, welches sich mit dem ältesten Vorgehensmodell beschäftigt, erläutert. In den folgenden vier Unterkapiteln werden weitere konventionelle bis aktuelle Vorgehensmodelle beschrieben. Im sechsten Unterkapitel ist eine Zusammenfassung und der Nutzen der Vorgehensmodelle für diese Masterarbeit zu finden.

2.1 Wasserfallmodell

Weil die Wichtigkeit einer systematischen Vorgehensweise bei der Softwareentwicklung bereits Anfang der 60er Jahre erkannt wurde, entstand in diesem Zeitraum die neue Disziplin des Software-Engineerings. Sie hatte drei Hauptziele: [7] Eine Software sollte eine gesicherte hohe Qualität aufweisen, innerhalb des Budget-Rahmens kostengünstig entwickelt werden und zum geplanten Zeitpunkt auslieferbar sein. Eine der wichtigsten Erkenntnisse dieser Zeit, welche heute noch weitgehend gültig ist, war die Aufsplittung der komplexen Anforderungen in einzelne Entwicklungsschritte, welche wiederum in weitere Schritte aufgeteilt werden können. Dazu wurde ein allgemeines Vorgehensmodell, dessen erstmalige Beschreibung W. Royce zugeordnet wird, entwickelt. [8] Dieses Wasserfallmodell, welches von W. Royce jedoch noch nicht so genannt wurde, ist in nachfolgender Abbildung 2 ersichtlich. [9]

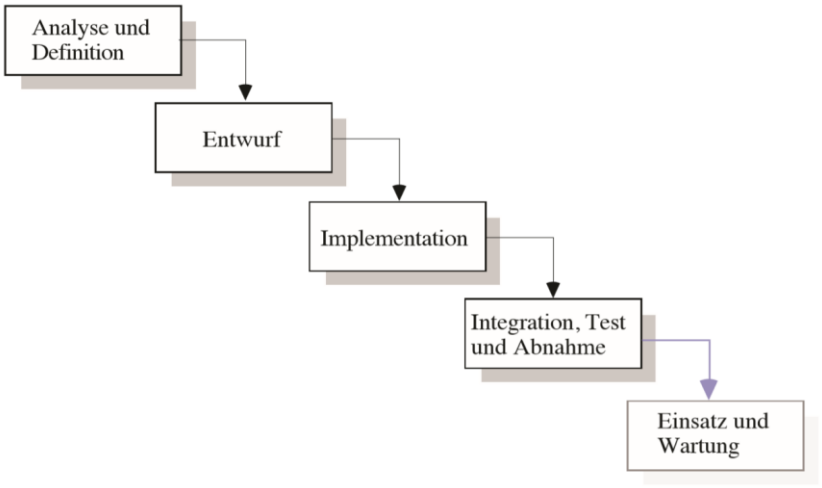


Abbildung 2: Wasserfallmodell, Quelle: Partsch (2010), S. 3.

[5] Vgl. Stahlknecht/Hasenkamp (2002), S. 219.

[6] Vgl. Henrich (2002), S. 30.

[7] Vgl. Partsch (2010) S. 2, zitiert nach Naur/Randell (1968) und Buxton/Randell (1969).

[8] Vgl. Royce (1970).

[9] Vgl. Partsch (2010), S. 2.

Die erste Phase des Modells, die Analyse- und Definitionsphase beschreibt die genaue Spezifikation der Aufgabenstellung in einem Pflichtenheft (*Was* des softwaregestützten Systems). Mit dieser Phase beschäftigt sich auch das Requirements Engineering. In der Entwurfsphase erfolgt die Konzeption der Architektur der neuen Software. Neben der Architektur wird auch festgelegt, welche Teile hardware- und welche softwaretechnisch gelöst werden sollen (*Was*, *Wo* und *Wie* der Bausteine). Die Realisierung des Konzeptes durch Übertragung der Softwarekomponenten in eine funktionsfähige Software erfolgt in der Implementationsphase (*Wie* des Softwareteilsystems). Die letzte Phase des Entwicklungsprozesses, Integration, Test und Abnahme, beschäftigt sich mit dem Zusammenführen der einzelnen Teilsysteme zu einer Gesamtsoftware und deren Testung. Zu dieser Phase gehören auch sämtliche interne und externe Abnahmen. Ist die Entwicklung der Software bzw. des Systems abgeschlossen, wurden alle Prozessschritte, außer dem letzten, Einsatz und Wartung, durchlaufen. Hier werden alle Aktivitäten zusammengefasst, die nicht zur Entwicklung gezählt werden, wie z.B. Optimierung, Wartung und Fehlerkorrektur. [10]

Das in Abbildung 2 gezeigte Wasserfallmodell ohne Zyklen zum vorherigen Schritt wird auch als strenges Wasserfall- oder Einbahnstraßenmodell bezeichnet. Dieses wurde Ende der 70er Jahre als das korrekte, weil rational begründete Modell bezeichnet. Da beim strengen Wasserfallmodell jede Phase einer Aktivität zugeordnet werden kann und jede Aktivität als Ergebnis ein Dokument hervorbringt, ist das Wasserfallmodell in der Literatur auch als Dokumentenmodell bekannt. Dieses Modell führte jedoch zu heftigen Auseinandersetzungen zwischen Akademikern und Praktikern. Von den Akademikern wurde der Vorwurf strapaziert, dass die Praktiker einfach zu undiszipliniert seien. Diese wiederum entgegneten, dass eine in der Praxis gängige Entwicklung ohne eine vollständige Spezifikation mit diesem Modell nicht vereinbar ist. Vor allem für das Projektmanagement ist das Wasserfallmodell auch in der nicht so strengen Auslegung, in der es Zyklen zur vorherigen Phase gibt, ungeeignet, da sich der Fortschritt aufgrund dieser Phasen nicht objektiv beurteilen lässt. Z.B. kann morgen wieder am Konzept gearbeitet werden, obwohl heute bereits die Realisierung der Software nahezu abgeschlossen ist. Einen wesentlichen Beitrag zur Lösung dieses Problems leisten die Meilensteine. Diese kommen beim Phasenmodell, welches im folgenden Unterkapitel beschrieben wird, zum Einsatz. [11]

2.2 Das Phasenmodell

Beim Phasenmodell wird die Entwicklung einer Software oder eines Systems wie beim Wasserfallmodell in einzelne Phasen aufgeteilt. Es gibt zwei wesentliche Unterschiede zwischen dem Phasen- und dem Wasserfallmodell. Erstens wird dem Ende jeder Phase ein Meilenstein zugeordnet. Ein Meilenstein ist ein ereignisorientierter Zeitpunkt. [12] Das heißt, dass jedem Meilenstein zwar ein Zeitpunkt als Ziel zugeordnet wird, der Meilenstein jedoch erst erreicht ist, wenn ein bestimmtes Ereignis eingetreten ist. Z.B. kann für die Phase *Abnahme* der Meilenstein *Software vom Kunden abgenommen* lauten. Zweitens gibt es im Phasenmodell keine Zyklen, d.h. mit Erreichen eines Meilensteines kann die vorhergehende Phase nicht wieder geöffnet werden. Kein Prozessmodell kann Fehler, welche erst in einer späteren Phase auffallen,

[10] Vgl. Partsch (2010), S. 2 f.

[11] Vgl. Ludewig/Lichter (2013), S. 156 – 160.

[12] Vgl. Wallin/u.a. (2002).

verhindern oder den Kunden davon abhalten, Anforderungen nach Abschluss der Definitionsphase vorzutragen. Somit kann das Phasenmodell die Rückkehr zu vorherigen Tätigkeiten nicht verbieten. Jedoch wird die Tätigkeit, welche einer vorhergehenden Phase zugeordnet werden müsste, unter den Bedingungen der gerade laufenden Phase abgearbeitet. Mit dieser Änderung wurde das Problem des nicht objektiven Fortschrittes, welches am Ende von Unterkapitel 2.1 beschrieben wurde, gelöst. [13]

Das Phasenmodell bringt gegenüber dem Wasserfallmodell noch weitere Vorteile. Einer davon ist die präzise Planung und Organisation des Projektes. Wird ein Meilenstein nicht zum geplanten Zeitpunkt erreicht, fällt der Verzug sofort auf. Dadurch können auch die nötigen Ressourcen, wie das Personal, einfacher geplant werden. Tritt der beschriebene Fall ein, ist es eventuell notwendig, in der folgenden Phase mehr Personal bereitzustellen. Ein weiterer wesentlicher Vorteil ist, dass die Durchführung der Prüfungen gewährleistet ist. Da jeder Meilenstein und somit Abschluss einer Phase an ein Dokument geknüpft ist, welches auch geprüft werden muss, werden etwaige Abweichungen bereits spätestens am Ende jeder Phase und nicht erst am Ende des Projektes erkannt. Entspricht ein Dokument nicht den Anforderungen, wurde der Meilenstein nicht erreicht. Auch das sogenannte „90 %-fertig-Syndrom“ ist ausgeschlossen. Dieses Syndrom beschreibt, dass für den gefühlten Fertigstellungsgrad von 90 % nur 10 % der Zeit aufgewendet werden und für die restlichen 10 % des Fertigstellungsgrades 90 % der Zeit notwendig sind. Der Grund dafür ist, dass leichtere Aufgaben am Anfang gelöst werden. [14]

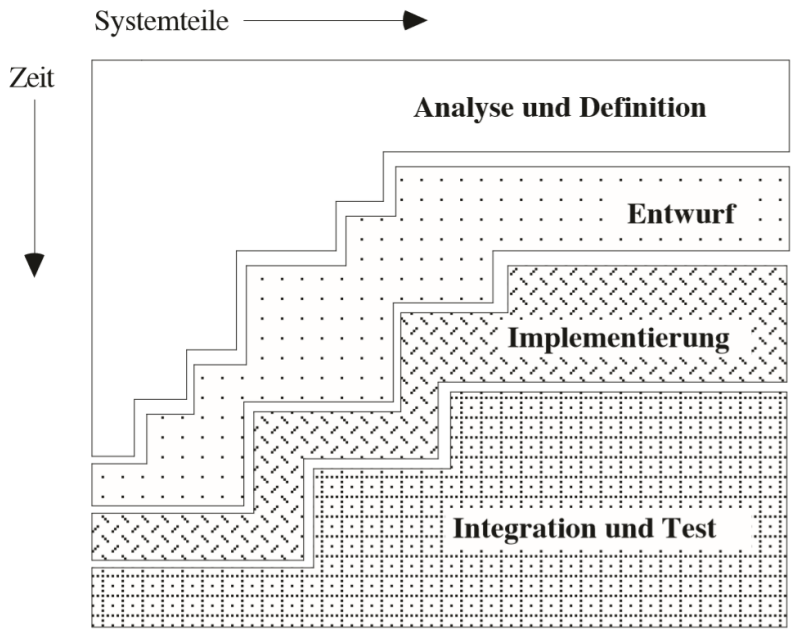


Abbildung 3: Realistisches Phasenmodell, Quelle: Partsch (2010), S. 3.

Da das sequentielle Durchlaufen der Phasen aufgrund von z.B. unvollständigen Spezifikationen eine Idealvorstellung ist und in der Praxis großteils nicht möglich, wurde ein realistisches Phasenmodell, welches in Abbildung 3 ersichtlich ist, entwickelt. Der wesentliche Unterschied zum realen Phasenmodell besteht darin, dass die Abarbeitung der Phasen überlappend sein kann. So können sich verschiedene Systemteile

[13] Vgl. Ludewig/Lichter (2013), S. 183 – 186.

[14] Vgl. Ludewig/Lichter (2013), S. 187.

zu einem bestimmten Zeitpunkt in unterschiedlichen Bearbeitungsphasen befinden. Es dominiert jedoch immer eine Phase und das Ende einer Phase, wird wie beim idealen Phasenmodell mit einem Meilenstein abgeschlossen. Eine Weiterentwicklung des Phasenmodells ist das V-Modell, welches im nachfolgenden Unterkapitel 2.3 näher erläutert wird. [15]

2.3 V-Modell

Auf der Website des deutschen Bundesministeriums des Inneren, für Bau und Heimat wird das aktuelle V-Modell, das V-Modell XT als ein *flexibles Modell zum Planen und Durchführen von Systementwicklungsprojekten* beschrieben. Die erste Version des V-Modells wurde 1992 von der Bundeswehr Deutschland veröffentlicht. Bisher gab es zwei Revisionen, die im Jahr 1997 mit dem Titel V-Modell 97 und im Jahr 2005 unter dem Titel V-Modell XT erschienen sind. [16] Seit Veröffentlichung des V-Modells XT wird es halbjährlich aktualisiert. [17] Die aktuelle Version 2.2 des V-Modells XT wurde im April 2018 veröffentlicht.

Bei der ersten Version und beim V-Modell 97 stand vor allem der Inhalt im Mittelpunkt. Da diese relativ starr aufgebaut sind, wurde 2002 das Projekt WEIT (Weiterentwicklung des IT-Entwicklungsstandards des Bundes) gestartet, aus dem das V-Modell XT entstand. Ziel dieses Projektes war es, das V-Modell so zu ändern, dass zum einen die Anpassung auf konkrete Projekte und zum anderen die Anpassung auf verschiedene Organisationen erleichtert wird. Um das erreichen zu können, basiert das gesamte V-Modell auf einem Metamodell. Somit wird das Hauptaugenmerk nicht mehr auf den Inhalt, sondern auf die Beziehungen zwischen den Modellelementen und die Modellierungsmöglichkeiten an sich gelegt. Dadurch ist es möglich, dass Werkzeuge nach dem Prinzip des V-Modells arbeiten, ohne direkt auf dessen Inhalt einzugehen. Sie können auch mit jedem Werkzeug, welches auf dem Metamodell des V-Modells aufbaut, interagieren. [18]

Aufgrund der oben beschriebenen Flexibilität der neuesten Revision des V-Modells bekam es den Zusatz XT, welcher für eXtreme Tailoring steht. [19]

Wie in Abbildung 4 ersichtlich, ist der Namensgeber des V-Modells die V-förmige Vorgehensweise bei der Entwicklung von Software und Systemen. Diese wurde erstmals von Barry Boehm Ende der 1970er Jahre beschrieben. Kernidee des Systems ist die Beschreibung der Anforderungen vom großen Ganzen (in Abbildung 4 *Anforderung* genannt) und die Zerlegung derer in kleine, detaillierte Einheiten (in Abbildung 4 *Komponente* genannt). Außerdem wird jedem Dekompositionsschritt am absteigenden Ast ein Test- bzw. Prüfschritt am aufsteigenden Ast zugeordnet. [20]

---

[15] Vgl. Partsch (2010), S. 3 f.

[16] Vgl. Beauftragte der Bundesregierung Deutschland für Informationstechnik (2018), Online-Quelle [27.November.2018].

[17] Vgl. Höhn/Höppner (2008), S. 3.

[18] Vgl. Kuhrmann/Ternité/Friedrich (2011), S. 1 f.

[19] Vgl. Beauftragte der Bundesregierung Deutschland für Informationstechnik (2018), Online-Quelle [27.November.2018].

[20] Vgl. Angermeier/u.a. (2018), Online-Quelle [27.November.2018], S. 5.

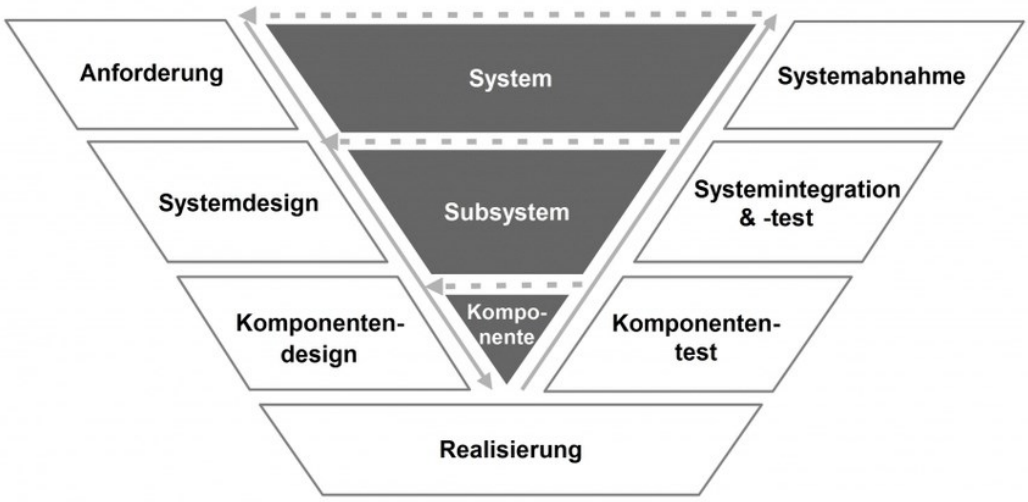


Abbildung 4: Das V-Modell, Quelle: Togar (2017), Online-Quelle [27.November.2018].

2.4 Alternative Vorgehensmodelle

Da auch das Phasen- oder V-Modell nicht die perfekten Vorgehensmodelle für die Entwicklung einer Software oder eines Systems sind und vor allem die Phasenorientierung an sich in Frage gestellt wird, wurden alternative Vorgehensmodelle entwickelt. Diese werden im Folgenden kurz erläutert. Ein Modell, das speziell auf die Managementaspekte eingeht, ist das Spiralmodell. Dabei handelt es sich um ein generisches Modell, welches von der spezifischen Vorgehensweise abstrahiert und nur festlegt, dass die Entwicklung eines Systems aus aufeinanderfolgenden Entwicklungsschritten besteht. Alle Schritte tragen zu den Gesamtkosten bei, wodurch die Spiralform entsteht. Eine weitere Alternative zum Wasserfall- und Phasenmodell ist der Prototyping-Ansatz. Bei dieser Methode werden frühzeitig lauffähige Programme (Prototypen) erstellt, mit denen experimentiert und so Fehler oder Missverständnisse frühzeitig erkannt werden. Beim Prototyping-Ansatz wird zwischen dem explorativen Prototyping und der evolutionären Softwareentwicklung unterschieden. Beim explorativen Prototyping steht das Klären der Aufgabenstellung im Vordergrund. Die evolutionäre Softwareentwicklung sieht den Prototyp als Rohling des Zielsystems, der sukzessive modifiziert und erweitert wird. Alle beschriebenen Vorgehensmodelle, außer der evolutionären Softwareentwicklung sind lineare Vorgehensmodelle. [21]

Neben der evolutionären Softwareentwicklung ist auch die iterative Entwicklung ein Beispiel für eine nichtlineare Vorgehensweise. Bei dieser wird ein System in mehreren geplanten und kontrollierten Iterationsschritten entwickelt. Jede Iteration stellt einen vollständigen Entwicklungszyklus dar, bei dem vor allem die Erkenntnisse aus vorhergehenden Schritten einfließen sollen. Bei der inkrementellen Entwicklung, welche ein weiteres Beispiel für eine nichtlineare Vorgehensweise ist, wird das System in vorher festgelegten Ausbaustufen entwickelt. Ein gänzlich anderer Ansatz wird mit der formalen Softwareentwicklung verfolgt. Hier wird im Zuge der Analyse eine formale Spezifikation, in der das Problem präzise definiert wird, erstellt. Die Übereinstimmung der formalen Definition mit der ursprünglichen Problemstellung wird im Zuge einer Validierung überprüft. Beim Übergang von der Spezifikation zur Implementierung (auch Verifikation genannt)

[21] Vgl. Partsch (2010), S. 4.

wird mittels mathematischer Modelle überprüft, ob die Implementation der Spezifikation entspricht. Bei der Vorgehensweise des Unified Process werden die Stärken des Phasenmodells, welche die Planung und das Management umfassen mit den Stärken der iterativen und inkrementellen Entwicklung, die vor allem die Erkennung von Risiken und deren Beseitigung sind, kombiniert. [22]

2.5 Agile Prozesse

Seit den späten Neunzigern steigt die Anzahl neuer Prozesse, welche weniger schwergewichtig, dafür adaptiver sind, stetig an. Ein Mitgrund für diese Entwicklung war das Aufkommen wesentlicher Änderungen in der Softwareentwicklung wie z.B. die objektorientierte Programmierung oder das Test-Driven Development. Die Grundannahme dieser Prozesse ist, dass eine Software schnell zu entwickeln, diese vom Kunden testen zu lassen und die Fehler zu korrigieren oder die Kundenwünsche einfließen zu lassen, effizienter ist, als vor Beginn der Entwicklung eine genaue Spezifikation zu schreiben. Ein paar bekannte Beispiele für agile Prozesse sind: Dynamic Systems Development Method (DSDM), Feature-Driven Development (FDD), Adaptive Software Development, Scrum, Extreme Programming (XP), Open Unified Process (Open UP), Agile RUP, Kanban, Lean und Crystal Methods. Um die Gemeinsamkeiten hinter den agilen Entwicklungsmethoden herauszufinden und niederzuschreiben, trafen sich im Jahr 2001 Entwickler und Anwender von verschiedenen agilen Prozessen. Das Manifest, welches bei diesem Treffen erstellt wurde, besitzt bis heute seine Gültigkeit: [23]

„Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

- *Individuen und Interaktionen mehr als Prozesse und Werkzeuge*
- *Funktionierende Software mehr als umfassende Dokumentation*
- *Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung*
- *Reagieren auf Veränderung mehr als das Befolgen eines Plans*

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.“ [24]

Neben diesen vier Hauptaussagen des Manifests gibt es noch zwölf weitere Prinzipien, wie z.B. *„Heisse Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.“* [25] oder *„Einfachheit -- die Kunst, die Menge nicht getaner Arbeit zu maximieren -- ist essenziell.“* [26] Alle agilen Prozesse haben sich die Einfachheit und Flexibilität, welche sich im Manifest widerspiegeln, als Ziel gesetzt. Außer diesen grundsätzlichen Aussagen und Prinzipien sind sie jedoch ziemlich verschieden.

[22] Vgl. Partsch (2010), S. 4 f.

[23] Vgl. Leffingwell (2011), S. 12.

[24] Beck/u.a. (2001a), Online-Quelle [27.November.2018].

[25] Beck/u.a. (2001b), Online-Quelle [27.November.2018].

[26] Beck/u.a. (2001b), Online-Quelle [27.November.2018].

Im Kreisdiagramm der Abbildung 5 sind die am häufigsten verwendeten agilen Methoden illustriert. Hier ist z.B. ersichtlich, dass bei 56 % aller agilen Methoden das sogenannte Scrum verwendet wird. Wird die Summe aus ScrumBan, welches eine Kombination von Scrum und Kanban ist und die zweite Hybrid-Methode Scrum/XP und Scrum selbst gebildet, wird ein Marktanteil von 70 % erreicht. Aufgrund der Verwendungshäufigkeit wird im Folgenden nur auf Scrum näher eingegangen.

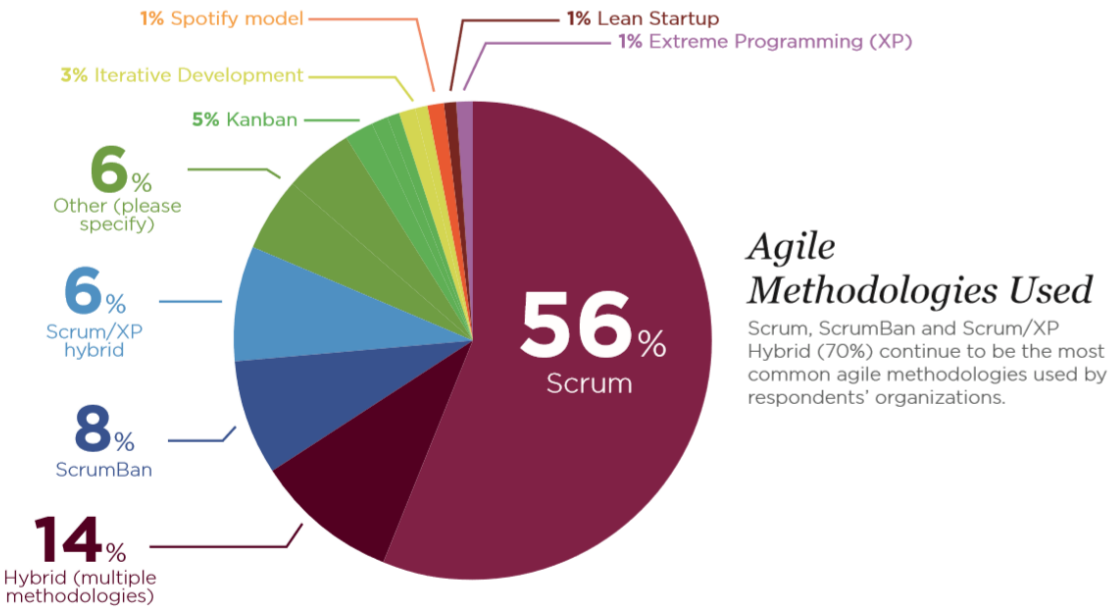


Abbildung 5: Einsatzhäufigkeit agiler Methoden, Quelle: COLLABNET/VERSIONONE (2017), S. 9.

Scrum ist ein agiler Ansatz zum Management von Software- und Systemprojekten. Er wurde Mitte der Neunziger von Jeff Sutherland, Ken Schwaber und Mike Beedle entwickelt und 1995 erstmals publiziert. Das Prinzip von Scrum basiert auf den Ideen von Takeuchi und Nonaka, welche im Zuge der Lean Production bereits 1986 entwickelt wurden. Ziel der Lean Production war es, Produktionsprozesse an die sich immer schneller ändernden Anforderungen anzupassen. Im Kern der Ideen von Takeuchi und Nonaka stand das sich selbst organisierende Team. Bei diesem Team sind die Arbeitsaufgaben nicht streng aufgeteilt, sondern jeder ist für den gesamten Erfolg verantwortlich. Von dieser Idee stammt auch der Name der Methode. Er bezieht sich auf das Gedränge (englisch: *Scrum*) einer Rugby-Mannschaft nach jeder Spielunterbrechung, um den Ball zu erobern. Ist das gelungen, erfolgt ein schneller *Sprint*, um möglichst viel Raum zu gewinnen. Der Begriff des Sprints wird auch bei Scrum verwendet und später in diesem Kapitel beschrieben. Scrum definiert einen iterativen, inkrementellen Prozess für die Entwicklung einer Software oder eines Systems. Im Gegensatz zu anderen agilen Methoden wie XP beschränkt sich Scrum auf das Management und schlägt keine Techniken vor. Deswegen umfasst der offizielle Scrum Guide von 2016 nur 17 Seiten inklusive Deckblatt, Inhaltsverzeichnis und Danksagungen. [27] Aufgrund dieser Einfachheit kann Scrum schnell erlernt und eingeführt werden. [28]

[27] Vgl. Maximini (2018), S. IX.

[28] Vgl. Ludewig/Lichter (2013), S. 228 f.

Generell gibt es im Scrum Team drei Rollen. Den Scrum Master, den Product Owner und das Entwicklungsteam. Dabei ist es nicht erlaubt, dass dieselbe Person, die die Rolle des Scrum Masters innehat, auch die Rolle des Product Owners ausfüllt und umgekehrt. Jede Rolle hat einen definierten Aufgabenbereich und eine Person einer Rolle darf der Person einer anderen Rolle nicht dessen Vorgehensweise vorschreiben. Sämtliche spezielle Scrum Begriffe, die im Folgenden verwendet werden, werden im nächsten Absatz näher erläutert. Der Scrum Master ist für die reibungslose Ausführung des Scrum Prozesses verantwortlich. Somit organisiert und moderiert er die Meetings, beseitigt störende äußerliche Einflüsse, ist für die Gruppendynamik verantwortlich und beschafft, wenn nötig, zusätzliche Ressourcen. Der Product Owner ist, wie der Name schon sagt, für das Produkt an sich verantwortlich. Dazu gehören auch die Erstellung der Product Vision, das Managen der Anforderungen der Stakeholder und die fachlichen Anforderungen für das Product Backlog zu formulieren und zu pflegen. Das Entwicklungsteam besteht im Optimalfall aus 3-6 Personen. Diese sind alle gleichberechtigt (keine Hierarchie) und sollten interdisziplinär besetzt sein. Die Aufgabe des Entwicklungsteams ist neben der Umsetzung der geplanten Aufgaben in ein Produkt, die Organisation der Sprints und deren Planung. Die Sprint-Planung wird mit dem Product Owner gemeinsam durchgeführt. Zusammengefasst ist das Entwicklungsteam für die Lieferung eines nützlichen Produktes verantwortlich. [29]

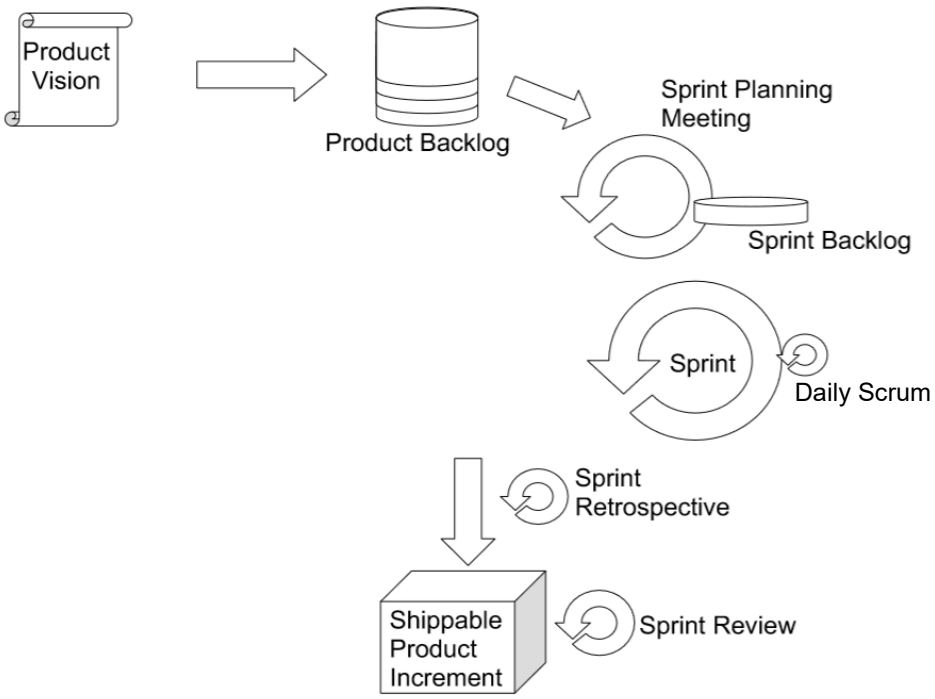


Abbildung 6: Übersicht über den Scrum Prozess, Quelle: Goll/Hommel (2015), S. 87.

Nachdem die Rollen des Scrum Prozesses geklärt sind, wird nun auf den eigentlichen Prozess und dessen Begrifflichkeiten eingegangen. In Abbildung 6 ist der generelle Scrum Prozess illustriert. Die Entwicklung eines Produktes beginnt dabei immer mit einer Product Vision. Diese beschreibt das generelle Ziel und den Zweck einer Entwicklung. Sie gibt die Richtung vor. Ist die Product Vision bekannt, werden alle

[29] Vgl. Goll/Hommel (2015), S. 88 – 91.

Anforderungen im Product Backlog gesammelt. Diese werden mit Prioritäten versehen und sind für das gesamte Scrum Team einsehbar. Nur die hoch priorisierten Anforderungen werden detailliert beschrieben, da sich die restlichen bis zu ihrer Durchführung noch ändern können. Des Weiteren ist es wichtig den Aufwand der einzelnen Einträge zu schätzen. Im Sprint Planning Meeting werden die Anforderungen mit der höchsten Priorität und diejenigen, die im Zuge dieser Anforderungen auch einfach erledigt werden können, vom Product Backlog in den Sprint Backlog übertragen. Im Sprint Backlog sind immer nur diejenigen Anforderungen enthalten, die innerhalb eines Sprints erfüllt werden können. Ein Sprint ist ein Zyklus bzw. eine Zeiteinheit, in der die Arbeitspakete, welche im Sprint Backlog definiert sind, zu einem potenziell auslieferbaren Produkt umgesetzt werden. Ein Arbeitspaket sollte möglichst klein, im Idealfall in einem Tag bearbeitbar sein. [30]

Neben dem bereits beschriebenen Sprint Planning Meeting gibt es noch drei weitere Meetings, welche Scrum vorschreibt. Am Ende eines jeden Sprints wird ein Sprint Review Meeting abgehalten. In diesem werden die Ergebnisse des letzten Sprints, welche als Produktinkrement vorhanden sind, begutachtet und die nächsten Schritte besprochen. Beim Sprint Retrospective, welches auch nach jedem Sprint durchgeführt wird, diskutiert der Scrum Master mit dem Entwicklungsteam den vergangenen Sprint. Aus der Analyse werden Maßnahmen für eine bessere Abwicklung der nächsten Sprints abgeleitet. Das Daily Scrum ist ein maximal 15-minütiges Meeting, welches täglich abgehalten wird. Es dient dem Informationsaustausch über den aktuellen Stand und Fortschritt unter den Entwicklungsteammitgliedern und dem Scrum Master, sowie der Planung des nächsten Tages. Der Product Owner nimmt an dem Daily Scrum nicht teil. [31]

2.6 Conclusio

In diesem Kapitel 2 werden verschiedene Vorgehensweisen in der Reihenfolge der zeitlichen Entwicklung für die Erstellung einer Software oder eines Systems, beschrieben. Das älteste Modell, das Wasserfallmodell, ist sehr starr aufgebaut. Das Grundprinzip, die Einteilung der gesamten Entwicklung einer Software in Phasen, wird heute noch immer angewendet. Diese geben einem Projekt einen geordneten Rahmen. [32] Jedoch ist der starre Ablauf in der Praxis selten anwendbar und nicht wirklich zielführend. Das realistische Phasenmodell, bei dem die Phasen zeitlich überlappend sind und jede Phase mit einem Meilenstein abgeschlossen wird, ist in der Praxis schon eher umsetzbar. Mit dem V-Modell, vor allem dem neuesten V-Modell XT, welches eine Weiterentwicklung des Phasenmodells dargestellt, wurde ein relativ flexibles, jedoch auch umfangreiches Modell entwickelt. Der wesentliche Vorteil des V-Modells XT besteht darin, dass das gesamte Modell auf einem Metamodell aufbaut und dadurch an konkrete Projekte angepasst werden kann. Es gibt noch eine Reihe von alternativen Vorgehensmodellen, die die Phasenorientierung generell in Frage stellen. Ein Beispiel dafür ist die iterative Entwicklung, bei der jede Iteration einen vollständigen Entwicklungsschritt darstellt. Am Ende dieses Kapitels werden die agilen Prozesse, mit dem am weitesten verbreiteten Vertreter Scrum, vorgestellt. Alle agilen Prozesse zeichnen sich durch deren Leichtigkeit und

[30] Vgl. Goll/Hommel (2015), S. 87 – 100.

[31] Vgl. Goll/Hommel (2015), S. 97 f.

[32] Vgl. Sandhaus/Berg/Knott (2014), S. 41.

Flexibilität bei der Entwicklung einer Software oder eines Systems aus, z.B. steht hier eine funktionierende Software über einer umfassenden Dokumentation.

Bei dieser Masterarbeit fließen Aspekte verschiedener Modelle ein, jedoch orientiert sie sich generell am V-Modell XT. Dieses Modell passt am besten zu den Herausforderungen dieser Arbeit, da es relativ flexibel ist, aber doch eine grobe Richtung vorgibt. Bei agilen Projekten kann aufgrund der fehlenden Definition des Projektendes zu einem bestimmten Zeitpunkt nur ein variabler Leistungsumfang zugesichert werden. [33] Aus diesem Grund sind diese aktuelleren Methoden für diese Arbeit nicht besonders gut anwendbar. So werden im Kapitel 3 die theoretischen Grundlagen des absteigenden Astes des V-Modells beschrieben. Wie weiter oben erwähnt, muss jeder Anforderung ein entsprechender Testschritt zugeordnet werden. Die Theorie zu diesem Thema wird im Kapitel 4, Test Engineering, behandelt. Dieses Kapitel wird dem aufsteigenden Ast des V-Modells zugeordnet. Zusätzlich wird im Unterkapitel 3.4 die Theorie der Modellbildung, welche sowohl das Requirements- und Test Engineering, wie auch die Realisierung der Software (siehe unteren Bereich der Abbildung 4) unterstützt, beschrieben.

[33] Vgl. Stephens/Rosenberg (2003), S. 81.

3 REQUIREMENTS ENGINEERING

Dieses Kapitel beschäftigt sich mit den theoretischen Grundlagen, die das Requirements Engineering betreffen. Anfangs werden generellen Grundlagen, wie z.B. Begriffsdefinitionen, erläutert. Im Unterkapitel 3.2 werden die vorbereitenden Tätigkeiten, die für ein erfolgreiches Requirements Engineering, notwendig sind, beschrieben. In den Unterkapiteln 3.3 – 3.6 wird auf die Kerntätigkeiten des Requirements Engineerings eingegangen. Diese setzen sich aus dem Erheben, der Detaillierung, der Prüfung und Verwaltung der Anforderungen zusammen. Im Zuge des Unterkapitels 3.4 wird auch auf die Theorie der Modellbildung mittels UML, welche die Hauptdokumentationsart dieser Masterarbeit darstellt, näher eingegangen.

3.1 Grundlagen

In diesem Unterkapitel werden die theoretischen Grundlagen, welche für das Verständnis der weiteren Arbeit notwendig sind, näher erläutert. Zuerst wird erklärt, was generell unter dem Begriff Requirements Engineering verstanden wird und welche Tätigkeiten zum Requirements Engineering gezählt werden. Des Weiteren wird hier auch beschrieben, welche Vorbereitungen getroffen werden müssen, um mit den eigentlichen Tätigkeiten des Requirements Engineerings starten zu können. Danach wird ein Überblick über die Anforderungen und deren Einteilung gegeben. Am Schluss dieses Unterkapitels werden die Kriterien für qualitativ hochwertige Anforderungen dargelegt.

3.1.1 Definition Requirements Engineering

Unter Requirements Engineering, zu Deutsch *Anforderungsmanagement,* wird von verschiedenen Personen oft nicht dasselbe verstanden. Die unterschiedlichen Sichtweisen können sich auf Begriffe, Konzepte, Vorgehensweisen, Rollen und Aufgaben beziehen. Aus diesem Grund ist es wichtig, zu Beginn eines Projektes ein gemeinsames Verständnis unter den Projektmitarbeitern herzustellen. Als Orientierungshilfe können Standards dienen: [34]

Definition des Requirements Engineering laut dem IREB (International Requirements Engineering Board): [35]

- *„Die relevanten Anforderungen zu kennen, Konsens unter den Stakeholdern über die Anforderungen herzustellen, die Anforderungen konform zu vorgegebenen Standards zu dokumentieren und die Anforderungen systematisch zu managen.*
- *Die Wünsche und Bedürfnisse der Stakeholder zu verstehen und zu dokumentieren.*
- *Die Anforderungen zu spezifizieren und zu managen, um das Risiko zu minimieren, ein System auszuliefern, das nicht den Wünschen und Bedürfnissen der Stakeholder entspricht.“*

IREB ist eine Nonprofitorganisation, die sich mit Requirements Engineering beschäftigt und auch Personenzertifizierungen anbietet. [36] Zusammengefasst sagt diese Definition aus, dass das Requirements

[34] Vgl. Krallmann/Dockter/Ritter (2017), S. 15.

[35] Krallmann/Dockter/Ritter (2017), S. 15.

[36] Vgl. IREB (2018a), Online-Quelle [27.November.2018].

Engineering dafür verantwortlich ist, eine Software oder ein System zur richtigen Zeit, mit der richtigen Qualität, im vorgegebenen Budgetrahmen zu liefern.

In Abbildung 7 sind die Tätigkeiten, die für die Erreichung dieser Ziele notwendig sind, dargestellt. Hier ist zu sehen, dass das Requirements Engineering vier Haupttätigkeiten umfasst. Die erste Tätigkeit *Erheben* befasst sich mit der Sammlung von Wünschen und Informationen der Stakeholder. Diese werden als Ziele und Anforderungen formuliert. Die dazugehörigen möglichen Methoden sind nicht grau hinterlegt dargestellt. Die zweite Tätigkeit *Dokumentieren* geht auf die Dokumentation der Anforderungen ein. Diese kann aus Prosa-Texten oder auch verschiedenen Diagrammen bestehen. Darüber hinaus umfasst diese Tätigkeit die Detaillierung der groben Anforderungen in genauere Spezifikationen. Sind die Anforderungen ausformuliert, müssen sie auf gewisse Qualitätskriterien überprüft werden. Nicht nur die Anforderungen, sondern auch die Software selbst muss geprüft werden. Mit diesen Themen befasst sich die Tätigkeit *Prüfen*. Als letzte Tätigkeit im Zuge des Requirements Engineerings ist das *Verwalten* angeführt. Darunter werden Maßnahmen verstanden, die gewährleisten, dass alle Anforderungen nachvollziehbar und deren Herkunft bekannt sind.

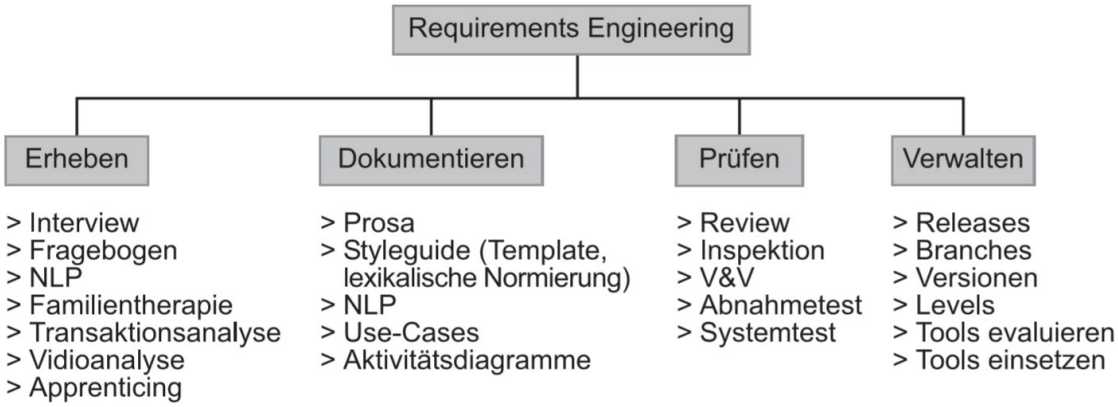


Abbildung 7: Tätigkeiten und Methoden des Requirements Engineerings, Quelle: Rupp/SOPHIST GROUP (2007) , S. 14.

Bevor mit den Tätigkeiten des Requirements Engineerings, welche in Abbildung 7 dargestellt werden, begonnen werden kann, müssen jedoch die richtigen Rahmenbedingungen geschaffen werden. Welche Kriterien zu beachten sind, wird im Unterkapitel 3.2 erläutert. In den darauffolgenden Unterkapiteln wird auf die vier Haupttätigkeiten des Requirements Engineerings näher eingegangen. Wie bereits im oberen Absatz beschrieben, werden im Zuge des Requirements Engineerings nicht nur die Anforderungen, sondern auch die Software selbst geprüft. Somit kann das Kapitel 4 – Test Engineering ebenfalls in die Haupttätigkeit *Prüfen* eingegliedert werden.

3.1.2 Definition Anforderungen

In diesem Unterkapitel werden die Anforderungen, welche auch im Unterkapitel 3.1.1 erwähnt werden, näher erläutert. Auch hier können wieder Standards als Vorlage dienen. Die erste Definition, welche im folgenden Absatz angeführt ist, ist die Originalversion, welche bereits 1990 veröffentlicht wurde. Diese besitzt grundlegend noch heute ihre Gültigkeit, wurde von der Norm IEEE 610.12-1990 übernommen und wird auch im aktuellen CPRE Glossar angeführt. Die Abkürzung CPRE steht für *Certified Professional for Requirements Engineering* und ist ein Personenzertifikat, welches von der Nonprofitorganisation IREB

entwickelt wurde. [37] In diesem Glossar befindet sich auch eine alternative, neuere und auch einfachere Definition einer Anforderung, welche im übernächsten Absatz beschrieben wird.

Definition der Anforderung nach IEEE 610.12-1990: [38]

- *„Eine Bedingung oder Fähigkeit, die von einem Benutzer (Person oder System) zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird.*
- *Eine Bedingung oder Fähigkeit, die ein System oder Teilsystem erfüllen oder besitzen muss, um einen Vertrag, eine Norm, eine Spezifikation oder andere, formell vorgegebene Dokumente zu erfüllen.*
- *Eine dokumentierte Repräsentation einer Bedingung oder Eigenschaft gemäß den zwei vorherigen Punkten.“*

Alternative, modernere Definition der Anforderungen laut des aktuellen CPRE Glossars: [39]

- *„Ein Erfordernis, welches ein Stakeholder stellt.*
- *Eine Fähigkeit oder Eigenschaft, die ein System haben soll.*
- *Eine dokumentierte Repräsentation eines Erfordernisses, einer Fähigkeit oder einer Eigenschaft.“*

Kurz zusammengefasst bedeuten diese zwei Definitionen, dass eine Fähigkeit oder Eigenschaft, welche ein System haben soll, den Vorstellungen der Stakeholder entsprechen und dementsprechend dokumentiert werden soll. Der Begriff Stakeholder wird im Unterkapitel 3.2.1 näher beschrieben.

3.1.3 Einteilung der Anforderungen

Um die oben beschriebenen groben Definitionen zu verfeinern, lassen sich die Anforderungen in sieben Gruppen einteilen: [40]

- **Funktionale Anforderungen:** funktionale Anforderungen beschreiben Tätigkeiten des Systems, welche entweder komplett selbstständig oder in der Interaktion mit einem Nutzer oder anderen Systemen (Eingaben, Ausgaben) ausgeführt werden sollen. Manchmal werden sie auch in Verhaltensdiagrammen dargestellt und deswegen als Verhaltensanforderungen bezeichnet. [41]
- **Technische Anforderungen:** Im Gegensatz zu den eher fachlich motivierten funktionalen Anforderungen, werden unter technischen Anforderungen, Ansprüche an die Hardware, die Architektur oder die Programmiersprachen gestellt. Technische Anforderungen sind oft notwendig, weil ein System in ein bestehendes technisches Umfeld eingebettet werden muss.
- **Anforderungen an die Benutzerschnittstelle:** Zum einen wird unter diesem Begriff die klassische Mensch-Maschine-Schnittstelle (HMI) eingeordnet. Diese beschreibt die Form und Funktion

[37] Vgl. IREB (2018b), Online-Quelle [27.November.2018].

[38] Pohl/Rupp (2015), S. 3.

[39] Glinz (2014), Online-Quelle [27.November.2018], S. 17.

[40] Vgl. Rupp/SOPHIST GROUP (2007), S. 15 ff.

[41] Vgl. Davis (1993), S. 498.

von Ein- und Ausgabegeräten, sowie die grafischen und funktionalen Anforderungen an die Benutzeroberfläche. Zum anderen zählen Schnittstellen zu angrenzenden Systemen zu dieser Kategorie, wenn die Systeme als Benutzer agieren.

- **Qualitätsanforderungen:** Mit den Qualitätsanforderungen sind alle Anforderungen, welche die Qualität oder auch Güte des Produktes, des Prozesses oder der am Prozess beteiligten Personen betreffen, gemeint. Sie können auch Dienstgüte- oder Dienstqualitäts-Anforderungen genannt werden. Diese beziehen sich aber anders als die Qualitätsanforderungen nur auf Produkte und Systeme, nicht jedoch auf Prozesse und Personen. Die DIN EN ISO 66272 unterteilt die Dienstqualitäts-Anforderungen in die fünf Merkmale Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit.
- **Anforderungen an sonstige Lieferbestandteile:** Wird ein System oder eine Software im Zuge eines Projektes entwickelt, kann dem Kunden üblicherweise nicht nur ein Source Code geliefert werden. Die zusätzlichen Bestandteile, wie z.B. das Benutzerhandbuch, zählen zu den Anforderungen an sonstige Lieferbestandteile. Unter den genannten Anforderungen können aber auch Anforderungen an das Benutzerhandbuch oder ähnlichen Dokumenten selbst verstanden werden.
- **Anforderungen an die Durchführung der Entwicklung und Einführung:** Die Anforderungen an den Entwicklungsprozess an sich und die Zusammenarbeit mit den Stakeholdern werden unter dem Begriff *Anforderungen an die Durchführung der Entwicklung und Einführung* zusammengefasst. So schreiben z.B. manche Kunden vor, wie der Entwicklungsprozess auszusehen hat oder wann welche Tests durchgeführt werden müssen. Auch Vorschriften wie das Einhalten verschiedener Standards oder die Verwendung bestimmter Werkzeuge wird zu dieser Anforderungsgruppe gezählt. Die Anforderungen an die Durchführung der Entwicklung und Einführung werden üblicherweise nicht in einer Anforderungsspezifikation, sondern in einem Projekthandbuch oder Vereinbarungen zur Zusammenarbeit zwischen Auftraggeber und Auftragnehmer angeführt.
- **Rechtlich-vertragliche Anforderungen:** Wie der Name bereits sagt, handelt es sich hierbei um Anforderungen, welche rechtlich bindend sind und üblicherweise in den Verträgen zwischen Auftraggeber und Auftragnehmer festgehalten werden. Es kann sich um Zahlungsmeilensteine, Umgang mit Änderungen, Liefermodalitäten usw. handeln.

Im Zuge der Anforderungen, fällt oft der Begriff der *nicht-funktionalen Anforderungen*. Unter dieser Bezeichnung werden alle Anforderungen, welche nicht als funktionale Anforderungen klassifiziert werden, zusammengefasst. Dazu zählen die technischen Anforderungen, die Anforderungen an die Benutzerschnittstelle, die Qualitätsanforderungen, die Anforderungen an sonstige Lieferbestandteile, die Anforderungen an die Durchführung der Entwicklung und Einführung und die rechtlich-vertraglichen Anforderungen. Oftmals werden die funktionellen Anforderungen als die wichtigsten angesehen. Der Grund dafür ist, dass diese meistens bekannt und am einfachsten zu beschreiben sind. Jedoch ist es unumgänglich, dass auch die nicht-funktionalen Anforderungen mit derselben Wichtigkeit betrachtet werden. Speziell die Qualitätsanforderungen und die Anforderungen an die Benutzerschnittstelle tragen zur Kundenzufriedenheit wesentlich bei. Zahlreiche Untersuchungen, wie z.B. von Charette [42]

[42] Vgl. Charette (1990).

zeigen, dass Stakeholder über fehlende Funktionen und Features hinwegsehen, wenn nur die Qualitätserwartungen erfüllt werden. [43]

Des Weiteren können die Anforderungen nach dem Modell von Kano eingeteilt werden. Dieses zeigt den Zusammenhang zwischen den Kundenanforderungen und der Kundenzufriedenheit. Das Kano-Modell basiert auf der Motivationstheorie von Herzberg. Herzberg unterscheidet zwischen Hygiene- und Motivationsfaktoren. Die Erfüllung der Hygienefaktoren führt zur Beseitigung der Unzufriedenheit, jedoch wird damit nicht automatisch die Zufriedenheit erreicht. Damit sich Zufriedenheit einstellt, müssen zusätzlich bestimmte Motivationsfaktoren erfüllt werden. Mit seiner Motivationstheorie sagt Herzberg generell aus, dass die Zufriedenheit nicht eindimensional, sondern als mehrdimensionales Konstrukt gesehen werden muss. Diese Mehrdimensionalität spiegelt sich im Kano-Modell durch die Einteilung der Kundenanforderungen in drei Dimensionen wider. [44]

Das Wissen, welche Bedeutung die einzelnen Anforderungen für die Stakeholder haben, ist für die Anforderungsermittlung von großer Bedeutung. Jede Anforderung kann in eine der drei Dimensionen, welche in Abbildung 8 ersichtlich sind und im Folgenden beschrieben werden, eingeteilt werden. [45]

- **Basisanforderungen:** Basisanforderungen sind unterbewusste Anforderungen, die mit Hygienefaktoren vergleichbar sind, d.h. dass die Kundenzufriedenheit mit diesen Faktoren nicht wirklich erhöht werden kann. Jedoch steigt bei Nichterfüllung die Unzufriedenheit massiv an. Somit müssen Basisfaktoren immer vollständig erfüllt werden. Diese Faktoren werden vor allem durch bereits vorhandene Systeme geprägt.
- **Leistungsanforderungen:** Unter Leistungsanforderungen werden die bewussten Anforderungen verstanden, die durch einfache Befragungen an die Stakeholder herausgefunden werden können. Zwischen Erfüllung dieser Anforderungen und der Zufriedenheit gibt es einen linearen Zusammenhang. Um eine möglichst hohe Kundenzufriedenheit zu erreichen, ist es erstrebenswert möglichst alle Leistungsanforderungen zu erfüllen. Jedoch wird der Kunde das Produkt auch akzeptieren, wenn nicht alle Wünsche erfüllt werden.
- **Begeisterungsfaktoren:** Anforderungen, welche der Kunde nicht unbedingt benötigt, aber eine hohe Zufriedenheit zur Folge haben, werden als Begeisterungsfaktoren bezeichnet. Sie zählen zu den unbewussten Anforderungen. Den Wert dieser Anforderungen erkennt der Stakeholder meist erst, wenn er sie selbst ausprobieren kann oder sie vom Requirements Engineer vorgeschlagen bekommt.

[43] Vgl. Rupp/SOPHIST GROUP (2007), S. 256 ff.

[44] Vgl. Jochem (2010), S. 27 – 54.

[45] Vgl. Pohl/Rupp (2015), S. 24 f., zitiert nach Kano/u.a. (1984).

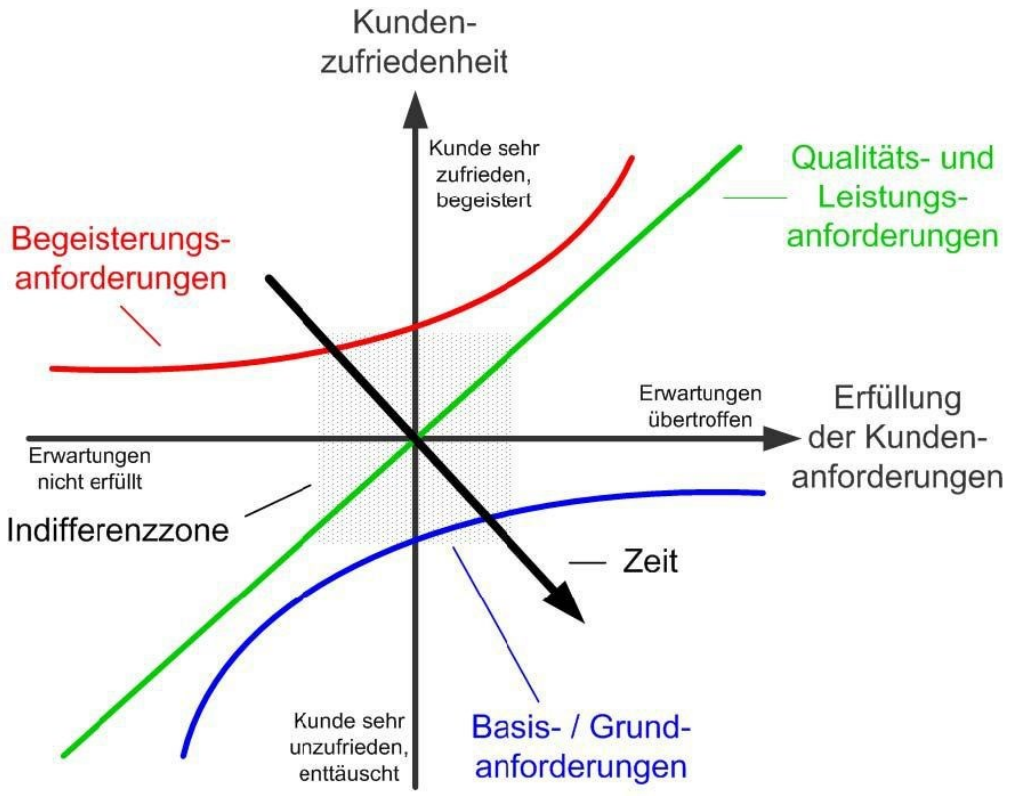


Abbildung 8: Kano-Modell, Quelle: Carl Hanser Verlag (2018), Online-Quelle [27.November.2018].

3.1.4 Qualitätskriterien

Um eine gute Ausgangsbasis für die weitere Entwicklung einer Software oder eines Systems zu schaffen, müssen die Anforderungen gewissen Qualitätskriterien entsprechen. Erstens sollte jede Anforderung *abgestimmt* sein. Das bedeutet, dass alle Stakeholder die Anforderung als korrekt und notwendig empfinden. Zweitens muss jede Anforderung eindeutig und vollständig beschrieben sein, es darf kein Interpretationsspielraum vorhanden sein. Des Weiteren muss eine Anforderung notwendig sein. Sie muss relevant sein und darf sich nicht auf etwas beziehen, dass die aktuelle Entwicklung nicht betrifft. Ein weiteres wichtiges Qualitätskriterium ist die Konsistenz. Das bedeutet, dass eine Anforderung in sich und gegenüber anderen Anforderungen keine Widersprüche enthalten darf. Der sechste Anspruch an die Anforderungen betrifft die Prüfbarkeit. Eine Anforderung ist nur sinnvoll, wenn deren Erfüllung durch einen Test oder durch eine Messung nachgewiesen werden kann. Es macht auch keinen Sinn, wenn Anforderungen an die Software oder an das System gestellt werden, die den zeitlichen oder budgetären Rahmen sprengen oder technisch nicht realisierbar sind. Wenn das der Fall ist, können sie in dieser Phase des Requirements Engineerings von Stakeholdern zurückgenommen werden. Ein weiterer wichtiger Punkt ist die Verfolgbarkeit. Das bedeutet, dass sowohl der Ursprung, wie auch die Umsetzung und Beziehung zu anderen Dokumenten nachvollziehbar sein muss. Das kann beispielsweise durch einen Anforderungsidentifikator sichergestellt werden. Als letztes und neuntes Qualitätskriterium ist die Verständlichkeit einer Anforderung zu beachten. Die Anforderungen müssen so formuliert sein, dass sie von allen Stakeholdern verstanden werden. Die Qualitätskriterien *eindeutig*, *notwendig, konsistent, prüfbar, realisierbar, verfolgbar* und *verständlich* werden auch in der ISO/IEC 29148:2011 angeführt. Darüber hinaus gibt es noch Empfehlungen wie z.B., dass einfache und kurze Sätze verwendet werden sollen oder unterschiedliche Qualitätskriterien an das

Anforderungsdokument eingehalten werden sollen. Wenn Anforderungen die beschriebenen Qualitätskriterien erfüllen, stellen sie eine methodisch korrekte und aussagekräftige Basis für das weitere Vorgehen dar. [46]

3.2 Rahmenbedingungen schaffen

Um problemlos mit den Haupttätigkeiten des Requirements Engineerings, angefangen mit dem Erheben der Anforderungen, beginnen zu können, müssen im Vorhinein die richtigen Rahmenbedingungen geschaffen werden. Wie in Abbildung 9 illustriert, müssen im Zuge der Vorbereitungen die Ziele klar definiert, die Stakeholder (am Projekt beteiligten Personen) identifiziert und die Systemgrenzen festgelegt werden. [47] In den folgenden Unterkapiteln wird beschrieben, wie diese Ziele erreicht werden können. Im ersten Unterkapitel wird der Begriff des Stakeholders erläutert und auch beschrieben, welche Daten von den Stakeholdern erhoben werden sollen. Das darauffolgende Unterkapitel 3.2.2 beschäftigt sich mit Zielen. Es beschreibt den Prozess der Zielerhebung, wie auch deren Dokumentation. Im letzten Teil dieses Unterkapitels werden die theoretischen Grundlagen zur Abgrenzung des Systemumfangs veranschaulicht.

Abbildung 9: Tätigkeiten im Vorfeld der Anforderungsermittlung, Quelle: Partsch (2010), S. 40.

3.2.1 Stakeholder

Der englische Begriff *Stakeholder* wird in das Deutsche mit *Beteiligte* übersetzt. Genauer gesagt beschreibt er eine Person oder Organisation, welche direkten oder indirekten Einfluss auf die Anforderungen des betrachteten Systems hat. [48] Darunter fallen sowohl Vertreter des Auftraggebers und Endanwenders, wie auch Entwickler und Tester des Systems. Da die meisten Anforderungen von den unterschiedlichen Stakeholdern aufgebracht werden, ist es wichtig, alle Stakeholder frühzeitig zu identifizieren und in das Projekt einzubinden. Je später im Entwicklungsprozess eine Anforderung geändert bzw. nachgereicht wird, desto schwieriger und teurer wird es, die Software darauf anzupassen. [49]

[46] Vgl. Pohl/Rupp (2015), S. 47 f.

[47] Vgl. Rupp/SOPHIST GROUP (2007), S. 86.

[48] Vgl. Glinz (2014), Online-Quelle [27.November.2018], S. 20.

[49] Vgl. Krallmann/Dockter/Ritter (2017), S. 16.

Um die beschriebenen Ziele bezogen auf die Stakeholder zu erreichen, sollen laut IREB mindestens folgende Informationen jedes Stakeholders gesammelt und niedergeschrieben werden: [50]

1. *„Name*
2. *Funktion (Rolle)*
3. *Weitere Personen- und Kontaktdaten*
4. *Zeitliche und räumliche Verfügbarkeit während der Projektlaufzeit*
5. *Relevanz des Stakeholders*
6. *Sein Wissensgebiet und -umfang*
7. *Seine Ziele und Interessen bezogen auf das Projekt“*

Neben dem Namen des Stakeholders ist es wichtig, die generelle Funktion und auch die Rolle im Projekt festzuhalten. So kann z.B. der Vorgesetzte des Projekt Managers die Rolle Test User im aktuellen Projekt ausfüllen. Ist die zeitliche und räumliche Verfügbarkeit der Stakeholder bekannt, wird der Zeitplan eher der Realität entsprechen, als wenn das nicht der Fall ist. Auch die Relevanz der einzelnen Stakeholder soll geklärt werden. Widersprechen sich z.B. die Vorstellungen der Stakeholder in einem Punkt und es ist nicht möglich einen Konsens zu erreichen, wird die Meinung des relevanteren Stakeholders bevorzugt werden. Des Weiteren können durch die Kenntnis des Wissensgebietes und der Interessen der Stakeholder die Anforderungen und Meinungen einfacher eingeordnet werden. Es empfiehlt sich eine Liste mit allen Stakeholdern und den oben beschriebenen Informationen anzulegen und zu pflegen. Außerdem ist es empfehlenswert die Stakeholder nach weiteren Personen und Organisationen, welche in das Projekt eingebunden werden sollen, zu befragen.

3.2.2 Ziele

„Unter einem Ziel wird ein erstrebenswerter Zustand verstanden, der in der Zukunft liegt und dessen Eintritt von bestimmten Handlungen bzw. Unterlassungen abhängig ist, der also nicht automatisch eintritt.“ [51] Durch diese Definition wird deutlich, dass die Festlegung von Zielen großen Einfluss auf die erfolgreiche Entwicklung eines Produktes hat. Die Ziele stellen die Ausgangsbasis für die anschließende Anforderungsanalyse dar. Sind die Ziele nicht klar definiert, hat der Requirements Engineer keine Vorgabe, welche Visionen in die Spezifikation der Anforderungen einfließen sollen. In Folge dessen kann es passieren, dass der Requirements Engineer und das gesamte Entwicklungsteam monatelang auf ein falsches Ziel hinarbeiten. Somit haben Ziele auch einen wesentlichen Einfluss auf die Motivation der Mitarbeiter. Darüber hinaus ist es wichtig, die Ziele nicht nur zu definieren, sondern auch ausreichend zu dokumentieren. Geschieht das nicht, sind sie nicht verfolgbar und werden meistens nicht erreicht. Im Endeffekt muss jede Anforderung einem Ziel zugeordnet werden. [52]

Beim Prozess der Zieldefinition wird in der Literatur zwischen fünf Tätigkeiten unterschieden: [53]

---

[50] Krallmann/Dockter/Ritter (2017), S. 16.

[51] Rupp/SOPHIST GROUP (2007), S. 86, zitiert nach: Gernert/Ahrend (2000).

[52] Vgl. Rupp/SOPHIST GROUP (2007), S. 86 f.

[53] Vgl. Rupp/SOPHIST GROUP (2007), S. 87 – 90.

- **Systematische Suche nach den Stakeholdern:** Wie im Unterkapitel 3.2.1 beschrieben, handelt es sich bei Stakeholdern um Personen oder Organisationen, die vom aktuellen Projekt betroffen sind. Die richtigen Personen mit den richtigen Aufgaben zur richtigen Zeit betraut zu machen zählt zu einem der wichtigsten Erfolgsrezepte. Am Anfang des Requirements Engineerings ist es schwer möglich alle relevanten Stakeholder einzubinden, da erst die spätere Analyse und Systemabgrenzung weiteren Aufschluss über die Stakeholder geben. Das bedeutet, dass die Suche nach Stakeholdern mit dieser Phase nicht abgeschlossen ist.
- **Erhebung der Ist-Situation:** Hier stellt sich die Frage, ob bereits ein Vorgänger des zu entwickelnden Systems besteht. Ist das der Fall, soll die bestehende Dokumentation so wertungsfrei wie möglich analysiert werden. Ist keine Dokumentation vorhanden, kann der Ist-Stand mithilfe von vergangenheitsorientierten Ermittlungstechniken, mit denen hauptsächlich die Basisanforderungen ermittelt werden, erhoben werden. Diese werden im Unterkapitel 3.3.1 erläutert. Eine Analyse der Ist-Situation ist auch sinnvoll, wenn die zu automatisierende Tätigkeit bisher manuell ausgeführt wurde.
- **Bewertung der Ist-Situation:** Da sich das neue System vom bestehenden positiv abheben soll, müssen die existierenden Probleme ermittelt und Optimierungspotentiale aufgezeigt werden. Hier sollen auch die Visionen über zukünftige Marktpotentiale miteinfließen. Bei diesem Prozessschritt muss speziell darauf geachtet werden, dass die Ursache des jeweiligen Problems gefunden und eliminiert wird, und nicht nur die Symptome bekämpft werden.
- **Ableitung von Zielen:** Mit einer fundierten Kenntnis der Ausgangssituation, der Stakeholder und der Optimierungspotentiale können nun die Ziele formuliert werden. Die erhobenen Ziele und Rahmenbedingungen müssen exakt definiert und dokumentiert werden. Die beschriebenen Ziele sollten vor allem gut kommunizierbar sein. Jedem Ziel muss bereits in diesem Stadium ein Urheber, im Normalfall ist das ein Stakeholder, zugewiesen werden. Welchen Qualitätskriterien Ziele entsprechen und wie sie formuliert werden sollen, wird im folgenden Absatz beschrieben.
- **Ziele bewerten:** Der letzte Punkt des Zieldefinitionsprozesses betrifft die Bewertung der ermittelten und dokumentierten Ziele. Hier soll bewertet werden, ob es sich überhaupt lohnt das Projekt durchzuführen. Es stellt sich die Frage, ob der Nutzen größer ist als die Kosten, die die Entwicklung des Systems und deren Wartung mit sich bringen.

Nachdem die Stakeholder ermittelt und befragt wurden und die Ist-Situation erhoben wurde, müssen die Ziele beschrieben werden. Nicht nur fehlende und falsche Ziele, sondern auch zweideutige oder schlecht formulierte Ziele gefährden den Erfolg eines Projektes. Es gibt einige Kriterien zu beachten, um von korrekten und gut formulierten Zielen sprechen zu können. Unterkapitel 3.1.4 beschreibt die Qualitätskriterien der Anforderungen. Wie Anforderungen, sollen auch Ziele *vollständig, korrekt, abgestimmt, konsistent, prüfbar, realisierbar, notwendig, eindeutig* und *aktuell* sein. Zusätzlich gibt es zwei weitere Qualitätsmerkmale, die jedes Ziel aufweisen sollte. Das erste Merkmal ist die *Lösungsneutralität*. Das heißt, dass die Formulierung von Zielen keinen Hinweis auf eine Lösung enthalten soll. Ist das nicht der Fall, wird von vornherein der Lösungsweg eingeschränkt und andere, vielleicht bessere Lösungen, nicht mehr in Betracht gezogen. Das zweite Merkmal bezieht sich auf die *einschränkenden Rahmenbedingungen.* Falls z.B. vom

Kunden explizit ein Excel-Interface gefordert wird, kann nicht von einem offenen Interface die Rede sein, sondern muss das Excel-Interface in der Zieldefinition dokumentiert werden. [54]

3.2.3 Umfang und Kontext des Systems

Um den Leistungsumfang eines Systems abgrenzen zu können, wird der Begriff der Systemgrenze und des Systemkontexts eingeführt. In diesem Unterkapitel wird zuerst beschrieben, wie ein System definiert ist und wie es eingeteilt werden kann, danach wird auf die Systemumgebung, auch Kontext genannt, näher eingegangen.

„Unter einem System versteht man eine Einheit von Komponenten (atomare Bestandteile oder Subsysteme), die nach einem bestimmten Kriterium von ihrer Umgebung abgegrenzt sind, über Beziehungen verknüpft sind oder miteinander interagieren und der Erreichung eines bestimmten Ziels oder Zwecks dienen." [55] Ein System wird durch zwei Merkmale charakterisiert, Struktur und Verhalten. Unter Struktur wird die Gesamtheit aller Komponenten und deren Beziehungen verstanden. Das Systemverhalten beschreibt das Zusammenwirken der Komponenten zur Erreichung der Systemziele und des Systemzwecks. [56]

Ein System kann sehr unterschiedliche Ausprägungen annehmen, wie beispielsweise ökologische, soziale oder ökonomische Systeme zeigen. Das System, welches in dieser Masterarbeit behandelt wird, wird als softwaregestütztes System bezeichnet. Ein softwaregestütztes System, ist ein System, bei dem zumindest eine Komponente aus einer Software besteht. Elektronik oder Mechanik sind zwei Beispiele für die anderen Komponenten. Softwaregestützte Systeme können nach verschiedenen Kriterien klassifiziert werden. Für das Requirements Engineering ist eine relativ grobe Einteilung ausreichend: [57]

- **Technisches System:** Ein technisches System ist ein System, das hauptsächlich aus technischen Artefakten besteht. Menschen spielen hier eine untergeordnete Rolle. Die Software in diesem System dient meist zur Steuerung und Kontrolle anderer Komponenten. Bekannte Beispiele für diese Systeme sind Prozessautomatisierungs-, Kommunikations-, Multimedia- oder auch eingebettete Systeme.
- **Organisatorisches System:** Bei organisatorischen Systemen spielt der Mensch eine zentrale Rolle. Die Softwarekomponente eines solchen Systems beschafft, verarbeitet und stellt üblicherweise Informationen über organisatorische Strukturen bereit. Typische Beispiele für organisatorische Systeme sind Buchhaltungs-, Abrechnungs- oder Informationssysteme.
- **Reine Softwaresysteme:** Reine Softwaresysteme werden auch als geschlossene Systeme bezeichnet und zeichnen sich dadurch aus, dass sie außer Eingaben und Ausgaben keine Interaktionen mit der Umgebung durchführen. Numerische Software oder Simulationssoftware sind Vertreter dieser Klasse.

[54] Vgl. Rupp/SOPHIST GROUP (2007), S. 100 f.

[55] Partsch (2010), S. 23.

[56] Vgl. Partsch (2010), S. 23.

[57] Vgl. Partsch (2010), S. 23 f.

Nachdem geklärt wurde, was ein System ist, wird im Folgenden die Abgrenzung zwischen dem System und dem Systemkontext beschrieben. Um den Rahmen der Systementwicklung abstecken zu können, müssen die Systemgrenzen definiert werden. Alles was sich innerhalb des Systems befindet, wird in den Zielen und Anforderungen behandelt. Somit ist der Lieferumfang, auch *Scope* genannt, wie er in Abbildung 10 bezeichnet wird, definiert. Der Teil, welcher sich außerhalb des Systems befindet, wird zwar in den Zielen nicht behandelt, soll zwecks des besseren Verständnisses aber trotzdem erläutert werden. Dieser Teil wird Systemkontext genannt. Unter Systemkontext werden alle Organisationen, Nachbarsysteme, Nachbarkomponenten usw. verstanden, die nicht zum System dazugehören, aber Einfluss auf das System haben. Die Grenze zwischen dem System und dem Kontext wird als Systemgrenze bezeichnet. Auch die Systemumgebung kann nicht unendlich groß sein. Diese wird mit der Kontextgrenze, welche auch in Abbildung 10 zu sehen ist, beschränkt. Die Kontextabgrenzung soll aber nur einen groben Rahmen für die weitere Anforderungsanalyse definieren. Im Zuge der Abgrenzung des Systems und des Kontexts sollen noch keine Schnittstellen genau spezifiziert werden. [58]

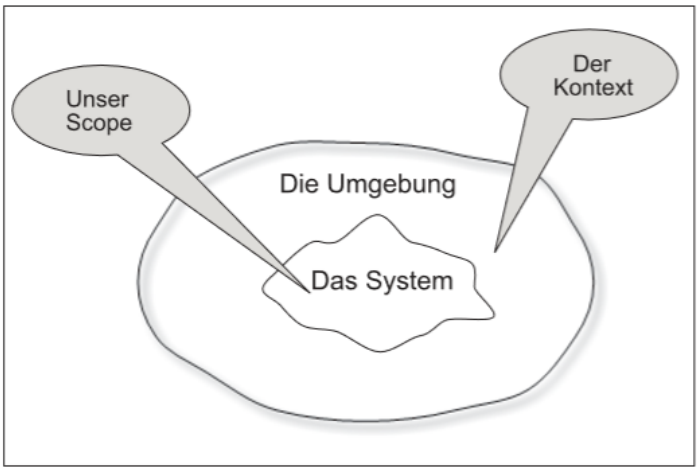


Abbildung 10: Das System und dessen Kontext, Quelle: Rupp/SOPHIST GROUP (2007), S.103.

3.3 Anforderungen ermitteln

Nachdem das Schaffen der erforderlichen Rahmenbedingungen im vorangegangenen Kapitel konkretisiert wird, kann auf die erste Tätigkeit des Requirements Engineerings, welche auch in Abbildung 7 zu sehen ist, näher eingegangen werden. In diesem Kapitel wird zuerst ein grober Überblick über die unterschiedlichen Techniken für die Ermittlung der Anforderungen gegeben. Danach wird auf die in dieser Arbeit verwendeten Ermittlungstechniken näher eingegangen. Anschließend wird eine Methode für die Beschreibung der ersten groben Anforderungen, die auch in dieser Masterarbeit verwendet wird, vorgestellt.

3.3.1 Ermittlungstechniken

Das Hauptziel aller Ermittlungstechniken ist die Unterstützung bei der Ermittlung von Anforderungen und Wissen der Stakeholder. Sie sollen dafür sorgen, dass alle relevanten Anforderungen frühzeitig erkannt und berücksichtigt werden können. Die Einsatzmöglichkeiten der unzähligen Ermittlungstechniken hängen

[58] Vgl. Rupp/SOPHIST GROUP (2007), S. 103 f.

von vielen Faktoren ab. Der erste der vier wichtigsten Faktoren ist die Unterscheidung nach der Art der Anforderung, welche im Unterkapitel 3.1.3 beschrieben wird. So führen bei unbewussten Anforderungen andere Ermittlungstechniken zum Ziel als bei bewussten. Ein weiterer wichtiger Faktor sind die Termin- und Budgetvorgaben, sowie die Verfügbarkeit der Stakeholder. An dritter Stelle ist die Erfahrung mit der entsprechenden Ermittlungstechnik zu nennen. Ist der Requirements Engineer mit einer Ermittlungstechnik vertraut, kann er sie effizienter einsetzen. Auch die Chancen und Risiken stellen einen wichtigen Einflussfaktor auf die Wahl der Ermittlungstechnik dar. Unter Chancen und Risiken werden unter anderem fachlich inhaltliche Einflüsse, wie die Komplexität des neuen Systems oder menschliche Einflüsse, wie die Kommunikationsfähigkeit zwischen Requirements Engineer und Stakeholder oder unter den Stakeholdern selbst, verstanden. [59]

Bei Befragungstechniken wie z.B. einem Interview oder einem Fragebogen liegt der Fokus auf der Erhebung von bewussten Anforderungen. Das Interview, wie auch der Fragebogen können offene oder geschlossene Fragen enthalten. Der größte Nachteil eines Fragebogens ist, dass nur abgefragt wird, was der Requirements Engineer bereits weiß oder vermutet. Im Gegensatz dazu können im Zuge eines Interviews neue Fragen entstehen und auch direkt besprochen werden. Wird das Interview gut geführt, können damit auch unterbewusste Anforderungen ermittelt werden. Der größte Nachteil des Interviews ist der große Zeitaufwand. Kreativitätstechniken dienen zur Ermittlung von innovativen Anforderungen und Begeisterungsfaktoren. In der Regel eignen sie sich nicht zum Ausarbeiten von detaillierten Anforderungen an das Systemverhalten. Bekannte Kreativitätstechniken sind z.B. Brainstorming oder Perspektivenwechsel. Beim Brainstorming werden in einer Gruppe von 5 bis 10 Personen möglichst viele Ideen in einer vorgegebenen Zeit gesammelt. Diese werden vorerst nicht bewertet oder kommentiert. Ist die vorgegebene Zeit um, werden die Ideen sortiert und es wird darüber diskutiert. Zum Schluss werden die besten Ideen dokumentiert. Besonders effektiv ist diese Methode, wenn die Gruppe interdisziplinär besetzt ist. Um die Anforderungen der dritten Dimension des Kano-Modells, die Basisanforderungen, zu ermitteln, werden am besten dokumentenzentrierte Techniken eingesetzt. Diese sind vergangenheitsorientierte Techniken. Im Falle einer Ablösung des Altsystems stellen diese Techniken sicher, dass alle Funktionen der alten Software in die neue übernommen werden. Systemarchäologie und die Wiederverwendung sind zwei beispielhafte Vertreter der dokumentenzentrierten Techniken. Beobachtungstechniken wie die Feldbeobachtung oder das Apprenticing eignen sich ebenfalls zur Ermittlung von Basisanforderungen. Im Gegensatz zur Feldbeobachtung, bei der der Requirements Engineer die Stakeholder beobachtet, versetzt er sich beim Apprenticing in die Lage des Stakeholders, indem er ihn bei seiner Aufgabenerledigung begleitet oder sie selbst durchführt. Wird die Methode der Feldbeobachtung oder des Apprenticing verwendet, erkennt der Requirements Engineer wahrscheinlich Funktionen oder Abläufe, welche von den Stakeholdern so nicht beschrieben werden würden, da sie für sie selbstverständlich sind. Dabei sollte der Requirements Engineer ständig die Schritte und Prozesse kritisch hinterfragen, um nicht umständliche Praktiken in die neue Software zu übernehmen. Da mit Erfüllung der Basisanforderungen die Kundenzufriedenheit nicht erhöht werden kann, soll diese Methode immer in Kombination mit einer anderen Methode eingesetzt werden. [60]

[59] Vgl. Pohl/Rupp (2015), S. 26.

[60] Vgl. Pohl/Rupp (2015), S. 27 – 32.

Zusammenfassend kann gesagt werden, dass es sinnvoll ist, eine Kombination mehrerer Ermittlungstechniken einzusetzen, um eine möglichst hohe Kundenzufriedenheit zu erreichen. Im Zuge dieser Masterarbeit werden Interviews geführt und die Methode des Brainstormings verwendet. Aus diesen Ermittlungstechniken ist die Anforderungsliste, welche im folgenden Unterkapitel beschrieben wird, entstanden.

3.3.2 Dokumentation der groben Anforderungen

Dieses Unterkapitel befasst sich mit der Dokumentation der im Brainstorming und in Interviews ermittelten Ideen. Eine übliche Dokumentationsart der groben Anforderungen, welche auch in dieser Arbeit verwendet wird, ist die textuelle Notation. Die Grundlagen zur textuellen Notation werden im folgenden Unterkapitel 3.3.2.1 beschrieben. Danach wird auf eine in dieser Arbeit verwendete Dokumentationsform, die Anforderungsliste, näher eingegangen. Neben der textuellen Notation können die groben Anforderungen auch mittels eines Use-Case-Diagramms dargestellt werden. Diese Notationsart wird im Unterkapitel 3.4.4.1 näher erläutert.

3.3.2.1 Grundlagen zur textuellen Darstellung

Ein wesentlicher Vorteil der textuellen gegenüber der im Unterkapitel 3.4 behandelten modellbasierten Darstellung ist, dass mit dieser funktionale und nicht-funktionale Anforderungen beschrieben werden können. Es gibt auch keine Beschränkung bezüglich des Einsatzgebietes. Der Grad der Präzision der textuellen Beschreibung reicht von informell bis halbformal. Textuelle, umgangssprachlich formulierte Formalismen sind in der Regel leicht verständlich und setzen kein Vorwissen voraus. Auch die textuelle Darstellung kann klassifiziert werden. Generell kann sie in strukturierte und unstrukturierte Darstellungen eingeteilt werden. So kann ein unstrukturierter Text durch Absätze, Überschriften oder ähnliches strukturiert werden. Strukturierte Texte lassen sich nach der Dimension der Struktur und der Form der Darstellung genauer klassifizieren. Listen werden als eindimensionale, strukturierte Darstellungen bezeichnet. Zu der eindimensionalen, strukturierten Darstellung zählen ebenfalls formalisierte Texte. Hier werden Textschablonen verwendet, um den Satzbau vorzugeben. Ähnlich sind Normsprachen und Pseudocode. Bei beiden Darstellungen handelt es sich um Text, bei dem bestimmte Schlüsselwörter mit frei wählbarem Text kombiniert werden. Um etwas kompliziertere Zusammenhänge einfach darstellen zu können, werden Matrizen oder Tabellen, welche auch bei der Anforderungsliste im Einsatz sind, verwendet. Diese zählen zu den zweidimensionalen, strukturierten Darstellungen. Eine weitere Form textueller Beschreibungen sind strukturierte Darstellungen mit einfachen, grafischen Elementen. Typische Vertreter dieser Kategorie sind Formulare. [61]

3.3.2.2 Anforderungsliste

Um die im Zuge des Brainstormings und der Interviews ermittelten Ideen zu dokumentieren, wird in dieser Arbeit die Anforderungsliste verwendet. Diese besteht aus einer Tabelle, bei welcher in den Zeilen die einzelnen Anforderungen und in den Spalten die unterschiedlichen Informationen zu den jeweiligen Anforderungen eingetragen werden. Für die Nachvollziehbarkeit im weiteren Entwicklungsprozess ist es zwingend, jeder Anforderung eine eindeutige ID zuzuweisen. Neben der eindeutigen ID sollen die Bezeichnung,

[61] Vgl. Partsch (2010), S. 71 – 78.

die Herkunft, die Priorität und der Status jeder Anforderung angeführt werden. Zu beachten ist, dass in der Anforderungsliste keine Anforderung gelöscht wird, sodass die Nachvollziehbarkeit gegeben bleibt. [62]

In Abbildung 11 ist eine Anforderungsliste illustriert. Diese dient als Beispiel zum leichteren Verständnis, die Einträge sind willkürlich gewählt. Zur Identifikation bekommt jede Anforderung eine eindeutige ID und eine kurze Beschreibung. Mit der dritten Spalte *eingestellt von* und der vierten Spalte *eingestellt am* ist die Herkunft der Anforderung bekannt. In den Spalten fünf und sechs sind die Priorität und der Status der jeweiligen Anforderung eingetragen. Die Anforderung mit der ID *3* wurde als *nicht mehr relevant* markiert. Das bedeutet, dass die Anforderung nicht weiter berücksichtigt wird. Aufgrund der Nachvollziehbarkeit wird sie, wie oben beschrieben, jedoch nicht gelöscht.

ID	Anforderung	eingestellt von	eingestellt am	Priorität	Status
1	Erstellung der Anlagendokumentation soll vollautomatisiert ablaufen (1-Klick)	D. Pichler	22.03.2018	hoch	in Konzeption
2	einfache Bedienung -> graphische Benutzeroberfläche (von Standarduser nach kurzer Einschulung bedienbar)	D. Pichler	22.03.2018	hoch	offen
3	Erstellung / Änderung der Templates soll einfach sein (von Key User nach kurzer Einschulung bedienbar)	D. Pichler	22.03.2018	mittel	nicht mehr relevant

Abbildung 11: Anforderungsliste – Beispiel, Quelle: Eigene Darstellung.

3.4 Detaillierung der Anforderungen / Modellbildung

Dieses Unterkapitel beschäftigt sich mit der Detaillierung der groben Anforderungen, welche im vorherigen Kapitel erhoben werden und der Dokumentation dieser Anforderungen. Somit wird dieses Kapitel der Tätigkeit *Dokumentieren* der Abbildung 7 zugeordnet. Im ersten Unterkapitel werden die generellen Methoden, welche für die Detaillierung angewandt werden können, beschrieben. Darauffolgend wird auf die Modellbildung mittels UML näher eingegangen. Dieses Thema beginnt mit den Grundlagen der Modellbildung und den Grundlagen von UML. Im Zuge der Beschreibung der Grundlagen von UML werden die Objektorientierung und die drei unterschiedlichen Sichtweisen von UML erläutert. Danach werden die Diagrammtypen, welche in dieser Masterarbeit verwendet werden, vorgestellt.

3.4.1 Methodik

Es gibt allgemeine, methodische Vorgehensweisen, um ein komplexes Problem zu lösen. Bekannte Methoden, um von einer groben Anforderung zu einer detaillierten Spezifikation zu kommen, werden durch Bezeichnungen wie *top-down*, *bottom-up*, *outside-in*, *inside-out* oder *hardest-first*, charakterisiert. Die wohl bekannteste Vorgehensweise, um zu einer Anforderungsbeschreibung zu kommen, ist die top-down Methode, welche in nachfolgender Abbildung 12 grafisch dargestellt wird. Dabei wird das große Ganze schrittweise in kleinere Einheiten zerlegt, bis die Ebene der atomaren Einheit erreicht wird. Mit der Zerlegung geht oft auch eine schrittweise Konkretisierung und Detaillierung einher. Somit werden bei jedem Zerlegungsschritt weitere, konkretere Einzelheiten hinzugefügt. Das Gegenteil zur top-down Methode stellt die

[62] Vgl. Krallmann/Dockter/Ritter (2017), S. 23 f.

bottom-up Methode dar. Hier werden kleine Komponenten sukzessive zu größeren Einheiten zusammengefügt. Besonders effektiv ist diese Methode, wenn geeignete Komponenten bereits zu Verfügung stehen. [63]

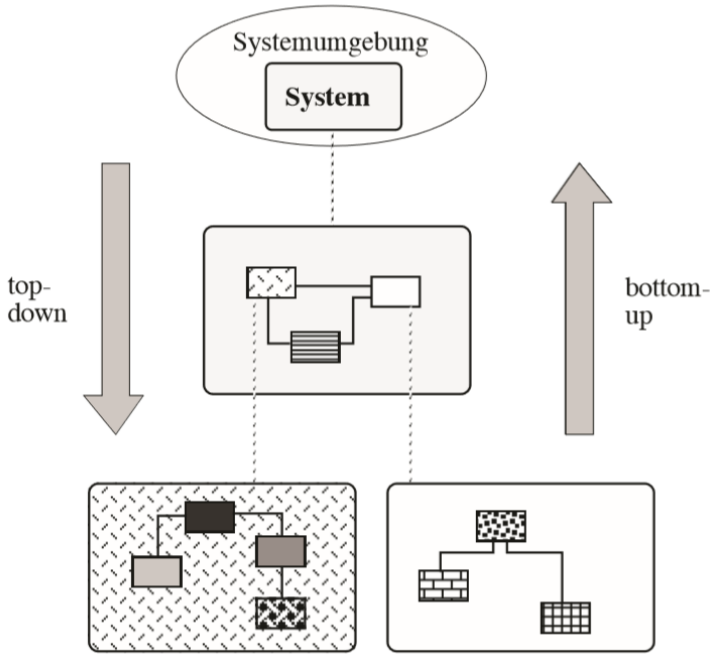


Abbildung 12: top-down und bottom-up Methode, Quelle: Partsch (2010), S. 58.

Unter *outside-in* wird eine Vorgehensweise zur Erstellung einer Anforderungsbeschreibung, bei der zuerst die Umgebung eines Systems modelliert wird, verstanden. Das kann durch Zusammenfassen mehrerer Sichtweisen und Informationen geschehen. Hierbei werden vor allem die Schnittstellen des Systems zu seiner Umgebung sichtbar. Ausgehend von den Schnittstellen, werden nun die Anforderungen des Systems sukzessive erarbeitet. Somit wird von außen (der Umgebung) zum Kern des Systems (innen) entwickelt. Die gegenteilige Methode ist das inside-out Verfahren. Hier wird als Erstes das Herzstück des Systems vollständig beschrieben und dieses dann nach unterschiedlichen Gesichtspunkten erweitert. Die Erweiterungen können zusätzliche Funktionen, Komponenten oder Daten betreffen. Auch Beschreibungen verschiedener Systemaspekte können dabei berücksichtigt werden. Als letzte Methode, die in diesem Abschnitt vorgestellt wird, ist die *hardest-first* Vorgehensweise zu nennen. Damit wird eine Methode bezeichnet, bei der zuerst die schwierigsten Aspekte bearbeitet werden. Es wird davon ausgegangen, dass der Rest leicht zu erledigen ist. Das Problem dabei ist, dass im Vorhinein oft nicht klar ist, was die schwierigen und was die einfachen Aspekte sind. Oftmals stellen sich im Zuge der Aufarbeitung auch anfangs als leicht gekennzeichnete Aspekte als schwierig dar. In der Praxis kommen üblicherweise Mischformen dieser Methoden zum Einsatz. Beispielsweise beeinflussen Informationen über das Ziel die Vorgehensweise und verhindern damit eine reine Form der top-down oder bottom-up Methode. Ähnliches gilt auch für die inside-out, outside-in und hardest-first Vorgehensweise. [64]

[63] Vgl. Partsch (2010), S. 58.

[64] Vgl. Partsch (2010), S. 59.

3.4.2 Grundlagen von Modellen

„Ein Modell ist ein abstrahiertes Abbild einer existierenden Realität oder Vorbild für eine zu schaffende Realität. Gegenstand der Abbildung können sowohl materielle wie auch immaterielle Objekte einer existierenden oder zu schaffenden Realität sein.“ [65] Diese Definition besagt, dass ein Modell immer ein Abbild der Realität darstellt, das Original bereits existiert oder erst entwickelt wird. Im Folgenden werden Beispiele aufgezählt, wo die Erstellung eines Modells sinnvoll ist. Oftmals werden Modelle erstellt, wenn die Größe des Originals schwer handhabbar ist. Ein weiterer Grund für die Erstellung eines Modells ist die Zugänglichkeit. Beispielsweise ist in nahezu jedem Arztzimmer ein DNA-Doppelhelix-Modell ausgestellt. Erstens ist das Original schwer zugänglich und zweitens für eine Betrachtung mit freiem Auge zu klein. Wird ein Objekt erst entwickelt, kann ein Modell für die frühzeitige grafische Darstellung verwendet werden. Auch können bereits Tests am Modell durchgeführt werden. Da Änderungen am Modell einfacher und somit kostengünstiger als am Original sind, sinken die gesamten Entwicklungskosten. Außerdem können mit Modellen Experimente in einem akzeptablen Zeitraum durchgeführt werden, welche sich sonst über sehr kurze oder lange Zeit, wie z.B. einige hundert Jahre, erstrecken. Im Folgenden wird näher beschrieben, was ein Modell auszeichnet und welche Vorteile die Modellbildung bietet.

Prinzipiell beschreiben drei Eigenschaften ein Modell, welche gleichzeitig auch die Vorteile von Modellen darstellen. Diese Definition eines Modells geht auf Stachowiak zurück: [66]

- **Abbild der Realität:** Jedes Modell bildet Aspekte der Realität ab. Die Tätigkeit der Modellbildung kann deskriptiv, wie auch präskriptiv sein. Bei der deskriptiven Modellbildung stellt das Modell Aspekte der Realität dar. Bei einer präskriptiven Modellbildung gibt das Modell die Realität, welche noch nicht existiert, vor. Abhängig vom Blickwinkel kann ein Modell gleichzeitig präskriptiv und auch deskriptiv sein. Beispielsweise können im Zuge des Requirements Engineerings die Forderungen der Stakeholder als deskriptiv und die Vorgaben für das noch nicht existente Modell als präskriptiv betrachtet werden.
- **Verkürzung der Realität:** Verkürzung der Realität bedeutet, dass immer nur ein gewisser Teil der Realität dargestellt wird. Hinsichtlich der Verkürzung wird zwischen Selektion und Verdichtung unterschieden. Bei der Selektion werden nur gewisse Aspekte des Originals abgebildet. Im Gegensatz dazu werden bei der Verdichtung die Aspekte zusammengefasst.
- **Pragmatische Eigenschaft:** Ein Modell wird immer für einen spezifischen Verwendungszweck in einem spezifischen Verwendungskontext konstruiert. Die betrachteten Aspekte in einem Modell (Verkürzung der Realität) werden vom Verwendungszweck abgeleitet. Im Idealfall enthält das Modell genau die Informationen, welche für den spezifischen Verwendungszweck notwendig sind.

Vor allem aufgrund der Verkürzung der Realität und der pragmatischen Eigenschaft werden Modelle so einfach wie möglich, aber so aufwendig wie nötig, gestaltet.

[65] Pohl/Rupp (2015), S. 63.

[66] Vgl. Stachowiak (1973), S. 131 ff.

3.4.3 Grundlagen UML

In diesem Unterkapitel werden die Grundlagen von UML beschrieben. Im nächsten Absatz wird auf die Entstehung und den Zweck von UML eingegangen. Im Anschluss daran werden die Methoden von UML, wie die Objektorientierung und die drei unterschiedlichen Sichtweisen näher erläutert.

Die Modellierungssprache Unified Modeling Language (UML) hat sich in den letzten Jahren als ein Quasi-Standard für die Modellierung von Anforderungen etabliert. Sie wurde Mitte der 90er Jahre entwickelt und entstand als Zusammenführung von drei unterschiedlichen Modellierungssprachen, der Booch-Methode, der OMT (Object Modeling Technique) und der OOSE (Object Oriented Software Engineering). [67] Mit der UML wurden zwei wesentliche Ziele verfolgt. Erstens sollten damit möglichst abstrakte, aber verständliche Beschreibungen von Softwaresystemen erstellt werden können. Damit sollten Programmabläufe und Designentscheidungen vor der Umsetzung in Form von Source Code diskutiert werden können. Das zweite Ziel der UML war es, eine gemeinsame Sprache für die Diskussion zwischen Fachbereich und IT-Abteilungen zu schaffen. Das einzige Manko für das Requirements Engineering ist die Notwendigkeit eines anderen Werkzeuges zur Modellierung von grafischen Benutzeroberflächen (GUI). [68]

3.4.3.1 Objektorientierung

UML ist eine objektorientierte Modellierungssprache. Die Prinzipien der Objektorientierung, die im Folgenden erläutert werden, sind jedoch nicht nur für UML, sondern auch für alle anderen objektorientierten Dokumentationsformen und objektorientierten Programmiersprachen gültig.

Der Durchbruch der objektorientierten Methoden gelang Anfang der 80er Jahre mittels der Programmiersprachen Smalltalk und C++. Seit Anfang der 90er Jahre werden objektorientierte Methoden auch im Bereich des Requirements Engineerings eingesetzt. Die Grundidee der objektorientierten Beschreibung besteht aus der Zusammenführung von Daten (Attribute) und Verhalten (Operationen) zu Objekten und Objektklassen. Dieses Prinzip, welches auch Kapselung genannt wird, wird in Abbildung 13 dargestellt. Insbesondere kann dadurch nicht direkt auf Daten zugegriffen werden, sie können nur durch Operationen gelesen oder verändert werden. Operationen können aber auch andere Aufgaben wie z.B. die Durchführung von Berechnungen oder das Erzeugen oder Löschen von Objekten haben. Ein Objekt kann ein Gegenstand, eine Person oder ein Begriff sein. Es hat eine eindeutige, nicht veränderbare Identität und charakteristische Eigenschaften, welche durch Attribute ausgedrückt werden. Jedes Attribut besitzt einen Namen und eventuell eine Typangabe, einen fixen Wert oder einen Initialwert. [69]

Eine Klasse ist eine Kollektion von Objekten mit denselben Eigenschaften (Attributen) und Funktionen (Operationen). Die Objekte einer Klasse werden Instanzen genannt. Klassen können hierarchisch organisiert sein. Somit werden die Eigenschaften und Funktionen von einer Superklasse auf die Subklasse der nächsten Ebene vererbt. Dadurch lassen sich Mehrfachspezifikationen und Inkonsistenzen vermeiden. Die Vererbung der Eigenschaften und Funktionen von einer Klasse zu einer Subklasse werden als *Vererbung*

[67] Vgl. Andresen (2003), S. 317.

[68] Vgl. Krallmann/Dockter/Ritter (2017), S. 57 f.

[69] Vgl. Partsch (2010), S. 159 – 162.

und das Zusammenfassen der Subklasse zu einer Superklasse als *Generalisierung* bezeichnet. Im Zuge der objektorientierten Programmierung gibt es noch die Möglichkeit, die vererbten Operationen zu überschreiben. Das ist im Zuge der objektorientierten Analyse nicht vorgesehen. Ein weiterer zentraler Aspekt der objektorientierten Betrachtungsweise ist die Unterscheidung zwischen Auftragserteilung und Auftragsdurchführung. Zuerst erhält das Objekt von einem anderen Objekt die Aufforderung, eine Aktion durchzuführen. Diese Aufforderung wird als Botschaft oder Nachricht bezeichnet. Danach führt das Objekt die Operation aus. [70]

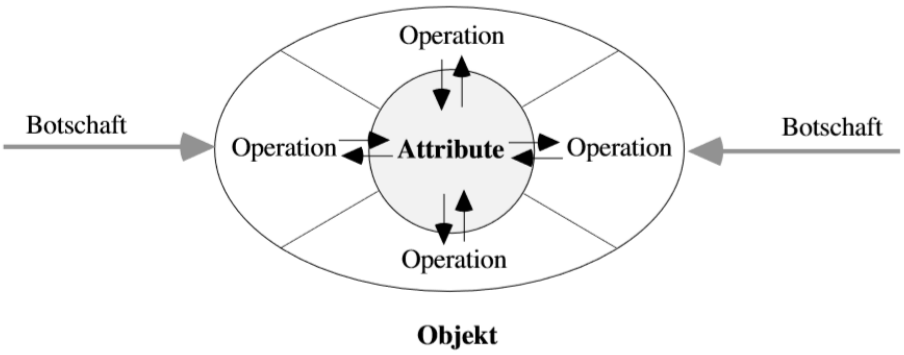


Abbildung 13: Kapselung, Quelle: Partsch (2010), S. 160.

3.4.3.2 Drei Perspektiven

Bei der Modellierung von Anforderungen mittels UML wird generell zwischen drei unterschiedlichen Perspektiven, der Strukturperspektive, der Funktionsperspektive und der Verhaltensperspektive, unterschieden.

Jede der drei Perspektiven Struktur, Funktion und Verhalten wird im Zuge der Anforderungsmodellierung getrennt voneinander unter Verwendung geeigneter konzeptueller Modellierungssprachen dokumentiert: [71]

- **Strukturperspektive:** In dieser Perspektive wird die statische Struktur des Systems, wie z.B. Ein- und Ausgabedaten dokumentiert. Des Weiteren werden die statisch-strukturellen Aspekte der Nutzungs- und Abhängigkeitsbeziehungen des Systems im Kontext behandelt. Zwei Beispiele für die Darstellung der statischen Struktur eines Systems sind das UML-Klassendiagramm und das Entity-Relationship-Diagramm.
- **Funktionsperspektive:** Die Funktionsperspektive beschreibt, welche Informationen aus dem Systemkontext durch die Funktionen des Systems manipuliert werden und welche Daten vom System an dessen Umgebung weitergeleitet werden. UML-Aktivitätsdiagramme, Datenflussdiagramme oder Ereignisgesteuerte Prozessketten sind Beispiele für die Dokumentation der Funktionsperspektive.
- **Verhaltensperspektive:** In der Verhaltensperspektive wird das System und dessen Einbettung in den Systemkontext zustandsorientiert dokumentiert. Das geschieht dadurch, dass z.B. die

[70] Vgl. Poetzsch-Heffter (2009), S. 2 ff.

[71] Vgl. Pohl/Rupp (2015), S. 75 – 91.

Reaktion des Systems auf ein bestimmtes Ereignis oder Bedingungen eines Zustandswechsels dargestellt werden. Für die Modellierung des dynamischen Verhaltens können beispielsweise Statecharts oder UML-Zustandsdiagramme verwendet werden.

Zwar werden die drei Perspektiven im Zuge der Modellierung voneinander unabhängig erstellt, jedoch können sich in einem Diagramm Aspekte einer anderen Perspektive wiederfinden. So können z.B. Daten, deren statische Struktur durch Klassendiagramme definiert ist, auch in den Aktivitätsdiagrammen abgebildet werden. Sie können dabei beispielsweise im Objektfluss die Ein- und Ausgaben für Aktionen darstellen. Aus diesem Grund sollen alle Diagramme regelmäßig auf Konsistenz und Vollständigkeit geprüft werden. [72]

Im Zuge der Masterarbeit wird ein Diagrammtyp pro Perspektive verwendet und vorgestellt. Näheres zu den verwendeten Diagrammtypen kann im Unterkapitel 3.4.4 nachgeschlagen werden.

3.4.4 Diagrammtypen

Die aktuelle Version UML 2.5 enthält 14 Diagrammtypen. [73] Im Zuge der Masterarbeit wird aber nur ein Diagrammtyp pro Perspektive verwendet und vorgestellt. Das UML-Klassendiagramm ist der Strukturperspektive, das UML-Aktivitätsdiagramm der Funktionsperspektive und das UML-Zustandsdiagramm der Verhaltensperspektive zuzuordnen. Zusätzlich werden in diesem Kapitel zwei weitere Diagramme beschrieben. Zum einen ist das das Use-Case-Diagramm, welches zur Modellierung der groben Anforderungen dient. Zum anderen wird auf das Präsentationsmodell, welches als einziges Diagramm nicht in UML enthalten ist, näher eingegangen.

3.4.4.1 Use-Case-Diagramm

Das Use-Case-Diagramm ist ein Diagrammtyp, welcher in der UML enthalten ist, und dient zur Darstellung der Hauptaufgaben des Systems und deren Interkation mit dem Systemkontext. Die Hauptaufgaben werden als Use-Cases bezeichnet. Somit beantwortet dieses Diagramm die Frage – *Wer soll was mit dem System tun können*. [74] Das Use-Case-Diagramm ist üblicherweise das erste Modell, welches im Zuge des Requirements Engineerings erstellt wird. Die Hauptaufgaben sind Prozesse, welche einen eindeutigen Anfang haben, ein klares Ergebnis liefern und logisch getrennte Aufgaben beschreiben. Wichtig ist, dass im Use-Case-Diagramm alle Aufgaben des Systems grundsätzlich dargestellt werden. [75]

Abbildung 14 zeigt die wichtigsten Elemente der Use-Case-Diagramme: [76]

- **Use-Case (1):** Die für das System definierten Use-Cases (Hauptaufgaben) werden durch Ovale dargestellt. Jeder Use-Case bekommt einen Namen, der im oder unterhalb des Ovals angegeben wird.

[72] Vgl. Pohl/Rupp (2015), S. 76.

[73] Vgl. Unhelkar (2018), S. 14.

[74] Vgl. Brandt-Pook/Kollmeier (2015), S. 58.

[75] Vgl. Kleuker (2018), S. 63.

[76] Vgl. Pohl/Rupp (2015), S. 70.

- **Akteur (2):** Akteure sind Personen oder Systeme, welche sich außerhalb des Systems befinden und mit diesem interagieren. Akteure können als Strichmännchen oder Rechtecke mit dem Stereotyp «actor» dargestellt werden. Handelt es sich bei dem Akteur um ein System, wird dem Strichmännchen der Stereotyp «system» hinzugefügt.
- **Systemgrenzen (3):** Die Systemgrenzen innerhalb eines Use-Case-Diagramms separieren die Systemteile (Use-Cases) von den Teilen des Systemkontextes (Akteure). Optional kann hier der Name des Systems angegeben werden.
- **Extend-Beziehung (4):** Eine Extend-Beziehung ist eine erweiterte, optionale Aufgabe eines Use-Cases. Bezugnehmend auf Abbildung 14 hat eine Extend-Beziehung von *Use-Case A* zu *Use-Case B* bestand. Das bedeutet, dass *Use-Case A* eine erweiterte Aufgabe von *Use-Case B* darstellt, welche aber nur bei Erfüllung einer bestimmten Bedingung ausgeführt wird. Der Einstiegspunkt wird als Extension Point bezeichnet und in diesem Beispiel, wie generell üblich bei *Use-Case B* dargestellt.
- **Include-Beziehung (5):** Eine Include-Beziehung sagt aus, dass die Interaktionsfolge eines Use-Cases in einem anderen enthalten ist. Im obigen Beispiel sind die Interaktionsfolgen von *Use-Case B* in denen von *Use-Case A* enthalten.
- **Beziehungen zwischen Akteuren und Use-Cases (6):** Findet während des Ausführens eines Use-Cases zwischen einem Use-Case und einem oder mehreren Akteuren eine Kommunikation statt, so wird das durch eine Linie dargestellt.

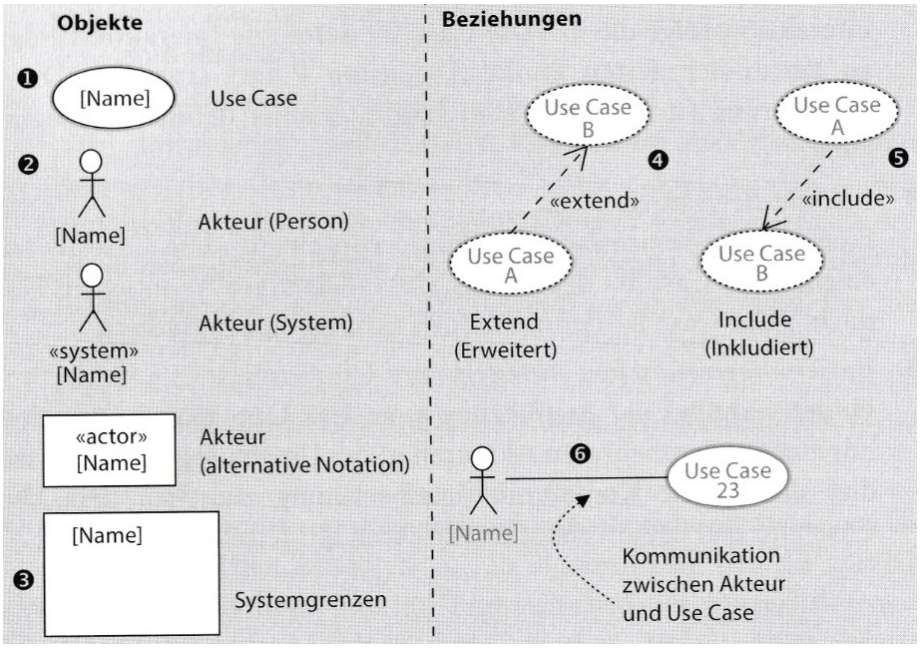


Abbildung 14: Modellelemente von Use-Case-Diagrammen, Quelle: Pohl/Rupp (2015), S. 69.

Des Weiteren sieht UML eine Generalisierungsbeziehung, welche im Unterkapitel 3.4.3.1 beschrieben wird, zwischen Use-Cases bzw. zwischen Akteuren vor. In diesem Fall werden die Eigenschaften des generalisierenden Use-Cases bzw. Akteurs auf die spezialisierten Use-Cases bzw. Akteure vererbt. Beispielsweise

werden alle Eigenschaften eines Akteurs *Mitarbeiter* auf einen spezialisierten Akteur *Werkstattmitarbeiter* vererbt. [77]

3.4.4.2 UML-Klassendiagramm

Klassendiagramme bilden nach wie vor die wichtigste und meist genutzte Modellierungstechnik der UML. Entstanden sind Klassendiagramme aus Anleihen von ER (Entity-Relationship)-Diagrammen und der grafischen Darstellung von Modulen, die ihrerseits von Datenflussdiagrammen beeinflusst wurden. Klassendiagramme beschreiben die statische Struktur eines Softwaresystems und bilden daher die erste behandelte Kernnotation der objektorientierten Modellierung. Ein Klassendiagramm besteht aus Klassen und Assoziationen zwischen diesen Klassen. Somit dokumentieren UML-Klassendiagramme das System aus der strukturellen Perspektive. Nahezu alle anderen Modellierungsdiagramme bauen auf Klassendiagramme auf. [78]

In Abbildung 15 sind die wichtigsten Modellelemente eines UML-Klassendiagrammes dargestellt. Eine Klasse beschreibt eine Kategorie von Objekten mit denselben Eigenschaften und Funktionen, die in der Anwendung eine Rolle spielen. Im Unterkapitel 3.4.3.1 werden Klassen und deren Eigenschaften genauer beschrieben. Die Klasse wird als Rechteck dargestellt, die den Namen der Klasse im oberen Teil und die Angaben zu den Attributen im unteren Teil enthalten. Optional können in einem dritten Teil Angaben zu den Methoden gemacht werden. Die Art der Daten, die ein Attribut enthalten kann, wird durch Attributtypen definiert. Beispielsweise können Attribute vom Typ string, integer oder ähnliches sein. Attributtypen werden getrennt durch einen Doppelpunkt nach dem Attributnamen angegeben. [79]

Assoziationen, der zweite wichtige Teil von Klassendiagrammen, werden durch eine verbindende Linie zwischen zwei Klassen dargestellt. Optional kann die Assoziation mit einem Namen versehen werden. Die Assoziationen und die Attribute einer Klasse beschreiben zusammen die Eigenschaften, die Objekte einer bestimmten Klasse haben können. Multiplizitäten sind Angaben über die Instanzebene eines Klassendiagramms. Diese geben an, wie viele Instanzen einer Klasse in Bezug auf die betrachteten Assoziationen mit wie vielen Instanzen der assoziierten Klasse in Beziehung stehen können. Die Multiplizität 0..1 gibt beispielsweise an, dass mindestens 0 und höchstens 1 Objekt miteinander in Beziehung stehen können. Im rechten oberen Eck der Abbildung 15 wird eine unvollständige Menge an Multiplizitäten dargestellt. Multiplizitäten können beliebe Mengen wie z.B. 3...* sein. Das Zeichen * steht für beliebig viele Objekte. Das heißt, dass drei oder mehr Objekte miteinander in Beziehung stehen können. Weiters kann einer Assoziation eine Rolle zugewiesen werden. Dadurch wird die Bedeutung der Instanzen einer Klasse in Bezug auf die betrachtete Assoziation genauer definiert. So hat die Klasse *Person* in der obigen Abbildung die Rolle *Besitzer* inne. [80]

[77] Vgl. Pohl/Rupp (2015), S. 71 f.

[78] Vgl. Rumpe (2011), S. 16.

[79] Vgl. van Randen/Bercker/Fieml (2016), S. 5 f.

[80] Vgl. van Randen/Bercker/Fieml (2016), S. 9 ff.

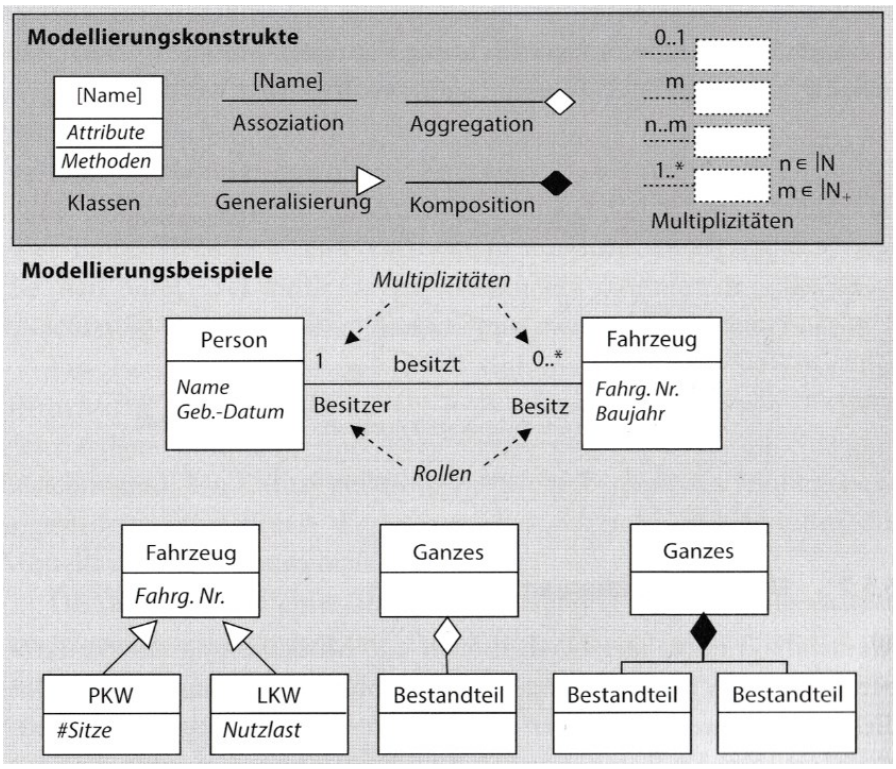


Abbildung 15: Modellelemente von UML-Klassendiagrammen, Quelle: Pohl/Rupp (2015), S. 80.

Komposition und Aggregation stellen eine Spezialform der Assoziationen dar. Beide beschreiben die Beziehung zwischen einem Ganzen und seinen Teilen. Die Komposition wird durch eine ausgefüllte Raute und die Aggregation durch eine nicht ausgefüllte Raute dargestellt. Die Komposition hat eine starke Bindung mit der Multiplizität 1 auf der Seite des Ganzen. Anders als bei der Komposition kann aufgrund der weniger starken Bindung bei Löschen des Ganzen einer Aggregation z.B. der Benutzer befragt werden, ob die Bestandteile auch gelöscht werden sollen. Die Bedeutung der Aggregation ist in UML nicht formal definiert. In Klassendiagrammen können darüber hinaus Generalisierungsbeziehungen zwischen Klassen dargestellt werden. Diese Beziehungen werden durch einen Pfeil dargestellt. Die Superklassen vererben bei einer Generalisierungsbeziehung die Attribute und Assoziationen an die Subklassen. So erben im Beispiel in Abbildung 15, die Klassen *PKW* und *LKW* die Attribute der Superklasse *Fahrzeug.* Die Subklassen können die Eigenschaften adaptieren bzw. erweitern. In obigem Beispiel wird der Klasse *LKW* das Attribut *Nutzlast* hinzugefügt. [81]

3.4.4.3 UML-Aktivitätsdiagramm

Um die Anforderungen bzw. das System aus der Funktionsperspektive modellieren zu können, ist das UML-Aktivitätsdiagramm ein weit verbreiteter Ansatz. Neben dem Aktivitätsdiagramm können Abläufe auch mittels der ereignisgesteuerten Prozessketten dargestellt werden. [82]

Aktivitätsdiagramme sind Kontrollflussgraphen, bestehend aus Aktivitätsknoten und dem Kontrollfluss zwischen den Aktivitätsknoten. Eine *Aktion*, welche auch in Abbildung 16 ersichtlich ist, repräsentiert eine

[81] Vgl. van Randen/Bercker/Fieml (2016), S. 11 – 17.

[82] Vgl. Pohl/Rupp (2015), S. 85.

Tätigkeit. Jede Aktion muss immer mit einem Namen gekennzeichnet sein. In den Aktivitätsdiagrammen sind zwei weitere Aktivitätsknoten, der Startknoten und der Endknoten, enthalten. Der Startknoten repräsentiert ein Ereignis, das die Ausführung des Aktivitätsdiagramms initiiert. Mit dem Endknoten wird das Aktivitätsdiagramm beendet. Mit dem Graphen *Objektfluss* können Ein- und Ausgabedaten modelliert werden. Zum Beispiel kann der Tätigkeit *Zielort erfragen* der Objektfluss *Zielort: [eingegeben]* nachgeschaltet werden. In diesem Fall werden die Daten des Zielortes weitergeleitet. *Kontrollflüsse*, welche die Objekte miteinander verbinden, werden immer als gerichtete Graphen dargestellt. Mit Aktivitätsdiagrammen können auch alternative Kontrollflüsse und nebenläufige Kontrollflüsse dargestellt werden. Den Start eines alternativen Kontrollflusses markieren *Entscheidungsknoten*. Nachfolgende Kontrollflüsse eines Entscheidungsknotens werden immer mit einer *Bedingung* versehen. Synchronisationsbalken, welche rechts oben in Abbildung 16 illustriert sind, gestatten die Darstellung von Nebenläufigkeiten. Die Hierarchisierung der Aktivitätsdiagramme erlaubt eine spezielle Form von *Aktionen*. So kann in einer Aktion eine andere Aktivität aufgerufen werden. Dieses Symbol wird im rechten, unteren Eck in Abbildung 16 dargestellt. [83]

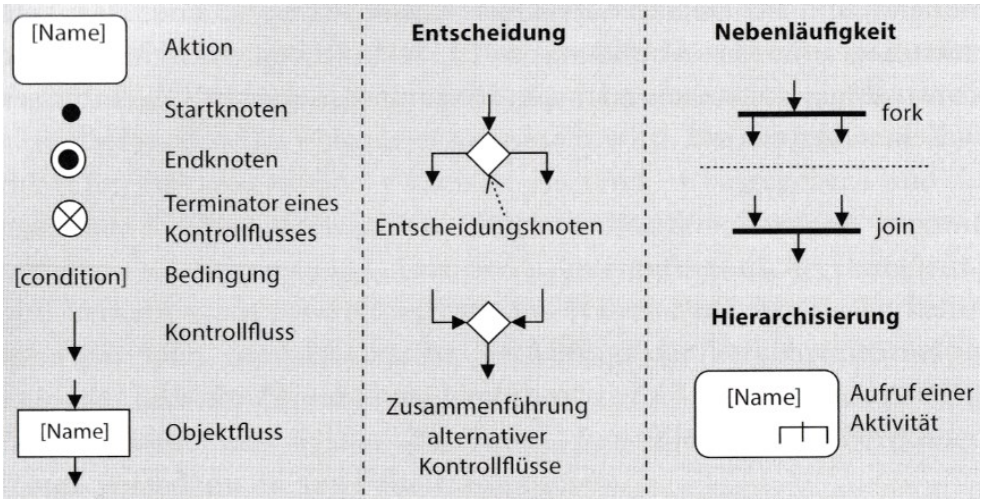


Abbildung 16: Modellelemente von UML-Aktivitätsdiagrammen, Quelle: Pohl/Rupp (2015), S. 85.

3.4.4.4 UML-Zustandsdiagramm

Um das dynamische Verhalten eines Systems zu modellieren, werden üblicherweise Modellierungsansätze verwendet, die auf der Theorie endlicher Automaten aufbauen. Die Praxis hat gezeigt, dass bei Verwendung der Theorie der endlichen Automaten Probleme auftreten, beispielsweise fehlt die Möglichkeit zur Abstraktion. Aufgrund dessen entstanden neue Methoden, wie z.B. die weit verbreiteten Statecharts. Diese bauen auf endlichen Automaten auf. Zusätzlich können mit Statecharts Hierarchisierungen, Bedingungen für Zustandsübergänge sowie Nebenläufigkeiten dargestellt werden. Auch UML bietet mit den UML-Zustandsdiagrammen eine Möglichkeit zur Modellierung des dynamischen Verhaltens. Die Notation der UML-Zustandsdiagramme wurde größtenteils von Statecharts übernommen. UML erweitert die Modellelemente von Statecharts z.B. um die Möglichkeit, explizite Ein- und Austrittspunkte bei hierarchischen Zuständen zu definieren. In nachfolgender Abbildung sind die wichtigsten Modellelemente der UML-Zustandsdiagramme dargestellt. Jeder *Zustand* wird mit einem abgerundeten Rechteck dargestellt und durch einen

[83] Vgl. Pohl/Rupp (2015), S. 85 f.

Namen identifiziert. Ein Zustand definiert einen Zeitraum, in dem das System ein bestimmtes Verhalten zeigt und auf das Eintreten eines definierten Ereignisses wartet. Ein *Zustandsübergang* wird durch das Eintreten eines bestimmten Ereignisses ausgelöst und verbindet einen bestimmten Zustand mit einem Folgezustand. In der linken unteren Ecke der Abbildung 17 ist zu sehen, wie Hierarchien dargestellt werden. Die Darstellung der *Ein-* und *Austrittspunkte*, welche bei UML 2 als Erweiterung von Statecharts entwickelt wurde, ist ebenfalls in folgender Abbildung ersichtlich. Ein Eintrittspunkt ist ein extern sichtbarer Pseudozustand, der unmittelbar mit einem internen Zustand verknüpft ist. Dagegen besitzt ein Austrittspunkt, welcher ebenfalls ein extern sichtbarer Pseudozustand ist, als Ursprung einen internen Zustand. Jedes Zustandsdiagramm benötigt einen *Startzustand*. Das UML-Zustandsdiagramm kann auch einen *finalen Zustand* beinhalten. *Nebenläufige Zustandsmaschinen* werden über ein abgerundetes Rechteck mit einer strichlierten Trennlinie dargestellt. [84]

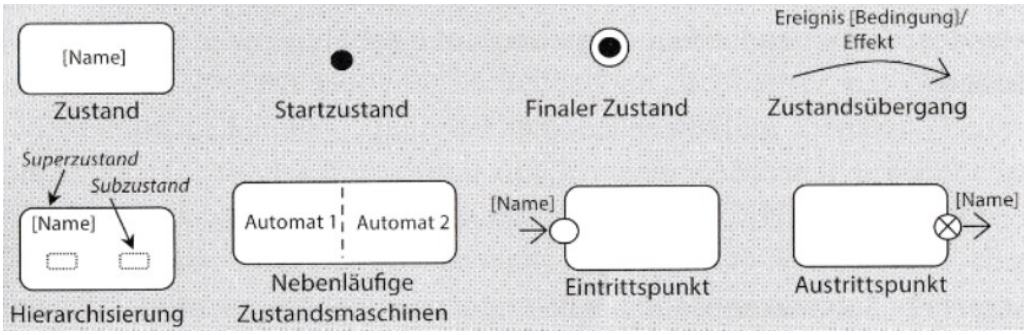


Abbildung 17: Modellelemente eines UML-Zustandsdiagrammes, Quelle: Pohl/Rupp (2015), S. 91.

3.4.4.5 Präsentationsmodell

Unter einem Präsentationsmodell ist ein Modell zu verstehen, dass einen ersten Eindruck der grafischen Benutzeroberfläche, auch GUI (Graphical User Interface) genannt, vermittelt. Wie bereits in der Einleitung des Unterkapitels 3.4.3 beschrieben, bietet UML keine Möglichkeiten für die Modellierung von grafischen Benutzeroberflächen. Da es im Zuge des Requirements Engineerings trotzdem sinnvoll ist, den Stakeholdern ein Modell der Benutzeroberfläche vorzulegen, wird auf eine Software namens *Pencil* zurückgegriffen. Im folgenden Absatz wird auf die Wichtigkeit einer anwenderfreundlichen GUI eingegangen. Des Weiteren werden Prinzipien beschrieben, an die sich eine qualitativ hochwertige Benutzeroberfläche orientiert. Im letzten Absatz wird die Software *Pencil* und deren Modellelemente vorgestellt.

Die grafische Benutzeroberfläche ist für den Erfolg einer Software essenziell. Es ist der einzige Teil der Software, der für den User sichtbar ist. So kann durch eine anwenderfreundliche Benutzeroberfläche, bei derselben Funktion der Software, die Akzeptanz und die Zufriedenheit der User bzw. der Kunden gesteigert werden. Jedoch wird mittels der grafischen Benutzeroberfläche nicht nur das subjektive Empfinden der Benutzer beeinflusst, sondern können hohe Kosten entstehen, wenn ein Design unübersichtlich und ineffizient ist. So konnte in verschiedenen Versuchen die Produktivität der Benutzer nur durch Änderung der Benutzeroberfläche um bis zu 25 % oder sogar 40 % gesteigert werden. Auch die Fehlerrate sank signifikant. Zusätzlich fallen Schulungskosten niedriger aus und User benötigen weniger oft Hilfe des Support-

[84] Vgl. Pohl/Rupp (2015), S. 89 – 92.

Teams. Sind User frustriert, sinkt auch deren generelle Motivation. Aus diesen Gründen wurden Prinzipien entwickelt, nach denen grafische Benutzeroberflächen aufgebaut sein sollen. Diese Prinzipien wurden aus verschiedenen Büchern, herausgegeben von Microsoft, IBM, Open Software Foundation, Galitz usw., abgeleitet. Insgesamt handelt es sich um 26 Prinzipien, die zusammengefasst aussagen, dass eine Software unter anderem einfach, effizient und intuitiv bedienbar sein soll. Um das zu erreichen, dürfen nur so viele Informationen wie nötig am Bildschirm ersichtlich sein. Außerdem soll eine GUI ein ansprechendes Design besitzen, sodass die Benutzer bei Verwendung der Software von einer positiven Erfahrung berichten. Des Weiteren sollen möglichst viele Fehleingaben des Benutzers durch die Software abgefangen werden. Keine Kombination von Fehleingaben soll zum Absturz des Systems führen. Darüber hinaus muss die Software dem Benutzer unmissverständliche und schnelle Rückmeldungen und Statusinformationen zur Verfügung stellen. [85]

Pencil ist eine Open-Source Prototyping Software, welche kostenlos verwendet werden kann. [86] Mit dieser Software können Mockups im Windows, Android oder iOS Design erstellt werden. Ein Mockup kann mit *augenscheinliches Modell* übersetzt werden. Das heißt, dass das Modell so aussehen, aber nicht so funktionieren muss wie das Original. Neben den genannten Designs können Mockups mit *Pencil* auch in einem *Desktop – Sketchy GUI* Design erstellt werden. In Abbildung 18 ist zu sehen, dass dieses Design einer Handskizze ähnelt. Das hat den Vorteil, dass bei Vorlage dieses Modells, welche üblicherweise in der frühen Phase des Requirements Engineerings stattfindet, nicht über die Graphik spezifischer Buttons, sondern das generelle Aussehen der Bedienoberfläche diskutiert wird. Außerdem schränkt sich das Design damit nicht auf ein Betriebssystem ein.

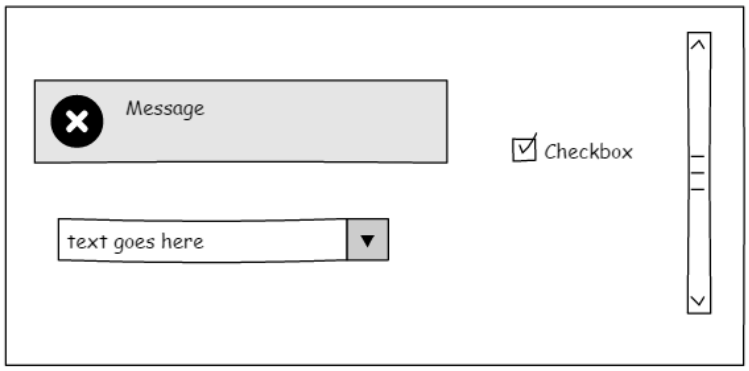


Abbildung 18: Pencil Desktop - Sketchy GUI, Quelle: Eigene Darstellung.

3.5 Anforderungen prüfen

Die Prüfung der Anforderungen soll sicherstellen, dass sie den Qualitätskriterien, welche im Unterkapitel 3.1.4 beschrieben werden, entsprechen. Somit ist dieses Kapitel der Tätigkeit *Prüfen* der Abbildung 7 zuzuordnen. In diesem Unterkapitel werden zuerst zwei mögliche Methoden zur Qualitätssicherung beschrieben. Danach wird auf die Klassifizierung der Qualitätskriterien und auf ein paar Prüftechniken eingegangen. Am Schluss dieses Unterkapitels wird die Theorie hinter der Auswahl der prüfenden Personen beschrieben.

[85] Vgl. Galitz (2007), S. 3 – 58.

[86] Vgl. Evolus (2017), Online-Quelle [27.November.2018].

Der einfachste und kostengünstigste Weg, um qualitätsvolle Ziele und Anforderungen und eine hochwertige Stakeholderanalyse zu erhalten, ist die konstruktive Qualitätssicherung. Bei der konstruktiven Qualitätssicherung werden vorbeugende Maßnahmen getroffen, die bereits im Zuge der Erstellung der Anforderungen, Ziele, etc. eine hohe Qualität garantieren. So wird im Unterkapitel 3.2 erläutert, worauf bei der Auswahl der Stakeholder und der Abgrenzung des Systems zu achten ist. Des Weiteren wird in diesem Unterkapitel auch beschrieben, welchen Anforderungen Ziele entsprechen müssen. Wurden diese Kriterien erfüllt und die Anforderungen nach den im Unterkapitel 3.1.4 geforderten Qualitätskriterien ermittelt und dokumentiert, ist von Vornherein eine gute Qualität der Anforderungen sichergestellt. Im Gegensatz zur konstruktiven Qualitätssicherung werden bei der analytischen Qualitätssicherung die Anforderungen nach der Dokumentationsphase und vor Verwendung gegen die beschriebenen Qualitätskriterien geprüft. [87]

Um die große Anzahl der Qualitätskriterien, welche im Kapitel 3.1.4 vorgestellt werden, übersichtlicher darzustellen, können sie in drei Hauptkategorien eingeteilt werden: [88]

- **Syntaktisch:** Syntaktische Qualitätskriterien sind Kriterien, die beschreiben, dass Anforderungen gut formuliert sind und repräsentiert werden. Zu syntaktischen Kriterien zählen Verfolgbarkeit, Redundanzfreiheit, Eindeutigkeit und Notwendigkeit.
- **Semantisch:** Das sind Kriterien, welche sich auf inhaltliche und fachliche Aspekte wie Korrektheit, Gültigkeit und Vollständigkeit beziehen.
- **Weiterverwendung:** Darunter werden alle Aspekte verstanden, welche die Weiterverwendbarkeit von Anforderungen garantieren. Dazu zählen die Realisierbarkeit, Bewertbarkeit und Verständlichkeit.

Zu den am häufigsten verwendeten manuellen Prüftechniken zählen Reviews. Diese können aufgrund des Grades der Formalität, der Vorbereitung und der Durchführung in Stellungnahme, Walkthrough und Inspektion eingeteilt werden. So gehört die Stellungnahme zu den wenig formalen, aber dadurch schnell durchzuführenden Prüftechniken. Der Autor gibt dabei das zu prüfende Dokument z.B. einem fachlich versierten Kollegen, welcher die Auffälligkeiten in diesem Dokument markiert. Eine etwas formalere Prüftechnik ist der Walkthrough. Bei dieser Methode leitet der Autor die Prüfer schrittweise durch den Prüfgegenstand. Die Inspektion kann für Anforderungen, Code, Testfälle usw. verwendet werden und läuft in mehreren aufeinanderfolgen Phasen ab. Somit zählt sie zu den formalsten und damit aufwendigsten manuellen Prüfverfahren. Daneben gibt es noch weitere Prüfverfahren wie z.B. Prototyping/Simulationsmodell oder Agile Specification Quality Control. Im ersten Beispiel wird ein Prototyp oder ein Simulationsmodell erstellt, anschließend wird der Prototyp oder das Simulationsmodell gegen die Anforderungen validiert. Dabei werden auch Unvollständigkeiten und Missverständnisse in den Anforderungen erkannt. Die agile Prüfmethode ist eine neue Variante der Inspektion und zeichnet sich durch einen früheren Startzeitpunkt, einer häufigeren Frequenz und kleineren Teams aus. Die Agile Specification Quality Control verwendet unter anderem Ansätze der Stichprobentechnik und der Prozessverbesserung. [89]

[87] Vgl. Rupp/SOPHIST GROUP (2007), S. 306.

[88] Vgl. Rupp/SOPHIST GROUP (2007), S. 306.

[89] Vgl. Rupp/SOPHIST GROUP (2007), S. 311 – 315.

Neben der Methode der Prüfung selbst, ist auch die Wahl der Prüfer von großer Bedeutung. Analytiker z.B. prüfen sehr effizient, finden aber weniger semantische Fehler, da sie darauf geschult sind, Zusammenhänge zu erkennen. Für die inhaltliche Prüfung sind wiederum Experten aus dem Fachbereich gefragt, da diese den Inhalt verstehen. Ungeübte Stakeholder brauchen in der Regel mehr Zeit für eine Prüfung, als geübte. Abbildung 19 stellt eine Übersicht der Prüfkompetenzen der unterschiedlichen Stakeholder dar. Aus diesem Bild ist ersichtlich, dass es Sinn macht, mehrere Stakeholder bei der Prüfung zu berücksichtigen. So ist hier z.B. zu sehen, dass Entwickler für die Prüfung der Realisierbarkeit sehr gut geeignet sind. Wird dagegen auf Verständlichkeit oder Vollständigkeit geprüft, sollte auf andere Stakeholder, wie z.B. den Anwendern zurückgegriffen werden. Generell können nur die Stakeholder selbst prüfen, ob die Anforderungen ihren Vorstellungen entsprechen. Des Weiteren spielt die Anzahl der Prüfer eine wichtige Rolle. Dabei sind zwei Begriffe von Bedeutung, die Effizienz und die Effektivität. Steigt die Anzahl der Prüfer, steigt auch die Effektivität. Die Effektivität beschreibt die Anzahl der Auffälligkeiten, die die Prüfer insgesamt finden können. Dieser Anstieg ist jedoch nicht linear, da unterschiedliche Prüfer auch immer eine gewisse Menge an identischen Auffälligkeiten finden. Der zweite Begriff, die Effizienz, stellt die gefundenen Auffälligkeiten in Relation zum Aufwand. Für die Inspektion existieren Studien, die besagen, dass die maximale Effizienz mit Teamgrößen von zwei bis drei Prüfern und die maximale Effektivität mit vier bis fünf Prüfern erreicht wird. [90]

-- Gar nicht geeignet - Nicht gut geeignet o Kein Einfluss -> geeignet + Gut geeignet ++ Sehr gut geeignet	Fachliche Richtigkeit	rechtliche Verbindlichkeit	Notwendigkeit / Auftrag des Projektes /	Vollständigkeit	Verständlichkeit, gute Struktur und Konsistenz	Eindeutigkeit, Widerspruchsfreiheit	Traceability / Nachvollziehbarkeit	Realisierbarkeit	Testbarkeit	Kritikalität
Anwender	+	+	++	++	0	0	0	0	0	
Fachlich Verantwortlicher	++	+	0	0	0	0	0	0	++	
Analytiker	0	0	++	+	++	++	+	+	+	
Projektleiter	+	++	0	+	0	0	0	0	+	
Entwickler	0	0	0	0	0	0	++	0	0	
Tester	0	0	0	+	+	0	0	++	0	
Projektexterne Auditoren	0	0	++	+	+	+	0	+	0	

Abbildung 19: Prüfende Personen, Quelle: Rupp/SOPHIST GROUP (2007), S. 309.

3.6 Anforderungen verwalten

Im Zuge dieses Unterkapitels werden die zu beachtenden Aspekte bei der Verwaltung der Anforderungen beschrieben. Im folgenden Absatz wird auf die Verfolgbarkeit und die Abhängigkeiten zwischen den Anforderungen näher eingegangen. Das Änderungsmanagement der Anforderungen, speziell die Versionierung und Verfeinerung wird im darauffolgenden Absatz beschrieben.

[90] Vgl. Rupp/SOPHIST GROUP (2007), S. 308 f.

Um die Herkunft einer Anforderung oder einer Entscheidung nachvollziehen zu können, muss die Verfolgbarkeit, auch unter dem Begriff *traceability* bekannt, gegeben sein. Außerdem müssen im Zuge der Verfolgbarkeit die Abhängigkeiten aller Informationen und Repräsentationsformen bekannt sein. Eine Anforderung kann als verfolgbar bezeichnet werden, wenn deren Herkunft, die Entwicklung über das gesamte Projekt und die Zusammenhänge zu nachgelagerten Entwicklungsartefakten bekannt sind. In der Literatur kommen die zwei Begriffe *pre-* und *post-traceability* häufig vor. Erstere beschreibt Zusammenhänge zu vorgelagerten, zweitere zu nachgelagerten Anforderungsebenen oder Entwicklungsartefakten. Eine weitere mögliche Einteilung ist die zwischen der *horizontalen* und *vertikalen* Verfolgbarkeit. Zusammenhänge auf derselben Entwicklungsstufe werden als horizontal, Zusammenhänge auf unterschiedlichen Entwicklungsstufen als vertikal bezeichnet. Sind die Zusammenhänge bekannt, ist das vor allem bei Änderungen von großem Nutzen. Mit diesem Wissen ist beim Aufkommen einer Änderungsanfrage sofort bekannt, welche Entwicklungsartefakte diese Änderung betrifft und somit kann der entstehende Aufwand schnell abgeschätzt werden. Jedoch wächst die Anzahl der Zusammenhänge exponentiell mit der Anzahl der Anforderungen, wodurch schnell eine sehr große Anzahl von Abhängigkeiten entstehen kann. Eine große Herausforderung stellt somit die Entscheidung dar, welche Zusammenhänge berücksichtigt und welche im Zuge der Verfolgbarkeit nicht beachtet werden. [91]

Ein weiterer wichtiger Punkt in Bezug auf das Verwalten der Anforderungen sind die Versionierung und die Verfeinerung. Wird eine Anforderung geändert und somit versioniert, ersetzt die neue Version die alte und die alte Version verliert ihre Gültigkeit. Aufgrund der Nachvollziehbarkeit darf die alte Version jedoch nicht einfach gelöscht werden, sondern muss erhalten bleiben und dementsprechend markiert werden. Um die Nachvollziehbarkeit gewährleisten zu können, müssen auch den Versionen verschiedene zusätzliche Informationen hinzugefügt werden. Bei jeder Version sind drei Informationen essenziell. Die erste ist der Ersteller der Version, die zweite das Erstellungsdatum und die dritte der Grund der neuen Version. Da das Requirements Engineering ein iterativer Prozess ist, wurde die Methode der Verfeinerung eingeführt. Am Beginn ist eine Anforderung oftmals nur grob definiert und wird mit der Zeit verfeinert. Um auch hier wieder die Nachvollziehbarkeit garantieren zu können, werden Eltern-Kind-Beziehungen eingeführt. Dadurch entsteht eine Baumstruktur, bei der die Herkunft der detaillierten Spezifikationen einfach abgelesen werden kann. Der wichtigste Unterschied zur Versionierung ist, dass die alte grobe Spezifikation nicht durch die neue feinere ersetzt wird, sondern beide ihre Gültigkeit behalten. [92]

[91] Vgl. Partsch (2010), S. 64 f.

[92] Vgl. Rupp/SOPHIST GROUP (2007), S. 415 f.

4 TEST ENGINEERING

Bei der Herstellung eines Industrieproduktes werden die entstehenden Teilprodukte und das Endprodukt daraufhin kontrolliert, ob sie den gestellten Anforderungen entsprechen. Je nach Produkt gibt es auch unterschiedliche Anforderungen an die Qualität. Dieselben Aussagen gelten auch für ein Softwareprodukt. Durch das Testen verringern sich die Risiken beim Einsatz der Software, da Fehler im Zuge der Testung, also vor Auslieferung der Software, erkannt werden. Als Fehler oder Mangel ist eine Abweichung zwischen dem Soll- und dem Ist-Zustand definiert. Ein Mangel kann beispielsweise auch eine Beeinträchtigung der Verwendbarkeit bei gleichzeitiger Erfüllung der Funktionalität sein. [93]

Im Verlauf eines typischen Entwicklungsprozesses nach dem V-Modell werden die Anforderungen vom Groben ins Feine detailliert. Wie in Abbildung 20 illustriert, verlaufen die Tests genau umgekehrt. Der erste Test im Entwicklungsprozess ist der Modultest, welcher auch Komponententest genannt wird, der letzte der Abnahmetest. Im Zuge der Entwicklung werden die einzelnen Programmteile, welche beim Modultest geprüft werden, sukzessive zu größeren Einheiten zusammengefügt, bis beim System- und Abnahmetest das gesamte System gegenüber den Anforderungen geprüft wird. Die Grundidee hinter dem V-Modell sagt auch, dass jedem Dekompositionsschritt ein Testschritt zugeordnet werden muss. [94]

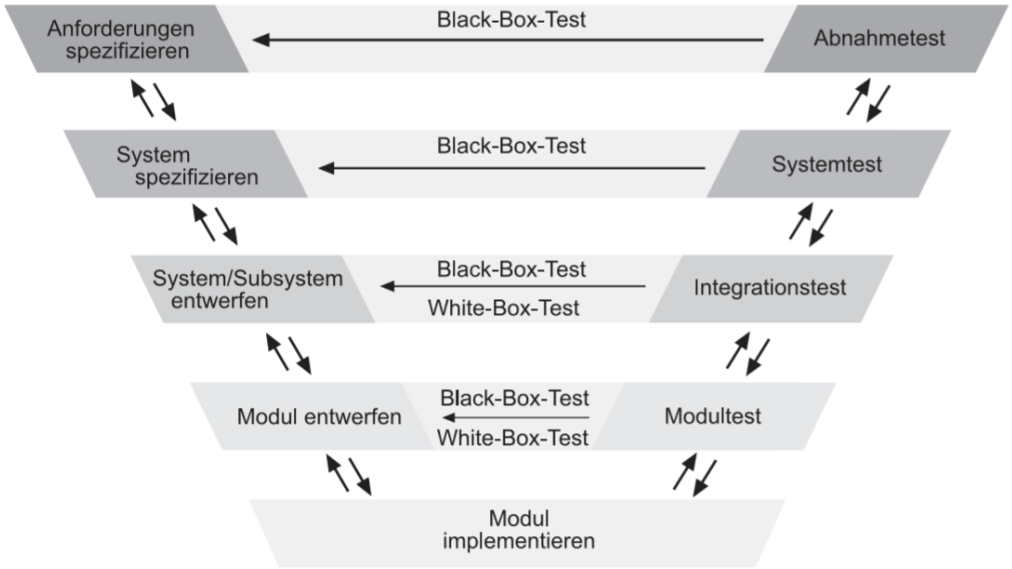


Abbildung 20: Testaktivitäten gemäß V-Modell, Quelle: Rupp/SOPHIST GROUP (2007), S. 333.

In obiger Abbildung wird zwischen Black-Box- und White-Box-Test unterschieden. Unter einem Black-Box-Test wird ein Test ohne Berücksichtigung der inneren Struktur bezeichnet. Es werden nur die Ein- und Ausgaben des Systems betrachtet. Im Gegensatz dazu wird bei White-Box-Tests, welche auch als Glass-Box-Tests bekannt sind, die innere Struktur eines Systems, wie z.B. dessen Source Code betrachtet und bewertet. Die Wahl der Testmethode ist unter anderem von der Detaillierungsebene abhängig. So kann es bei einzelnen Modulen sinnvoll sein den Source Code zu bewerten. Bei der Testung des gesamten Systems

[93] Vgl. Spillner/Linz (2012), S. 6.

[94] Vgl. Rupp/SOPHIST GROUP (2007), S. 325.

werden aufgrund des Umfangs ausschließlich Black-Box-Tests zum Einsatz kommen. Ebenso werden Funktionen in UML in verschiedenen Detaillierungsgraden dargestellt. So ist die Darstellung der Use-Cases sehr grob und wird den oberen Bereichen des V-Modells zugeordnet. Klassendiagramme oder Aktivitätsdiagramme stellen dagegen die Basis für Modul- und Integrationstests dar. [95]

In den folgenden Unterkapiteln werden die vier Tests Komponenten-, Integrations-, System- und Abnahmetest, welche auch in Abbildung 20 ersichtlich sind, erläutert.

4.1 Komponententest

Beim Komponenten- oder Modultest werden die kleinsten Softwareeinheiten eines Systems unmittelbar nach der Erstellung erstmalig einem systematischen Test unterzogen. Abhängig von der Programmiersprache werden diese Einheiten als Module, Units oder Klassen bezeichnet. Kennzeichnend für den Komponententest ist, dass nur jeweils ein isolierter Softwarebaustein getestet wird. Somit können komponentenexterne Einflüsse ausgeschlossen und Fehler oder Mängel lokalisiert werden. Alternativ dazu kann die zu testende Einheit auch aus mehreren zusammengesetzten Bausteinen bestehen. Entscheidend für den Komponententest ist, dass komponenteninterne Aspekte und nicht deren Wechselwirkung mit benachbarten Komponenten geprüft werden. [96]

Im Zuge des Komponententests wird das Hauptaugenmerk auf die vier Aspekte Funktionalität, Robustheit, Effizienz und Wartbarkeit, gelegt. Die Funktionalität beschreibt das Ein-/Ausgabeverhalten einer Komponente. Hier stellt sich die Frage, ob die Komponente auf bestimmte Eingaben so reagiert, wie es in den komponentenspezifischen Anforderungen definiert ist. Zusätzlich soll die Reaktion der Komponente auf falsche oder nicht vorgesehene Eingaben getestet werden. Dies geschieht im Zuge des Robustheitstests. Im Prinzip funktioniert er gleich wie der Funktionalitätstest, nur dass als Eingabe Methodenaufrufe, Daten und Sonderfälle verwendet werden, die laut Spezifikation nicht zulässig oder nicht vorgesehen sind. Diese zwei Tests können als Black-Box-Tests bezeichnet werden. Die Effizienz gibt an, wie wirtschaftlich eine Komponente mit den verfügbaren Rechnerressourcen umgeht. Betrachtete Werte sind z.B. der benötigte Speicherplatz oder die benötigte Rechenzeit. Vor allem für eingebettete Systeme, bei denen hardwaremäßig limitierte Ressourcen zur Verfügung stehen, spielt dieser Test eine Rolle. Im Zuge der Testung der Wartbarkeit wird geprüft, wie leicht oder schwer es fällt, das Programm zu ändern oder weiterzuentwickeln. Wesentlichen Einfluss auf diese Größen haben die Struktur des Codes, die Ausführlichkeit der Kommentare, die Verständlichkeit und Aktualität der Dokumentation. Da bei diesem Test die innere Struktur einer Komponente betrachtet wird, wird er als White-Box-Test bezeichnet. [97]

In der Literatur wird manchmal zwischen Modul- und Komponententest unterschieden. Hierbei ist der Modultest mit der oben beschriebenen Prüfung äquivalent, beim Komponententest werden zudem die

[95] Vgl. Baker/u.a. (2008), S. 11 f.

[96] Vgl. Spillner/Linz (2012), S. 44 f.

[97] Vgl. Spillner/Linz (2012), S. 48 ff.

Interaktionen der einzelnen Klassen geprüft. In dieser Sichtweise wird der Komponententest als eine weitere Teststufe zwischen dem Modul- und Integrationstest gesehen. [98]

Als Testbasis für den Komponententest können die komponentenspezifischen Anforderungen und das Softwaredesign der Komponenten herangezogen werden, d.h. dass vor allem Klassendiagramme eine gute Basis darstellen. Zusätzlich können Zustandsdiagramme oder Aktivitätsdiagramme mit einem hohen Detaillierungsgrad für die Prüfung der Interaktion der Klassen verwendet werden. [99]

4.2 Integrationstest

Die zweite Teststufe im V-Modell, nach dem Komponententest, ist der Integrationstest. Er setzt voraus, dass die ihm übergebenen Testobjekte getestet sind und die Fehler korrigiert wurden. Als Vorbereitung zum Integrationstest werden einzelne Bausteine zu größeren Baugruppen zusammengefasst. Dieser Vorgang wird als Integration bezeichnet. Der Integrationstest stellt sicher, dass die Interaktionen zwischen den einzelnen Bausteinen reibungslos ablaufen. Er hat also das Ziel, Fehlerzustände in Schnittstellen und im Zusammenspiel zwischen den integrierten Komponenten aufzudecken. Neben den internen Schnittstellen des Systems werden im Zuge des Integrationstestes auch Schnittstellen mit der Systemumgebung betrachtet. Dies wird auch als Integrationstest im Großen bezeichnet. [100]

Im Zuge der Integration werden die Einzelbausteine schrittweise zu größeren Einheiten zusammengesetzt. Im Idealfall soll jedem Iterationsschritt ein Test zugeordnet sein. Die getesteten Einheiten können wiederum die Basis für einen weiteren Integrationsschritt bilden. Für diesen Prozess gibt es mehrere Herangehensweisen. In der Praxis häufig vertreten ist die *Ad-hoc* Strategie. Dabei werden die einzelnen Bausteine in der zufälligen Reihenfolge ihrer Fertigstellung zusammengefügt. Der Vorteil ist, dass jeder Baustein sofort nach dessen Fertigstellung verwendet wird. Der Nachteil dieser Strategie ist die Verwendung vieler Platzhalter, da für den Test benötigte Bausteine sich noch in der Entwicklung befinden. Bei der strukturierteren Vorgehensweise der *Top-down-Integration* wird mit der Komponente, die weitere Komponenten aufruft, aber selbst nur durch das Betriebssystem aufgerufen wird, begonnen. Das hat den Vorteil, dass keine bzw. nur einfache Testtreiber benötigt werden. Als Testtreiber wird eine Komponente verstanden, welche für die Prüfung eines Bausteins oder Systems benötigt wird. Der Testtreiber interagiert mit der zu testenden Komponente, in dem er z.B. Methoden aufruft. Ist eine untergeordnete Komponente jedoch nicht verfügbar, kann die Erstellung eines Platzhalters sehr aufwendig sein. Den umgekehrten Weg stellt die *Bottom-Up-Integration* dar. Hier wird mit der Komponente begonnen, die keine weiteren Komponenten aufruft. Somit werden keine Platzhalter, dafür aber Testtreiber benötigt. Eine weitere Methode für die Integration der Bausteine ist die *Backbone-Integration*. Bei dieser wird ein Programmskelett erstellt, bei dem schrittweise die zu integrierenden Komponenten hinzugefügt werden. Diese Methode ist flexibel, da die Komponenten in

[98] Vgl. Witte (2016), S. 68.

[99] Vgl. Baker/u.a. (2008), S. 12.

[100] Vgl. Spillner/Linz (2012), S. 52 ff.

beliebiger Reihenfolge dem Skelett zugefügt werden können. Der Nachteil dieser Strategie ist die aufwendige Erstellung des Programmskeletts. [101]

Als Testbasis für den Integrationstest können das Software- und Systemdesign bzw. die Systemarchitektur, wie auch schnittstellenübergreifende Beschreibungen verwendet werden. Vor allem Aktivitäts- und Zustandsdiagramme werden als Basis für diese Tests herangezogen. Die externen Schnittstellen des Systems können auf Basis der Use-Cases geprüft werden. [102]

4.3 Systemtest

Die dritte Teststufe des V-Modells ist der Systemtest. Bei diesem wird erstmalig das gesamte System getestet. Im Unterschied zu den vorherigen Prüfungen wird beim Systemtest aus Sicht des Kunden und nicht des Herstellers geprüft, das heißt, dass validiert wird, ob das System allen gestellten Anforderungen (funktionalen und nicht-funktionalen) entspricht. Es soll aber nicht nur das System in Bezug auf die Anforderungen, sondern auch die Anforderungen selbst, geprüft werden. So sollen sowohl unvollständige oder im System widersprüchlich umgesetzte Anforderungen, wie auch vergessene Anforderungen, identifiziert werden. Als Testbasis können alle Dokumente oder Informationen dienen, die das Testobjekt auf Systemebene beschreiben, wie z.B. System- und Softwareanforderungen. Statt Testtreibern und Platzhaltern soll die im Produktionsbetrieb verwendete Software und Hardware zum Einsatz kommen. Trotzdem soll in einer Testumgebung und nicht im Produktionssystem getestet werden. Sonst könnte es passieren, dass das System ungewollte Auswirkungen auf die Produktionsumgebung hat. [103]

Neben den funktionalen Tests sollte im Zuge der Prüfung der nicht-funktionalen Anforderungen auf acht Aspekte eingegangen werden. Als ersten Punkt sollen die Belastungsgrenzen des Systems getestet werden. Dies geschieht mittels *Lasttest, Performancetest, Massentest* oder *Stresstest*. Wenn das System an seine Leistungsgrenzen gebracht wird, ist es wichtig, dass es nicht abstürzt, sondern in einen kontrollierten Zustand mit einer akzeptablen Antwortzeit übergeht. Des Weiteren ist es wichtig, dass die *Datensicherheit* gewährleistet ist. Unberechtigte dürfen keinen Zugriff auf das System oder deren Daten haben. Als weitere wichtige Aspekte einer Software sind die *Robustheit* und *Stabilität* des Systems anzuführen. Unter Stabilität wird das Verhalten im Dauerbetrieb und das Wiederanlaufverhalten nach einem Systemausfall verstanden. Reagiert das System auf Fehlbedienungen und Falscheingaben angemessen, kann es als robust bezeichnet werden. Der fünfte Aspekt, welcher im Zuge des Systemtests betrachtet werden soll, ist die *Kompatibilität*. Neben der Kompatibilität zu benachbarten Systemen, wird bei diesem Test auch die Einhaltung von Normen, Standards und Vorschriften überprüft. Entspricht ein System zwar allen funktionellen Anforderungen, ist aber nicht *bedienbar*, wird das kein Kunde akzeptieren. Deshalb ist es von Bedeutung, dass die Benutzeroberfläche einfach und übersichtlich aufgebaut ist, die Meldungen verständlich sind und auch das Benutzerhandbuch vollständig und einfach zu verstehen ist. Oftmals wird eine Software nicht nur für eine Systemumgebung entwickelt. In diesem Fall muss die *Übertragbarkeit* gegeben sein. Die Voraussetzungen

[101] Vgl. Spillner/Linz (2012), S. 57 – 60.

[102] Vgl. Spillner/Linz (2012), S. 52.

[103] Vgl. Spillner/Linz (2012), S. 60 – 63.

für eine einwandfreie Funktion in einer Umgebung müssen dokumentiert werden. Der letzte wichtige Aspekt betrifft die *Änderbarkeit*. Damit die Änderbar- und Erweiterbarkeit einer Software gegeben ist, müssen gewisse Coding-Standards eingehalten werden und die Dokumentation muss vollständig sein. [104]

4.4 Abnahmetest

Bei den bisher beschriebenen Tests handelt es sich um Prüfungen, welche vom Hersteller oder in deren Verantwortung durchgeführt werden. Der letzte Test des V-Modells vor Inbetriebnahme der Software, der Abnahmetest, wird üblicherweise vom Kunden oder Anwender durchgeführt, oder sie sind zumindest daran beteiligt. [105]

Es gibt vier typische Formen des Abnahmetests, den *Test auf vertragliche Akzeptanz*, *Test der Benutzerakzeptanz*, *Akzeptanz durch den Systembetreiber* und den *Feldtest (Alpha-* oder *Beta-Test)*. Beim Test auf vertragliche Akzeptanz wird geprüft, ob das System allen vertraglich festgehaltenen Anforderungen entspricht und somit vom Kunden als fehler- und mangelfrei betrachtet wird. Auch die Erfüllung gesetzlicher Vorschriften und Sicherheitsnormen zählt zu den vertraglichen Akzeptanzkriterien. Im Gegensatz zum Systemtest, welcher in einer Testumgebung stattfindet, wird der Abnahmetest in der Abnahmeumgebung des Kunden durchgeführt. Eine Testdurchführung in der Produktivumgebung ist jedoch nach wie vor zu vermeiden. Ein Test auf Benutzerakzeptanz ist zu empfehlen, wenn der Kunde und Anwender nicht dieselben Personen sind. Unterschiedliche Anwendergruppen haben in der Regel unterschiedliche Vorstellungen. Im Zuge des Benutzerakzeptanztests soll die Akzeptanz aller Benutzergruppen gewonnen werden. Diese Tests werden vom Kunden oft selbst organisiert. Ein Test auf Akzeptanz der Systembetreiber soll sicherstellen, dass die neue Software in die bestehende IT-Landschaft eingefügt werden kann. Untersucht werden beispielsweise Backup-Routinen, die Benutzerverwaltung oder Aspekte der Sicherheit. Soll die Software in vielen unterschiedlichen Systemumgebungen zum Einsatz kommen, ist es für den Hersteller sehr kostenintensiv bis unmöglich das System in allen Umgebungen zu testen. In solchen Fällen kommen zusätzlich zu den beschriebenen Tests Feldtests zum Einsatz. Bei Feldtests wird die Software von einem repräsentativen Kundenstamm in der Produktionsumgebung getestet. Entweder werden vom Hersteller durchgeführte Testszenarien durchgeführt, oder die Kunden setzen das vorläufige Produkt probehalber unter realistischen Bedingungen ein. Alpha-Tests finden beim Hersteller, Beta-Tests beim Kunden statt. Besonders für Produkte für den Massenmarkt, sogenannte Commercial-off-the-shelf (COTS)-Systeme, ist dieser Test empfehlenswert. [106]

[104] Vgl. Witte (2016), S. 72 f.

[105] Vgl. Spillner/Linz (2012), S. 64.

[106] Vgl. Spillner/Linz (2012), S. 65 – 67.

5 ANFORDERUNGEN AN DIE SOFTWARE

In diesem Kapitel werden die spezifischen Anforderungen an den *Metris Engineering Configurator* erhoben. Dies beginnt im ersten Unterkapitel mit den vorbereitenden Tätigkeiten. Danach werden die Erhebung der Anforderungen und deren Detaillierung erläutert. Die Dokumentation dessen erfolgt sowohl in Prosa-Text, wie auch in Form von Modellen. Am Ende dieses Kapitels wird beschrieben, wie die erforderliche Qualität der Ziele und Anforderungen sichergestellt wird.

5.1 Vorbereitende Tätigkeiten

Bevor mit der Erhebung der Anforderungen begonnen werden kann, müssen vorbereitende Tätigkeiten getroffen werden. Dazu zählen das Ermitteln der Stakeholder, das Festlegen der Systemgrenzen und das Definieren der Ziele. Die konkreten Ergebnisse und verwendeten Methoden dieser Tätigkeiten werden in diesem Unterkapitel beschrieben.

5.1.1 Stakeholder ermitteln

Die erste Tätigkeit, die im Zuge des Requirements Engineerings für die Entwicklung einer Software zur automatisierten Erstellung der Anlagendokumentation durchgeführt wird, ist die Ermittlung und Dokumentation der Stakeholder. Wie im Unterkapitel 3.2.1 beschrieben, werden darunter alle am Projekt beteiligten Personen verstanden.

In Abbildung 21 sind alle für den *Metris Engineering Configurator* relevanten Stakeholder dargestellt. Die erhobenen Daten orientieren sich an der Empfehlung des IREB. Die Spalte, welche den Namen des Stakeholders enthält, wurde aus Datenschutzgründen entfernt. Somit werden die Stakeholder in dieser Arbeit durch deren *Rolle*, welche in der ersten Spalte sichtbar ist, identifiziert. Sie beschreibt im Gegensatz zur dritten Spalte *Funktion* die Rolle des jeweiligen Stakeholders im aktuellen Projekt. Beispielsweise fungiert ein *Plant Engineer SE*, welcher auch als Spezialist für die Planung von Anlagen für Fest-Flüssig-Trennung bezeichnet werden kann, gleichzeitig als *Test User 1* und *Berater 3*. Die Bedeutung der einzelnen Spalten kann ebenfalls im Unterkapitel 3.2.1 nachgelesen werden. Die Wertigkeit der Zahlen 1-5 in der Spalte *Relevanz* kann in der über der Tabelle zu sehenden Legende nachgeschlagen werden. Zusätzlich zu den in Abbildung 21 dargestellten Informationen werden personenbezogene Daten, wie Kontaktdaten, in MS Outlook verwaltet.

Legende: Relevanz 1-5 (1=für Projekt niedrige Relevanz, 5=für Projekt hohe Relevanz)

Rolle	Unternehmen	Funktion	Verfügbarkeit	Relevanz	Wissensgebiet	Ziele u. Interessen
Projekt Manager / Entwickler	ANDRITZ	Automation Lead Engineer	ständig	5	Comos Key User	anwenderfreundliche u. voll funktionsfähige Software /Einhaltung Budgetrahmen
Auftraggeber	ANDRITZ	Manager Detail Engineering and Tools	ständig	4	Comos Administrator	funktionsfähige Software / möglichst geringe Kosten/ Software soll für andere Bereiche einsetzbar sein
Kunde	ANDRITZ	Senior Project Manager	ständig	3	Filterpressenanlage, Comos Key User	anwenderfreundliche u. voll funktionsfähige Software
Berater 1	ANDRITZ	Comos Administrator	bis Ende 10/2018	1	Comos Administrator	einfache Wartbarkeit der Software
Berater 2	ANDRITZ	Comos Administrator	ständig	3	Comos Administrator/ Systemarchitekt	einfache Wartbarkeit der Software in Bezug auf die IT-Landschaft
Berater 3	ANDRITZ	Plant Engineer SE	ständig	4	Prozess Filterpressenanlage	einfache Wartbarkeit der Templates, anwenderfreundlich
Test User 1	ANDRITZ	Plant Engineer SE	ständig	4	Prozess Filterpressenanlage	einfache Wartbarkeit der Templates, anwenderfreundlich
Test User 2	ANDRITZ	Comos User	ständig außer 09/2018	4	Detail Engineering / Comos User	einfach zu bedienende Software
Test User 3	ANDRITZ	Comos Key User	ständig	4	Detail Engineering / Comos Key User	einfach Wartbarkeit der Templates
Test User 4	ANDRITZ	Comos Administrator	ständig	3	Comos Administrator/ Systemarchitekt	einfache Wartbarkeit der Software in Bezug auf die IT-Landschaft
Projekt Manager	SIEMENS	Head of Delivery Service AT	ständig	4	PM	funktionsfähige Software / möglichst wenig Aufwand
Entwickler Lieferant	SIEMENS	Delivery Service AT	bis Ende 11/2018	5	Comos Entwickler	funktionsfähige Software / möglichst wenig Aufwand
Sales Manager	SIEMENS	Enterprise Sales Manager (Comos)	bis Ende 10/2018	4	Vertrieb	funktionsfähige Software, die auch für andere Kunden in abgeänderter Form verwendet werden kann

Abbildung 21: Stakeholderliste, Quelle: Eigene Darstellung.

In der ersten Zeile der Stakeholderliste in Abbildung 21 ist der *Projekt Manager* dieses Projektes, welcher gleichzeitig als *Entwickler* fungiert, angeführt. Er ist bei der Fa. *ANDRITZ* als *Automation Lead Engineer* beschäftigt. Für das Projekt hat er eine hohe Relevanz (*5*) und das Wissensgebiet wird mit *Comos Key User* beschrieben. Das bedeutet, dass über das Engineering Programm Comos sehr gute Kenntnisse vorhanden sind. Das Hauptziel des *Projekt Managers* ist der erfolgreiche Abschluss des Projektes. Das inkludiert zum einen qualitative Ziele, wie eine *anwenderfreundliche und voll funktionsfähige Software* und zum anderen die *Einhaltung des Budgetrahmens*. In der zweiten Zeile der Anforderungsliste ist der *Auftraggeber* dieses Projektes, welcher die Funktion *Manager Detail Engineering and Tools* im Unternehmen *ANDRITZ* inne hat, angeführt. Sein Wissensgebiet ist als *Comos Administrator* im Engineering Programm Comos sehr gut ausgeprägt. Er hat ähnliche Ziele und Interessen wie der Projekt Manager. Zusätzlich zu den qualitativen und kaufmännischen Zielen des Projekt Managers erwartet er sich, dass die zu entwickelnde Software leicht erweiterbar ist. Das bedeutet, dass sie ohne großen Änderungsaufwand für unterschiedliche Anlagen eingesetzt werden kann.

Der Kunde dieses Projektes ist ebenfalls im Unternehmen ANDRITZ beschäftigt. Seine Kernkompetenzen erstrecken sich über den Bereich der *Filterpressenanlagen* und *Comos*. Er hat eine etwas niedrigere Relevanz als der Projekt Manager. Im Gegensatz zu den vorher beschriebenen Stakeholdern entfallen beim Kunden die kaufmännischen Interessen. Er fordert eine *anwenderfreundliche und voll funktionsfähige Software*. Des Weiteren werden drei *Berater* und vier *Test User* in der Anforderungsliste angeführt. Das Wissensgebiet der *Berater* umfasst das Engineering Programm Comos (*Berater 1* und *2*) und den Prozess einer Filterpressenanlage (*Berater 3*). Den *Beratern 1* und *2*, welche die Funktion *Comos Administrator* im Unternehmen ausfüllen, ist es wichtig, dass die einfache Wartbarkeit der Software an sich und in Bezug auf Comos und der IT-Landschaft gegeben ist. Da *Berater 3* vermutlich die Templates warten wird, ist ihm wichtig, dass die Software die Voraussetzungen für eine einfache Durchführung dieser Tätigkeit schafft. Zusätzlich ist für ihn eine einfache Bedienbarkeit der Software von Bedeutung. Da wie bereits beschrieben, der *Berater 3* und der *Test User 1* dieselbe Person sind, verfolgen Sie auch dieselben Ziele. *Test User 2* und *3* füllen die *Funktion* eines *Comos Users* und eines *Comos Key Users* beim Unternehmen *ANDRITZ* aus. Da ein *Comos User* die Software hauptsächlich nutzen und ein *Comos Key User* die Templates auch warten wird, beziehen sich die Interessen der *Test User 2* und *3* auf diese Tätigkeiten. Der *Test User 4* ist dieselbe Person, wie der *Berater 2* und hat somit dasselbe Wissensgebiet und verfolgt dieselben Ziele.

Zusätzlich zu den unternehmensinternen Stakeholdern, werden auch drei unternehmensexterne Stakeholder berücksichtigt. Die Fa. *SIEMENS* agiert bei diesem Projekt als Lieferant der Grundfunktionalitäten der zu entwickelnden Software in Form eines Source Codes. Dieses Team setzt sich aus dem *Projekt Manager* auf der Lieferantenseite, einem *Sales Manager* und einem oder mehreren *Entwicklern* zusammen. Da der *Projekt Manager* und der Hauptentwickler selbst entscheiden, ob es notwendig ist, zusätzliche Entwickler hinzuzuziehen und die Kommunikation zwischen ANDRITZ und SIEMENS immer über denselben Entwickler auf der Lieferantenseite abgewickelt wird, wird in der Stakeholderliste nur ein Entwickler angeführt. Der *Projekt Manager* und der *Entwickler* der Fa. *SIEMENS* verfolgen die Ziele, dass die Software voll funktionsfähig ist und die Entwicklung auch möglichst wenig Aufwand bedeutet. Der *Sales Manager* hat zusätzlich Interesse an der Wiederverwendbarkeit der Software für andere Kunden. Außer dem *Berater 1*, dem *Test User 2*, dem *Entwickler Lieferant* und dem *Sales Manager* sind alle Stakeholder ständig verfügbar.

5.1.2 Projektauftrag

Da das Unternehmen ANDRITZ die meisten aller Kundenaufträge in Form von Projekten abwickelt, ist der schriftliche Projektauftrag im Unternehmen weit verbreitet. Aus diesem Grund werden die generellen Ziele und die Systemgrenzen (Nicht-Ziele) als Prosa-Text in Form eines Projektauftrages dokumentiert. Der erste Schritt *Systematische Suche nach den Stakeholdern* im Prozess der Zieldefinition, welcher im Unterkapitel 3.2.2 beschrieben wird, wird bereits im Unterkapitel 5.1.1 behandelt. Aufgrund dessen, dass die Hauptaufgabe des *Metris Engineering Configurators* darin besteht, eine manuelle Tätigkeit zu automatisieren, ist es sinnvoll, auch auf die zweite Tätigkeit, die *Erhebung der Ist-Situation*, näher einzugehen. Für diese Aufgabe wurde die Methode des Apprenticing genutzt. Die Theorie zu den Ermittlungstechniken wird im Unterkapitel 3.3.1 näher erläutert.

Projektstartereignis: • Mündlicher Auftrag vom PAG (ANDRITZ AG)	**Projektstarttermin:** • 5.3.2018
Projektendereignis: • Abschlusspräsentation im Unternehmen	**Projektendtermin:** • 7.2.2019
Projektziele: Hauptziele: • (Z1) P&ID's, Stromlaufpläne, Funktionspläne und zugehörige Listen einer Filterpressenanlage können vollautomatisch erstellt werden • (Z2) Die Software ist einfach bedienbar (von Comos User) und wartbar (von Comos Key-User) • (Z3) Die Anlage kann im Comos oder über ein Excel-Interface vollständig konfiguriert werden Nebenziele: • Kompetenzen für die Abwicklung eines Entwicklungsprojektes werden erweitert (Requirement-, Test Engineering etc.)	**Nicht-Projektziele:** • (N1) Software ist für andere Anlagen außer Filterpressenanlagen verwendbar • (N2) Alle Eventualitäten einer Filterpressenanlage werden abgedeckt (Vergleich Kosten/Nutzen) • (N3) Software ist eigenständig oder mit anderen Programmen außer Comos einsetzbar

Abbildung 22: Projektauftrag - Ausschnitt, Quelle: Eigene Darstellung.

Nach der *Bewertung der Ist-Situation* wurden die Ziele, welche in Abbildung 22 ersichtlich sind, niedergeschrieben. Das erste Hauptziel *(Z1)* beschreibt die generelle Funktion der neuen Software. Hier ist zu sehen, dass unterschiedliche technische Dokumente einer Filterpressenanlage automatisiert erstellt werden sollen. Das Hauptziel zwei *(Z2)* bezieht sich auf die Bedien- und Wartbarkeit der Software. Die Konfiguration der Anlage über Comos oder ein Excel-Interface, welches das dritte Hauptziel *(Z3)* ist, stellt ein relativ spezifisches Ziel dar. Nebenziele betreffen oftmals nicht direkt das Projekt, sondern stellen in diesem Fall eine Erhöhung der unternehmensinternen Kompetenzen dar. Der dritte wichtige Punkt, betreffend den vorbereitenden Tätigkeiten des Requirements Engineerings, ist die Abgrenzung des Systems. Die Theorie zu diesem Thema kann im Unterkapitel 3.2.3 nachgelesen werden. Die Dokumentation dieser Abgrenzung erfolgt in Form von *Nicht-Projektzielen* im Projektauftrag. Alle Punkte, die in obiger Abbildung gelistet sind, wurden vom System explizit ausgeschlossen und werden somit in den Zielen nicht berücksichtigt. Jedoch darf der Systemkontext, zu dem die Nicht-Projektziele gezählt werden, nicht unbeachtet bleiben. Beispielsweise ist die Verwendbarkeit der Software für andere Anlagen außer Filterpressen nicht Teil dieses Projektes *(N1)*, die einfache Erweiterbarkeit soll aber gegeben sein. Weiters soll der Umfang des Systems bzw. des Projektes insofern abgegrenzt werden, sodass nicht alle Eventualitäten einer Filterpressenanlage abgedeckt werden können *(N2)*. Das bedeutet, dass z.B. eine Option, die nur in 2 % aller Filterpressenanlagen zum Einsatz kommt, nicht berücksichtigt wird. Die Erstellung der Templates würde in diesem Fall

gegenüber der manuellen Nachbearbeitung der Anlagendokumentation auch auf lange Sicht einen wesentlich höheren Aufwand bedeuten. Das dritte Nicht-Projektziel *(N3)* nimmt auf die Einsatzumgebung der zu entwickelnden Software bezug. Hier ist definiert, dass die neue Software nur in Kombination mit dem bei ANDRITZ verwendeten Engineering Programm Comos einsetzbar ist. Zusätzlich sind im Projektauftrag wichtige Eckdaten wie der zeitliche und budgetäre Rahmen und eine kurze Projektbeschreibung angeführt. Weitere Informationen, die im Projektauftrag enthalten sind, können im Anhang 1 nachgeschlagen werden.

5.2 Erheben der Anforderungen

Wurden die Rahmenbedingungen für ein erfolgreiches Requirements Engineering geschaffen, kann mit den Kerntätigkeiten des Requirements Engineerings begonnen werden. Die ersten groben Anforderungen für diese Masterarbeit wurden mit den Methoden des Interviews und des Brainstormings erhoben. Die theoretischen Grundlagen zu diesen Themen können im Unterkapitel 3.3.1 nachgelesen werden. Es wurden zwei unterschiedliche Formen für die Dokumentation der erhobenen Anforderungen gewählt. Zum einen die Anforderungsliste, zum anderen das Use-Case-Diagramm. Die Ergebnisse beider Dokumentationsformen werden in den folgenden zwei Unterkapiteln erläutert.

5.2.1 Anforderungsliste

Unter der Anforderungsliste wird eine zweidimensionale strukturierte Dokumentationsform in Prosa-Text verstanden. Der Vorteil der Anforderungsliste gegenüber einer modellbasierten Darstellung besteht darin, dass mit dieser auch nicht-funktionale Anforderungen dokumentiert werden können. Weitere theoretische Grundlagen zur Anforderungsliste können im Unterkapitel 3.3.2 nachgeschlagen werden.

In Abbildung 23 ist die Anforderungsliste, die im Zuge dieser Masterarbeit erstellt wurde, zu sehen. Die Anforderung mit der ID *1* stellt ein generelles Ziel dieses Projektes dar. Sie beschreibt, dass es mit der neuen Software möglich sein soll, alle relevanten Anlagendokumentationen vollautomatisch zu erstellen. Zusätzlich zur theoretischen Empfehlung in der Literatur (siehe Unterkapitel 3.3.2.2) wurde die aktuelle Anforderungsliste um die dritte und vierte Spalte erweitert. In der dritten Spalte ist die Zuordnung der Anforderungen zu den im Unterkapitel 5.1.2 beschriebenen Zielen ersichtlich. Mit der vierten Spalte wird die Einteilung der Anforderungen vorgenommen. Jede Anforderung wird zum einen nach ihrer Art und zum anderen nach dem Modell von Kano eingestuft. Beide Einteilungsarten werden im Unterkapitel 3.1.3 beschrieben. Die Legende zu den Abkürzungen in der dritten Spalte kann in Abbildung 23 über der Tabelle nachgelesen werden. Die achte Spalte beschreibt den Status jeder Anforderung. So ist zu sehen, dass die Anforderung *1* (die Abkürzung *ID* wird aufgrund der einfacheren Lesbarkeit im Folgenden weggelassen) mit dem *Release 0.3* umgesetzt wurde. Ein Release stellt einen zeitlich determinierten Meilenstein dar. Des Weiteren wurde dieser Anforderung die Priorität *hoch* zugewiesen.

Legende: allgemeine Einteilung: F - funtionale Anf., B - Anf. an Benutzerschnittstelle, Q - Qualitätsanf., T - technische Anf., sL - Anf. Sonstige Lieferbestandteile
Einteilung nach Kano: Ba - Basisanf., Bf - Begeisterungsfaktoren, La - Leistungsanf.

ID	Anforderung	Ziel	Einteilung	eingestellt von	eingestellt am	Priorität	Status	Anmerkungen
1	Erstellung der Anlagendokumentation soll vollautomatisiert ablaufen (keine Nachbearbeitung nötig)	Z1	F, La	PM / Entwickler	22.03.2018	hoch	umgesetzt in Release 0.3	
2	einfache, fehlerfreie Bedienung -> Software muss Userfehler ausschließen	Z2	B, Bf	PM / Entwickler	22.03.2018	hoch	umgesetzt in Release 0.2	
3	Erstellung / Änderung der Templates soll einfach sein (von Key User nach kurzer Einschulung bedienbar)	Z2	Q, La	PM / Entwickler	22.03.2018	mittel	umgesetzt in Release 0.4	
4	Konfiguration der Anlage über Excel-Interface oder ähnlich weit verbreitetes Tool	Z3	B, La	PM / Entwickler	22.03.2018	hoch	umgesetzt in Release 0.4	
5	Anlagendokumentation soll erstellt werden können, ohne Comos zu öffnen	Z3	F, Bf	PM / Entwickler	22.03.2018	niedrig	Umsetzung geplant in Release 1.1	eventuell über Enterprise Server/XML
6	Software soll für andere Anlagen/Produkte einfach erweiterbar sein	Z2	Q, Ba	Auftraggeber	22.03.2018	mittel	umgesetzt in Release 0.4	
7	soll mit Comos V10.1 lauffähig sein		T, Ba	PM / Entwickler	22.03.2018	hoch	umgesetzt in Release 0.1	
8	automatisches Updaten + Revisionieren der Dokumente bei Änderung der Konfiguration	Z1	F, Bf	Auftraggeber	13.04.2018	niedrig	Umsetzung geplant in Release 1.1	
9	Prozessmodell der Software soll erstellt werden	Z2	sL, La	PM / Entwickler	13.04.2018	hoch	umgesetzt in Release 0.1	
10	offenes Interface (XML)	Z2	B, La	Auftraggeber	13.06.2018	hoch	umgesetzt in Release 0.3	
11	mit Software erstellte Anlagen müssen wie manuell erstellte Projekte im Comos modifizierbar sein (kopierbar, einzelne Teile verschieben löschen etc.)	Z2	F, Ba	PM / Entwickler	15.06.2018	hoch	umgesetzt in Release 0.1	mit Auflösen der e-Blocks erledigt
12	Aktionen rückgängig machen	Z2	F, Ba	PM / Entwickler	15.06.2018	mittel	umgesetzt in Release 0.3	
13	verschiedene Versionen in Verkaufsphase (Working Layer), in Engineering Phase wird ein Working Layer freigegeben	Z1	F, La	Kunde	03.07.2018	mittel	umgesetzt in Release 0.3	
14	Ein Projekt kann mehrere Pressen (Linien) enthalten - muss im Konfigurator auswählbar sein	Z1	F, La	Berater 3	16.08.2018	mittel	Umsetzung geplant in Release 1.2	
15	Lieferumfang muss im Excel oder ähnlichem Tool auswählbar sein (eigene Gruppen) -> Kapitelkarte "Scope of Supply" und Farbe am P&ID ändern sich	Z1	F, La	Berater 3	22.10.2018	niedrig	Umsetzung geplant in Release 1.1	

Abbildung 23: Anforderungsliste, Quelle: Eigene Darstellung.

Die Anforderungen *5* und *8* beschreiben zwei weitere funktionale Anforderungen und wurden als Begeisterungsfaktoren deklariert. Zum einen soll die Dokumentation erstellt werden können ohne Comos zu

öffnen. Somit könnte die Software auch von Usern ohne Comos-Zugang verwendet werden. Zum anderen soll das Erstellen einer neuen Revision der Dokumente bei Ändern der Konfiguration von der Software automatisiert durchgeführt werden. Beide Anforderungen sind für die generelle Funktion und den generellen Nutzen der Software nicht essenziell und wurden daher mit der Priorität *niedrig* gekennzeichnet. Aufgrund des Zeitmangels in der Entwicklung werden diese Anforderungen nach dem Launch der Software (Release 1.0) im Zuge des *Release 1.1* umgesetzt.

Die Anforderungen *2* und *4* beziehen sich auf die Anforderungen an die Benutzerschnittstelle. Sie fordern eine Excel-Schnittstelle bzw. eine Schnittstelle über ein ähnlich weit verbreitetes Tool und eine einfache und fehlerfreie Bedienung. Wie im Unterkapitel 5.2.2 erläutert, wurde im Zuge der Entwicklung die Excel-Schnittstelle durch eine XML-Schnittstelle (entspricht der Anforderung *10*) ersetzt. Die Anforderung *2* bedeutet, dass die Software möglichst keine Fehleingaben zulässt. Dies kann durch Auswahllisten oder durch Überprüfung der Eingaben inklusive Hinweisfeld erfolgen. Auch an die Qualität werden Anforderungen gestellt. Die Anforderungen *3* und *6* betreffen Wart- und Erweiterbarkeit der Software. Erstens soll es einfach sein, die Templates zu ändern, zweitens soll die Erweiterbarkeit der Anwendung der Software auf andere Produkte gegeben sein. Da diese Anforderungen von den Stakeholdern als selbstverständlich gesehen werden, wurden sie als Basisanforderungen deklariert. Eine weitere wichtige Voraussetzung für eine qualitativ hochwertige und von den Stakeholdern akzeptierte Software stellt die technische Anforderung *7* dar. Sie beschreibt die Systemumgebung, in der die neue Software zum Einsatz kommen soll.

Der *Metris Engineering Configurator* soll eine offene Schnittstelle in Form von XML enthalten (Anforderung *10* der Abbildung 23). Mit dieser soll unter anderem die zukünftige Interaktion mit anderen Programmen vorbereitet werden. Eine Anforderung an sonstige Lieferbestandteile stellt die Anforderung *9* dar. Sie fordert ein Prozessmodell, dass den Ablauf der Software darstellt. Eine gute Dokumentation ist Voraussetzung für die Änderung bzw. Erweiterung einer Software. Daher wurde diese Anforderung dem Ziel 2 zugewiesen. Die restlichen Anforderungen, welche in Abbildung 23 dargestellt sind, beziehen sich allesamt auf die Funktion der Software. Die Basisanforderung *11* stellt Anforderungen an die manuelle Nachbearbeitung. Es soll möglich sein, die Dokumente in gleicher Weise zu modifizieren, wie wenn sie manuell erstellt worden sind. Außerdem soll es möglich sein, Aktionen rückgängig zu machen (Anforderung *12*), d.h. dass unbeabsichtigte Eingaben eines Users zurückgenommen werden können.

Es ist möglich, dass in der Verkaufsphase dem Kunden verschiedene Versionen einer Anlage angeboten werden. Bereits in dieser Phase sollen dem Kunden fertige Dokumente vorgelegt werden können. Entscheidet sich der Kunde für eine bestimmte Version, wird diese für das weitere Engineering freigegeben. Eine Idee zur Umsetzung dieser Funktion ist die Verwendung der im Comos bereits vorhandenen sogenannten Working Layer. Die Anforderung *13* beschreibt diese Funktion. Der Lieferumfang des Kunden, von ANDRITZ oder Lieferanten wird bei Filterpressenanlagen üblicherweise am P&ID durch unterschiedliche Farben dargestellt. Diese Farben werden durch die Spezifikation *Scope of Supply* von Objekten im Comos gesteuert. Die Anforderung *15* bezieht sich auf diese Funktion und setzt zusätzlich voraus, dass Objekte zu Liefergruppen zusammengefasst werden und ein Auswahlfeld pro Liefergruppe im Zuge der Konfiguration der Anlage befüllt werden kann. Die Umsetzung dieser Anforderung wurde aufgrund des späten Einstellungsdatums *22.10.2018* und des relativ knappen Zeitfensters auf das *Release 1.1* verschoben. Anforderung *14* beschreibt eine Funktion, deren Bedarf am Anfang des Requirements Engineerings ebenfalls

nicht bekannt war und erst durch Diskussion des Modelles, welches im Unterkapitel 6.2.2 dargestellt wird, aufgekommen ist. Es besteht die Möglichkeit, dass ein Filterpressenprojekt aus mehreren Linien besteht. Eine Linie enthält immer eine Filterpresse und entweder separate oder mit den anderen Linien gemeinsam verwendete Optionen. Diese Zusammenhänge werden im Unterkapitel 6.2 genauer beschrieben. In der Umsetzung stellt diese Anforderung eine große Herausforderung dar, da sie mit anderen Funktionen des *Metris Engineering Configurators* nicht vereinbar ist. Als diese Anforderung bekannt wurde, waren die Grundfunktionalitäten der neuen Software bereits entwickelt. Daher wurde nach Rücksprache mit dem Urheber dieser Anforderung (*Berater 3*) vereinbart, dass diese Anforderung erst mit *Release 1.2* weiter behandelt wird.

Die genauen Daten zu den Release-Terminen, welche in der Anforderungsliste angeführt sind, sind im Meilensteinplan, welcher in Abbildung 24 ersichtlich ist, dargestellt. Jedem Meilenstein in Form eines Release-Termines werden drei Daten zugeordnet, der *Plantermin*, der *neue Plantermin* und der *Ist-Termin*. In der Spalte *Plantermin* ist der ursprünglich geplante Umsetzungstermin der jeweiligen Anforderung definiert. Konnte dieser Termin nicht eingehalten werden, wird ein *neuer Plantermin* festgelegt. In der letzten Spalte *Ist-Termin* ist der tatsächliche Umsetzungszeitpunkt angeführt. Sechs von sieben Release-Terminen werden in der Anforderungsliste verwendet. Das *Release 1.0 (Product Launch)* beschreibt den Zeitpunkt der ersten Produktfreigabe. Aufgrund dieser zeitlichen Staffelung konnte die Zeit zwischen dem 09.11.2018 und dem 19.11.2018 für Tests und kleine Änderungen bzw. Fehlerkorrekturen genutzt werden. So konnte eine qualitativ hochwertige Software freigegeben werden. Das *Release 1.1* stellt bereits ein erstes Update der Software dar. Mit diesem werden zum einen die zusätzlich geplanten Funktionen, welche in der Anforderungsliste definiert sind, freigegeben. Zum anderen werden diverse Fehler oder Verbesserungswünsche, die erst im Betrieb aufkommen, korrigiert bzw. berücksichtigt. Beim zweiten Update, welches als *Release 1.2* bezeichnet wird, wird die Anforderung 14 der Anforderungsliste behandelt.

ID	Release	Plantermin	neuer Plantermin	Ist-Termin
1	Release 0.1	22.06.2018		29.06.2018
2	Release 0.2	07.09.2018		07.09.2018
3	Release 0.3	05.10.2018	25.10.2018	25.10.2018
4	Release 0.4	09.11.2018		09.11.2018
5	Release 1.0 (Product Launch)	19.11.2018		19.11.2018
6	Release 1.1	21.12.2018		
7	Release 1.2	24.01.2019		

Abbildung 24: Meilensteinplan, Quelle: Eigene Darstellung.

5.2.2 Use-Case-Diagramm

Mit dem Use-Case-Diagramm werden die Hauptaufgaben des Systems modellbasiert dargestellt. Die theoretischen Grundlagen des Use-Case-Diagrammes können im Unterkapitel 3.4.4.1 nachgeschlagen werden. In nachfolgender Abbildung 25 sind die Hauptaufgaben des Systems *Metris Engineering Configurator*, welches der im Zuge dieser Masterarbeit entwickelten Software entspricht, dargestellt. Die Hauptaufgabe *neue Anlage oder neue Version einer Anlage konfigurieren* beschreibt die Tätigkeit der Konfiguration einer neuen Anlage oder die Erstellung einer neuen Version einer bereits konfigurierten Anlage. Zweiteres hängt mit der Anforderung 13 der im vorhergehenden Unterkapitel beschriebenen Anforderungsliste zusammen.

Die Hauptaufgabe *neue Anlage oder neue Version einer Anlage konfigurieren* besitzt eine Assoziation mit der Hauptaufgabe *Anlegen der Objekte/Dokumente*. In diesem Fall ist der Abschluss der ersten Hauptaufgabe die Voraussetzung für den Start der zweitgenannten. Im Zuge dieser Hauptaufgabe werden alle relevanten Objekte und Dokumente im Comos angelegt. Weiters sind im Use-Case-Diagramm die möglichen Benutzer- und Systemschnittstellen ersichtlich. So ist zu sehen, dass ein *Comos User* oder das System *XML*, welches die XML-Schnittstelle (Anforderung 4 und Anforderung 10) repräsentiert, eine *neue Anlage oder neue Version einer Anlage konfigurieren* kann. Im Zuge der Realisierung wurde die Excel-Schnittstelle aufgrund der Komplexität der Benutzeroberfläche verworfen. Stattdessen wird ausschließlich das XML-Interface verwendet. Für die grafische Darstellung der Daten wird eine .NET basierte WPF-Anwendung (Windows Presentation Forms – Anwendung) entwickelt. Da zwischen dem *Comos User* und dem *Comos Key-User* eine Generalisierungsbeziehung besteht, erbt der *Comos Key-User* die Eigenschaften des *Comos Users* und kann ebenfalls diese Hauptaufgabe starten.

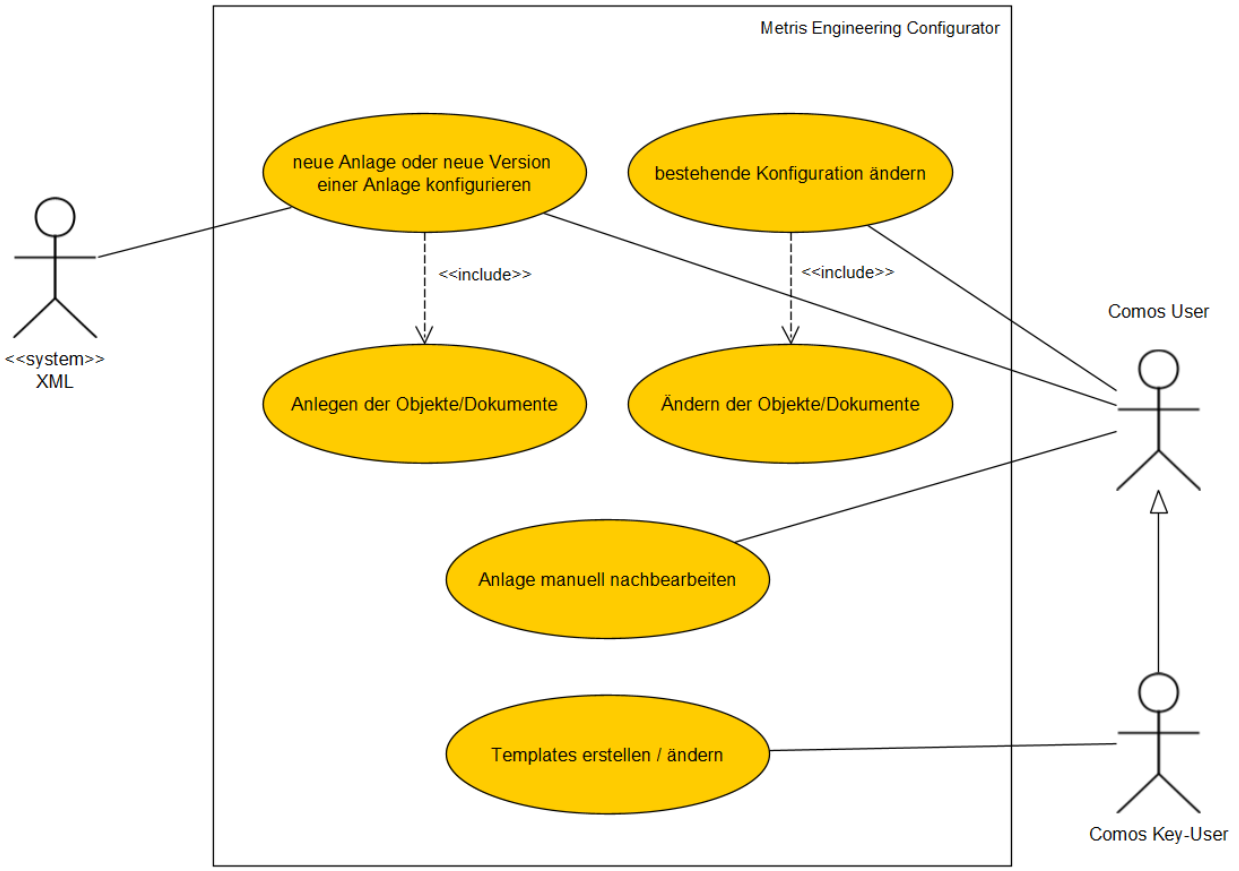


Abbildung 25: Use-Case-Diagramm, Quelle: Eigene Darstellung.

Die Hauptaufgabe *bestehende Konfiguration ändern* kann nur von einem *Comos User* oder *Comos Key-User* ausgeführt werden. Mit dieser Hauptaufgabe und der darauffolgenden Hauptaufgabe *Ändern der Objekte/Dokumente* können bestehende Konfigurationen geändert werden. Sollte es notwendig sein, kann ein *Comos User* und somit auch ein *Comos Key-User* die *Anlage manuell nachbearbeiten*. Die Hauptaufgabe *Templates erstellen / ändern* kann nur von einem *Comos Key-User* durchgeführt werden. Diese Hauptaufgabe ist auch in der Anforderungsliste unter Punkt 3 angeführt. In dem abgebildeten Use-Case-Diagramm werden nur die wichtigsten Aufgaben dargestellt. Aus diesem Grund sind die Anforderungen aus der

Anforderungsliste und die Anforderungen des Use-Case-Diagramms nicht äquivalent. Darüber hinaus können in dem Use-Case-Diagramm nur funktionale Anforderungen abgebildet werden.

5.3 Detaillierung der Anforderungen

In diesem Unterkapitel werden die Anforderungen aus der Anforderungsliste und dem Use-Case-Diagramm detailliert. Des Weiteren werden sie modellbasiert aus den drei Perspektiven, der Struktur-, der Funktions- und der Verhaltensperspektive, dokumentiert. Das vierte Modell, das Präsentationsmodell, definiert die Anforderungen an die Benutzeroberfläche. Die Bedeutung dieser Perspektiven und die generelle Theorie zur Modellbildung und allen verwendeten Diagrammtypen kann im Unterkapitel 3.4 – Detaillierung der Anforderungen / Modellbildung nachgelesen werden. Außerdem wird im Unterkapitel 3.4 die Methodik zur Detaillierung der Anforderungen beschrieben.

5.3.1 Statische Struktur der Software

Die statische Struktur des *Metris Engineering Configurators* wird in dieser Arbeit mithilfe eines UML-Klassendiagrammes beschrieben. Dadurch, dass die neu entwickelte Software in das Engineering Programm Comos integriert wird, muss neben der Struktur der neuen Software auch die Struktur von Comos betrachtet werden. Zuerst wird in diesem Unterkapitel eine Übersicht der Struktur von Comos inklusive der Struktur der neuen Software vorgestellt. Danach wird auf die einzelnen Klassen der neuen Software näher eingegangen. Es werden aber nicht alle im Comos verfügbaren Klassen behandelt, sondern nur diejenigen, die für den *Metris Engineering Configurators* von Bedeutung sind. Die Teile des Diagrammes, die für die Entwicklung der neuen Software nicht essenziell sind, können im Anhang 2 betrachtet werden.

In nachfolgender Abbildung 26 sind die hierarchisch höchsten bestehenden und neuen Klassen dargestellt. Die Klassen, welche im Comos bereits vorhanden sind, sind orange, die Klassen der neuen Software gelb hinterlegt. Comos ist ein objektorientiertes Engineering Programm. Aus diesem Grund ist ein *Comos-Objekt* die generellste Klasse im gesamten Programm. Jedes *Comos-Objekt* beinhaltet zumindest drei Attribute, einen *Namen*, ein *Label* und eine *Beschreibung*. Jede weitere Klasse erbt die Eigenschaften dieses generellen *Comos-Objektes*. Somit entsteht eine hierarchische Struktur. Die *Comos-Objekte* werden zwar als Objekte bezeichnet, besitzen laut der Definition von Unterkapitel 3.4.3.1 aber die Eigenschaften einer Klasse.

In der zweiten Hierarchieebene finden sich das *Struktur-Objekt*, das *Comos-Dokument*, der *logische Operator*, sowie das *Baugruppen Objekt*. Mit Hilfe der *Struktur-Objekte* kann die Struktur einer Anlage vollständig beschrieben werden. Ein *Orts-Objekt*, welches eine spezielle Klasse eines *Struktur-Objektes* darstellt, beschreibt die Örtlichkeiten der Komponenten einer Anlage im Engineering Programm Comos. Beispielsweise wird dadurch der Aufstellungsort eines Schaltschrankes definiert. Eine weitere Klasse, deren Superklasse ein *Struktur-Objekt* ist, ist eine *Funktion*. Eine *Funktion* ist eine Mess- und Regelstelle, welche auch am P&ID dargestellt werden kann. Das *Anlagen-Objekt* wird in diesem Unterkapitel noch detailliert beschrieben.

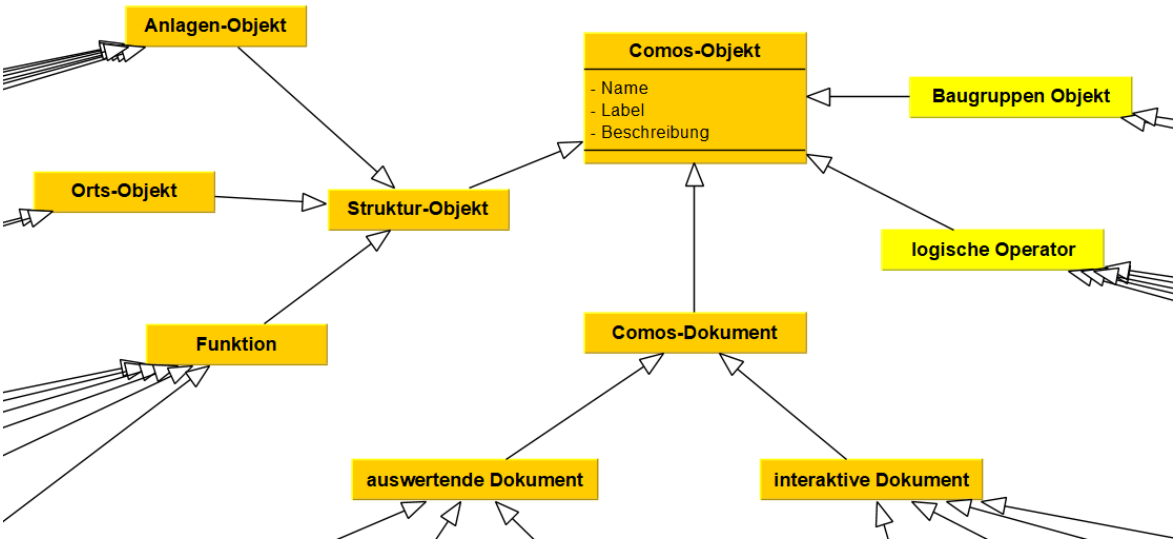


Abbildung 26: UML-Klassendiagramm – Übersicht, Quelle: Eigene Darstellung.

Als zweite Klasse der zweiten Hierarchieebene ist in obiger Abbildung das *Comos-Dokument* angeführt. Generell wird hier zwischen zwei Arten, dem *auswertenden Dokument* und dem *interaktiven Dokument* unterschieden. Unter *auswertendes Dokument* werden Dokumente verstanden, die Informationen von Objekten ausschließlich lesen, wie z.B. eine Stückliste. Interaktive Dokumente dagegen können Informationen der Objekte lesen und schreiben. So können beispielsweise auf einem Stromlaufplan Verbindungsinformationen der elektrischen Bauteile sowohl gelesen, wie auch geschrieben werden. Ein weiteres *interaktives Dokument*, der *Verknüpfungsplan,* wird in diesem Unterkapitel noch näher erläutert. Die *Baugruppen Objekte* und *logischen Operatoren*, welche für die neue Software entwickelt werden, werden auf den nächsten Seiten erklärt.

In Abbildung 27 wird das *Anlagen-Objekt* und dessen Subklassen illustriert. Unter einem *Vorlagenobjekt* werden Objekte im Comos verstanden, mit denen die Templates zusammengefasst werden. Unter anderem entspricht ein *Vorlagenobjekt* einem Variation Point. Das Pfadmakro gruppiert die Objekte einer Variante oder einer Komponente. Die Bedeutung der Variation Points, der Varianten und der Komponenten werden im Unterkapitel 6.2.1 näher erläutert. Die dritte neu entwickelte Klasse der Abbildung 27 ist das *Merge Objekt*. Die Objekte der Variation Points bzw. der Komponenten können in der Anlagenstruktur des Comos nicht strikt voneinander getrennt werden. Mit Hilfe des *Merge Objektes* werden jene Objekte in einem Template gruppiert, welche in die Anlagenstruktur eingeordnet werden müssen. Die Klassen *Anlage, Equipment* und *Schrank* stellen weitere *Anlagen-Objekte* zur funktionalen Gliederung einer Anlage dar. Ein *Schrank* kann sowohl ein elektrischer Schaltschrank, wie auch ein Pneumatik- oder Hydraulikschrank sein.

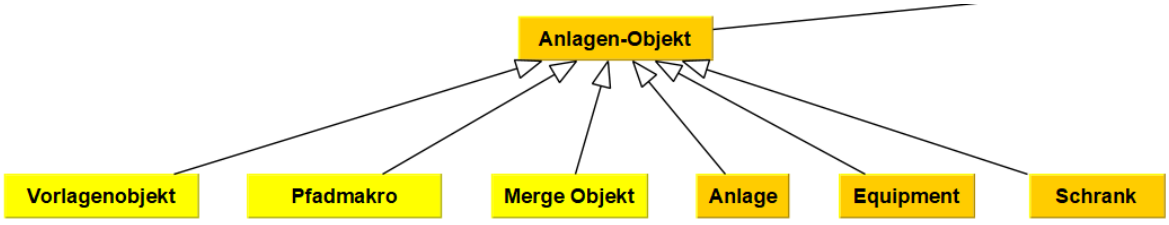


Abbildung 27: Anlagen-Objekt, Quelle: Eigene Darstellung.

Die Klassen, welche die Hauptfunktionalitäten des *Metris Enginering Configurators* beinhalten, werden mit der Klasse *Baugruppen-Objekt,* welche in Abbildung 28 ersichtlich ist, zusammengefasst. Die zwei wesentlichsten Elemente der *Baugruppen-Objekte* sind die *eBlocks* und die *Platzhalter (Graybox)*. Die spezialisierten Instanzen der Klasse *eBlock* führen die wesentlichen Funktionen der neuen Software aus. Beispielsweise kann ein Objekt der Klasse *Objekte löschen* beliebige Objekte im Comos löschen. Das Attribut *Zielobjekt,* welches direkt durch einen *Link* oder durch einen *Suchtext* befüllt wird, legt fest, welches Objekt gelöscht werden soll. Zusätzlich werden alle Objekte, welche dem Zielobjekt hierarchisch untergeordnet sind, gelöscht. Die Klasse *Objekte in Anlagenstruktur verschieben (Merge)* verwendet zusätzlich das Attribut *Quellobjekt*. Mit diesem werden die zu verschiebenden Objekte definiert. Die Objekte werden aber nicht nur verschoben, ferner werden die zu verschiebenden Objekte mit den existierenden Objekten, definiert durch das Attribut *Zielobjekt*, verglichen. Existiert bereits ein Objekt mit demselben Namen, wird das zu verschiebende Objekt verworfen, ansonsten wird es verschoben. Üblicherweise wird dem Attribut *Quellobjekt* ein Objekt der Klasse *Merge Objekt* zugewiesen. Die Notwendigkeit der Klasse *Objekte in Anlagenstruktur verschieben (Merge)* ergibt sich durch die Funktion der Objekte der Klassen *eine Baugruppe einkopieren*, die im nächsten Absatz erläutert wird.

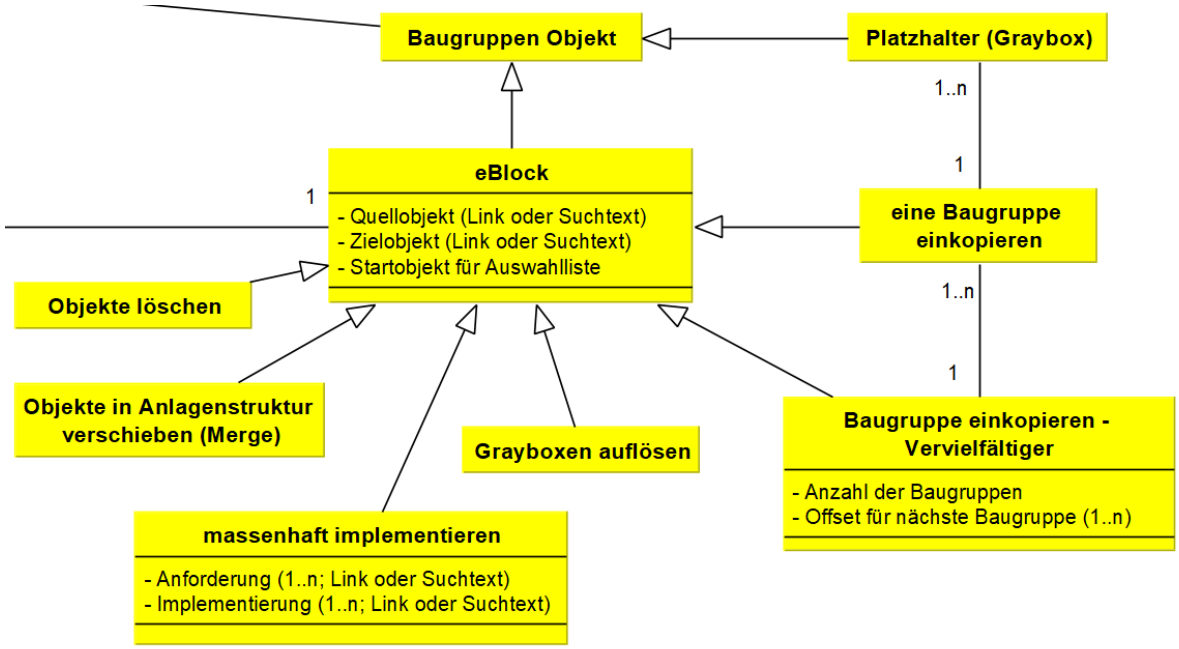


Abbildung 28: Baugruppen-Objekt, Quelle: Eigene Darstellung.

Eine Instanz der Klasse *eine Baugruppe einkopieren* hat im Wesentlichen zwei Funktionen. Erstens wird der Inhalt eines gesamten Dokumentes auf einem anderen Dokument platziert. Diese Funktion wird beispielsweise bei den P&IDs der Variations (Modul-P&IDs), auf denen alle für die Variation relevanten Objekte platziert, verwendet. Der Einfügepunkt am Zieldokument wird durch eine Instanz der Klasse *Platzhalter (Graybox)*, welche am Haupt-P&ID platziert ist, definiert. Die Instanz der Klasse *Platzhalter (Graybox)* wird dem Attribut *Quellobjekt* zugewiesen. Die Multiplizität *1..n* gibt an, dass mindestens 1 und maximal n Objekte der Klasse *Platzhalter (Graybox)* zugewiesen werden können. Diese Funktion wird benötigt, wenn Objekte auf mehreren Planarten, wie z.B. einem Stromlaufplan und einem Aufbauplan platziert werden sollen. Dem Modul-P&ID ist für gewöhnlich ein Objekt der Klasse *Pfadmakro* hierarchisch übergeordnet. Dieses wird mit dem Attribut *Zielobjekt* verlinkt. Zweitens kopiert das Objekt der Klasse *eine Baugruppe*

einkopieren die Objekte, welche auf dem Modul-P&ID platziert sind und üblicherweise eine Variante oder eine Komponente darstellen, von dem Template-Projekt in Comos in das gewünschte Projekt. Um die Objekte nun in die Anlagenstruktur zu verschieben, wird die Klasse *Objekte in Anlagenstruktur verschieben (Merge)* verwendet.

Soll nicht nur eine Baugruppe einkopiert, sondern wie z.B. bei Anforderung 14 der in Abbildung 23 dargestellten Anforderungsliste angeführt, mehrere Objekte desselben Templates angelegt werden, wird die Klasse *Baugruppe einkopieren – Vervielfältiger* benötigt. Diese besitzt zwei zusätzliche Attribute. Das Attribut *Anzahl der Baugruppen* definiert, wie oft eine bestimmte Baugruppe einkopiert und platziert werden soll. Das zweite neue Attribut *Offset für nächste Baugruppe (1..n),* legt den Offset für die Position des nächsten Objektes auf einem bestimmten Plan fest. Die Position des ersten Objektes wird wiederum durch eine Instanz der Klasse *Platzhalter (Graybox)* definiert. Im Comos werden diverse Objekte wie z.B. IO-Signale und Klemmen erst durch deren Implementierung vollständig beschrieben. Mit dieser wird beispielsweise dem IO-Signal ein Kanal eines IO-Modules zugewiesen. Im Rahmen der automatisierten Erstellung der Anlagendokumentation muss somit die Software auch diese Funktion beherrschen. Diese Aufgabe erledigt die Klasse *massenhaft implementieren*. Sie besitzt zusätzlich zwei Attribute, das Attribut *Anforderung (1..n)* und das Attribut *Implementierung (1..n).* Mit diesen zwei Attributen können eine oder mehrere Anforderungen und dieselbe Anzahl an Implementierungen verlinkt werden. Dies geschieht wiederum entweder durch direkte Zuweisung oder mittels Zuweisung durch Suchtexte. Wird eine Instanz der Klasse *massenhaft implementieren* ausgeführt, werden die verlinkten Implementierungen mit den verlinkten Anforderungen verknüpft.

Um die Anforderung *mit Software erstellte Anlagen müssen wie manuell erstellte Projekte im Comos modifizierbar sein* der Anforderungsliste erfüllen zu können, wird die Klasse *Grayboxen auflösen* verwendet. Durch Einkopieren der Baugruppen werden am Haupt-P&ID die Objekte der Modul-P&IDs nicht direkt platziert, sondern es wird durch die *Graybox* ein Zeiger an der richtigen Stelle erstellt. So sieht das Ergebnis am Ausdruck zwar richtig aus, jedoch können die einzelnen Objekte nicht direkt am Haupt-P&ID, sondern nur auf dem jeweiligen einkopierten Modul-P&ID bearbeitet werden. Eine Instanz der Klasse *Grayboxen auflösen* platziert die Objekte direkt am Haupt-P&ID und löscht die *Grayboxen*.

Die Entwicklung einer Software und somit auch die Entwicklung der Modelle ist ein iterativer Prozess. So enthielten beispielsweise die ersten Versionen des UML-Klassendiagrammes noch keine Klasse *e-Block*. All diese Funktionen sollten ursprünglich von der Klasse *Action*, die in Abbildung 29 dargestellt ist, abgedeckt werden. Erst im Zuge der Umsetzung der Entwicklung des *Metris Engineering Configurators* kam die Notwendigkeit einer Klasse *e-Block* auf.

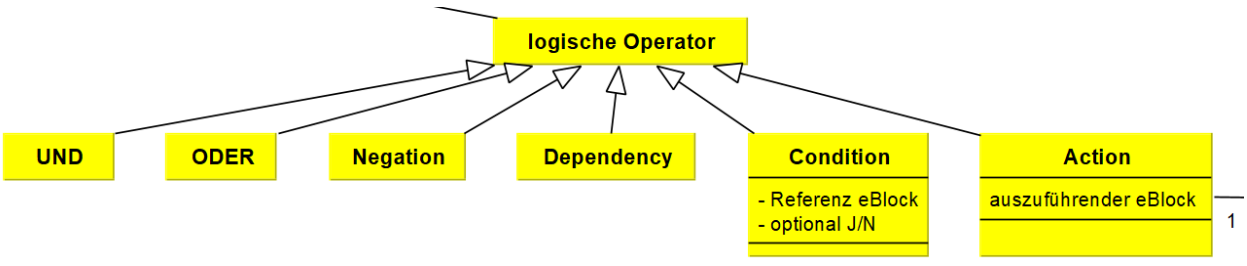


Abbildung 29: logische Operator, Quelle: Eigene Darstellung.

Nun besteht die einzige Funktion der Klasse *Action* darin, die spezialisierten Instanzen der Klasse *eBlock* aufzurufen. Aus diesem Grund besteht zwischen dieser Klasse und der Klasse *eBlock* eine Assoziation mit der Multiplizität von 1-1. Das bedeutet, dass genau einer Instanz der Klasse *Action* immer genau eine Instanz der Klasse *eBlock* oder einer Spezialisierung davon, zugewiesen werden muss. Dasselbe gilt auch für den umgekehrten Fall. Diese Assoziation wird in Abbildung 28 und Abbildung 29 illustriert. Die Klasse *Condition* dient als Basisobjekt für die grafische Benutzeroberfläche. Konkret bedeutet das, dass alle Instanzen der Klasse *Condition* auf der grafischen Benutzeroberfläche angezeigt werden. Eine Instanz der Klasse *Condition* besitzt zwei spezielle Attribute. Das erste Attribut *Referenz eBlock* dient der Anzeige der Auswahlmöglichkeiten einer Combobox. Ist die Instanz der Klasse *eine Baugruppe einkopieren* mit einem Variation Point verlinkt, werden alle hierarchisch untergeordneten Variations in der Combobox angezeigt. Zusätzlich muss diese Instanz der Klasse *eine Baugruppe einkopieren* eine Referenz zum Attribut *Referenz eBlock* der *Condition* besitzen. Mit dem zweiten speziellen Attribut *optional J/N* kann ein Comos Key-User auswählen, ob eine spezielle Instanz einer *Condition* für die Konfiguration einer Anlage optional oder obligatorisch ist. Wurde eine Instanz einer *Condition* als optional gekennzeichnet, erscheint bei der grafischen Benutzeroberfläche eine Checkbox, bei der die Option ausgewählt werden kann.

Es besteht die Möglichkeit, dass gewisse Teile einer Filterpressenanlage nur in Kombination mit anderen bestimmten Teilen sinnvoll eingesetzt werden können. Diese Beziehung wird als *requires*-Beziehung bezeichnet. Der Einsatz einer bestimmten Komponente kann eine andere Komponente nicht nur voraussetzen, sondern auch ausschließen. Eine *excludes*-Beziehung definiert diese Beziehung. Um beide Beziehungen im *Metris Engineering Configurator* umsetzen zu können, wird die Klasse *Dependency* eingeführt. Mit einem Objekt dieser Klasse können die *Conditions* am *Verknüpfungsplan*, welcher im nächsten Absatz erläutert wird, beliebig verschalten werden. Da eine Instanz der Klasse *Condition* oftmals nicht exakt einer Instanz der Klasse *Action* und somit einer Instanz der Klasse *eBlock* zugewiesen werden kann, werden die restlichen spezialisierten Klassen der Superklasse *logische Operatoren* eingeführt. Mit *UND* und *ODER*-Verknüpfungen können beliebige Kombinationen von Instanzen von *Conditions* und *Actions* definiert werden. Auch die *Negation*, mit welcher Zustände invertiert werden, ist ein wichtiger *logischer Operator*. Alle Instanzen der spezialisierten Klassen der *logischen Operatoren* können auf einem Verknüpfungsplan dargestellt und verschalten werden.

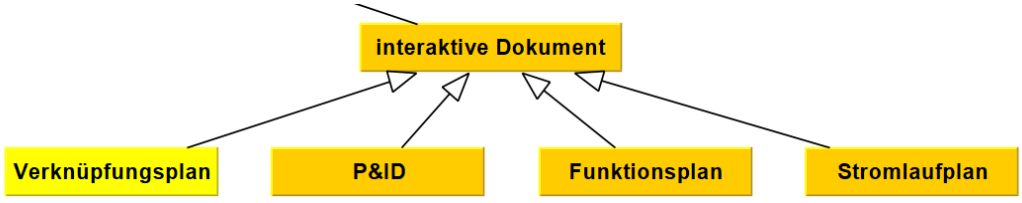


Abbildung 30: interaktive Dokument, Quelle: Eigene Darstellung.

In Abbildung 30 ist die Superklasse *interaktives Dokument* und Klassen, die mit dieser in einer Generalisierungsbeziehung stehen, dargestellt. Die einzige neue Klasse ist die Klasse *Verknüpfungsplan*. Mit Hilfe des *Verknüpfungsplans* können die im vorherigen Absatz beschriebenen logischen Verbindungen und Abhängigkeitsbeziehungen zwischen den *Conditions* und den *Actions* grafisch dargestellt werden. Weitere interaktive Dokumente, welche für die neue Software relevant sind, sind das P&ID, der *Funktionsplan* und der *Stromlaufplan*. Mit Hilfe eines *Funktionsplans* wird die Funktion einer Anlage definiert. Üblicherweise

dient ein *Funktionsplan* als Vorlage zur Erstellung der Automatisierungssoftware. Auf einem Stromlaufplan sind der Fluss des elektrischen Stromes und alle relevanten elektrischen Bauteile ersichtlich.

5.3.2 Funktionale Perspektive

In diesem Kapitel wird die Software aus der Funktionsperspektive mithilfe eines UML-Aktivitätsdiagrammes betrachtet. Wie im Use-Case-Diagramm im Unterkapitel 5.2.2 dargestellt, gibt es zwei wesentliche Unterscheidungen bei der Konfiguration einer Anlage mithilfe des *Metris Engineering Configurators*. Entweder kann ein *Comos-User* eine Anlage konfigurieren, oder die Konfiguration wird vom *System XML* durchgeführt. Aus diesem Grund wird auch die Beschreibung der Funktion des *Metris Engineering Configurators* in mehrere UML-Aktivitätsdiagramme aufgeteilt. Das erste Diagramm, welches in Abbildung 31 und Abbildung 32 zu sehen ist, beschreibt die Funktion der Software bei Anwendung durch einen *Comos-User*. Wird die Anlage konfiguriert ohne Comos zu öffnen, ist das zweite Diagramm, welches in Abbildung 33 dargestellt ist, gültig. In beiden Fällen wird die Aktivität *Objekte erstellen/ändern* aufgerufen. Diese wird in Abbildung 34 näher erläutert.

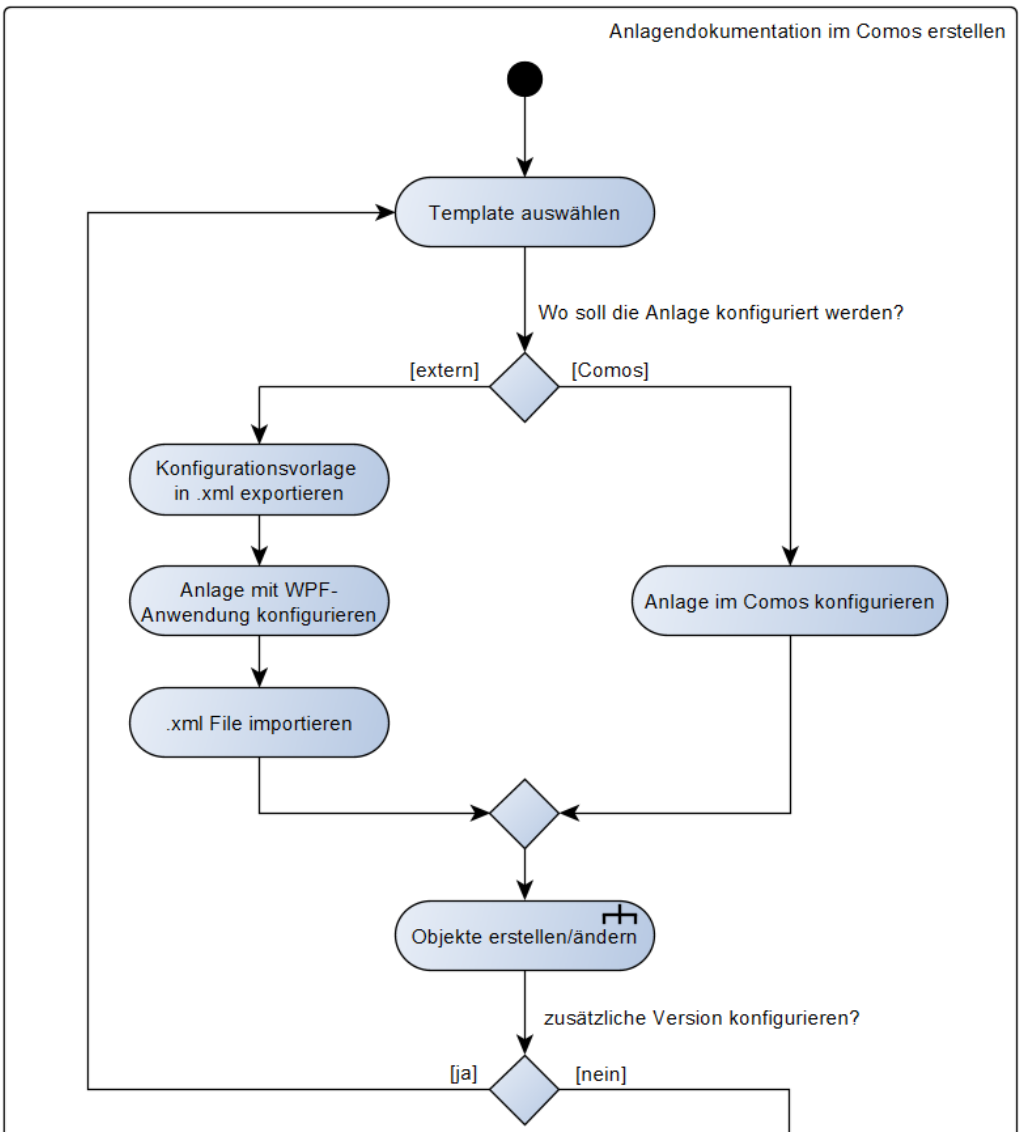


Abbildung 31: UML-Aktivitätsdiagramm *Anlagendokumentation im Comos erstellen* – Teil 1, Quelle: Eigene Darstellung.

Wird die Anlagendokumentation im Comos erstellt, muss als Erstes das Programm Comos geöffnet und die neue Software gestartet werden. Wie in Abbildung 31 ersichtlich, wird danach das gewünschte *Template ausgewählt*. Dieses kann im selben oder in einem anderen Comos-Projekt gespeichert sein. Wurde das Template ausgewählt, erscheinen die zu konfigurierenden Elemente auf der grafischen Benutzeroberfläche. Wie in obiger Abbildung zu sehen, kann die Anlage nun direkt im Comos oder extern konfiguriert werden. Wird der zweite Weg gewählt, muss die *Konfigurationsvorlage* zuerst in Form eines XML-Files *exportiert*, die *Anlage mit* der *WPF-Anwendung konfiguriert* und das *XML-File* danach wieder *importiert* werden. Bei Konfiguration der Anlage im Comos, wird nur die Aktivität *Anlage im Comos konfigurieren* benötigt. In beiden Fällen ruft die nächste Aktivität eine weitere Aktivität, welche in Abbildung 34 beschrieben wird, auf. Wurden die Objekte erfolgreich erstellt oder geändert, führt der Prozess zu einem weiteren Entscheidungsknoten. Soll eine *zusätzliche Version* (entspricht der Anforderung 13 der im Unterkapitel 5.2.1 beschriebenen Anforderungsliste) der Anlage erstellt werden, startet der beschriebene Prozess von vorne mit der *Auswahl der Templates*. In Abbildung 32 ist ersichtlich, dass eine Verneinung der Frage des beschriebenen Entscheidungsknotens zu einem weiteren Entscheidungsknoten führt.

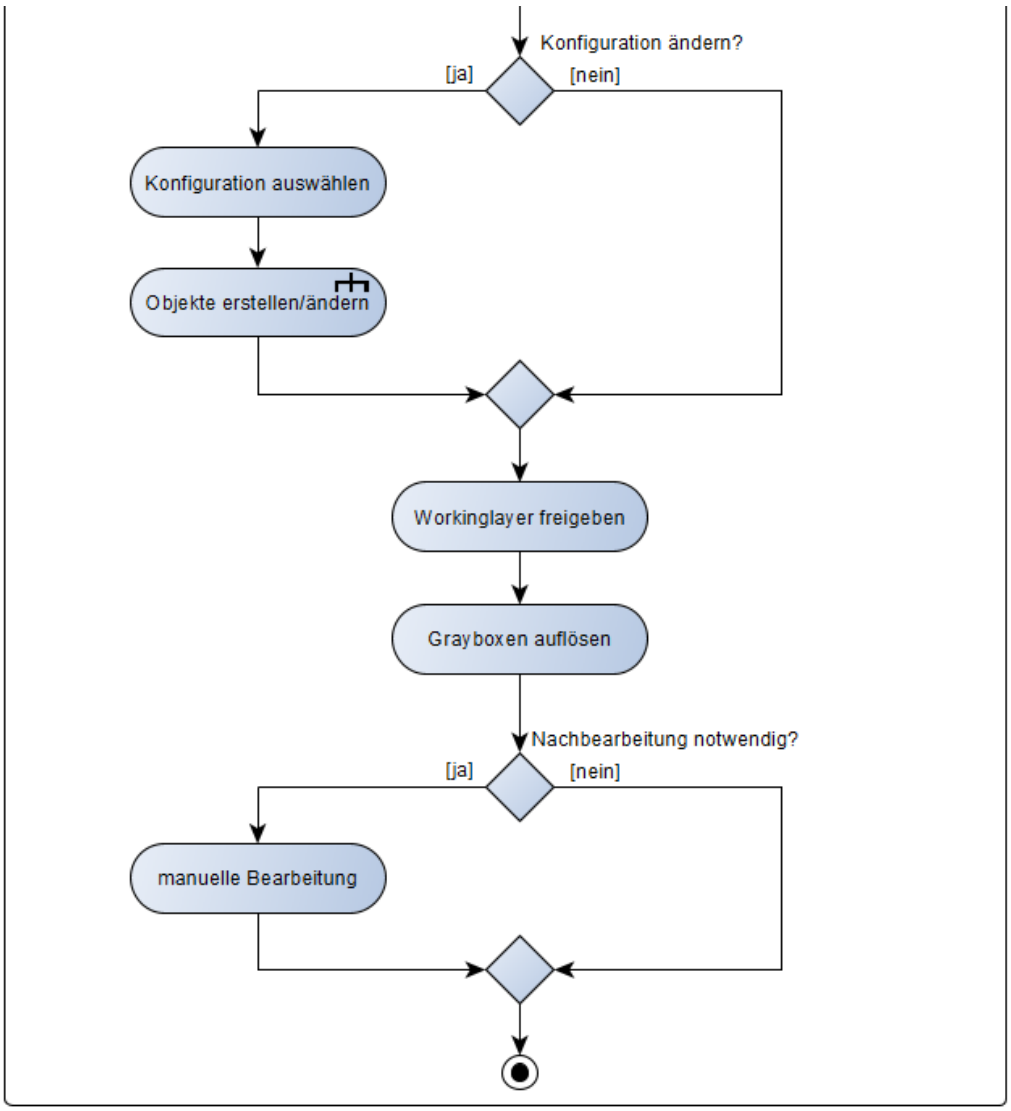


Abbildung 32: UML-Aktivitätsdiagramm *Anlagendokumentation im Comos erstellen* – Teil 2, Quelle: Eigene Darstellung.

Soll eine bestehende Konfiguration überarbeitet werden, wird die Frage *Konfiguration ändern?* mit *ja* beantwortet und der linke Zweig des UML-Aktivitätsdiagrammes, welcher in Abbildung 32 dargestellt wird, gewählt. Zuerst muss die *Konfiguration*, welche geändert werden soll, *ausgewählt werden*, danach wird wiederum die Aktivität *Objekte ändern/erstellen* aufgerufen. Wird die genannte Frage mit *nein* beantwortet, werden diese zwei Aktivitäten übersprungen. Wurden mehrere Versionen einer Anlage erstellt, wird mit der Aktivität *Workinglayer freigeben*, eine Version für den weiteren Prozess freigegeben. Für die Erfüllung der Anforderung 11 (*mit Software erstellte Anlagen müssen wie manuell erstellte Projekte im Comos modifizierbar sein*) der im Unterkapitel 5.2.1 beschriebenen Anforderungsliste, ist die Aktivität *Grayboxen auflösen* zuständig. Im Zuge dieser Aktivität werden die mit Hilfe der Grayboxen dargestellten Objekte am Plan direkt platziert und die Grayboxen aufgelöst. Dieser Vorgang ist irreversibel. Bevor der Vorgang der Erstellung einer Anlagendokumentation im Comos abgeschlossen wird, besteht die Möglichkeit die erstellten Dokumente manuell nachzubearbeiten.

Das zweite UML-Aktivitätsdiagramm, welches in Abbildung 33 illustriert wird, bezieht sich auf die Anforderung 5 (*Anlagendokumentation soll erstellt werden können ohne Comos zu öffnen*) der Anforderungsliste. Der Hintergrund dieses Anwendungsfalles besteht darin, dass der Verkäufer/die Verkäuferin üblicherweise nicht das Engineering Programm Comos nutzen. Stattdessen können sie mittels der *WPF-Anwendung* eine *Anlage konfigurieren*. Damit das möglich ist, muss ein Comos-User das *Template auswählen* und die *Konfigurationsvorlage* in Form eines *XML-Files exportieren.* Im Gegensatz zum in Abbildung 31 dargestellten Aktivitätsdiagramm wird das konfigurierte File nicht importiert, sondern beim *Enterprise Server abgelegt*. Das bedeutet, dass das XML-File in einem bestimmten Ordner im ANDRITZ-Netzwerk gespeichert wird. Daraufhin startet das System *Enterprise Server* die Aktivität *Objekte erstellen/ändern*. Ist der Vorgang abgeschlossen, *startet der Enterprise Server den Druckprozess für* die definierten *Dokumente*. Im Gegensatz zur *Erstellung der Anlagendokumentation im Comos* ergeben sich bei diesem Anwendungsfall einige Einschränkungen. Wurde bereits eine Anlage konfiguriert, wird automatisch eine neue Version erstellt. Weiters können Änderungen und manuelle Nachbearbeitungen nur direkt im Comos ausgeführt werden.

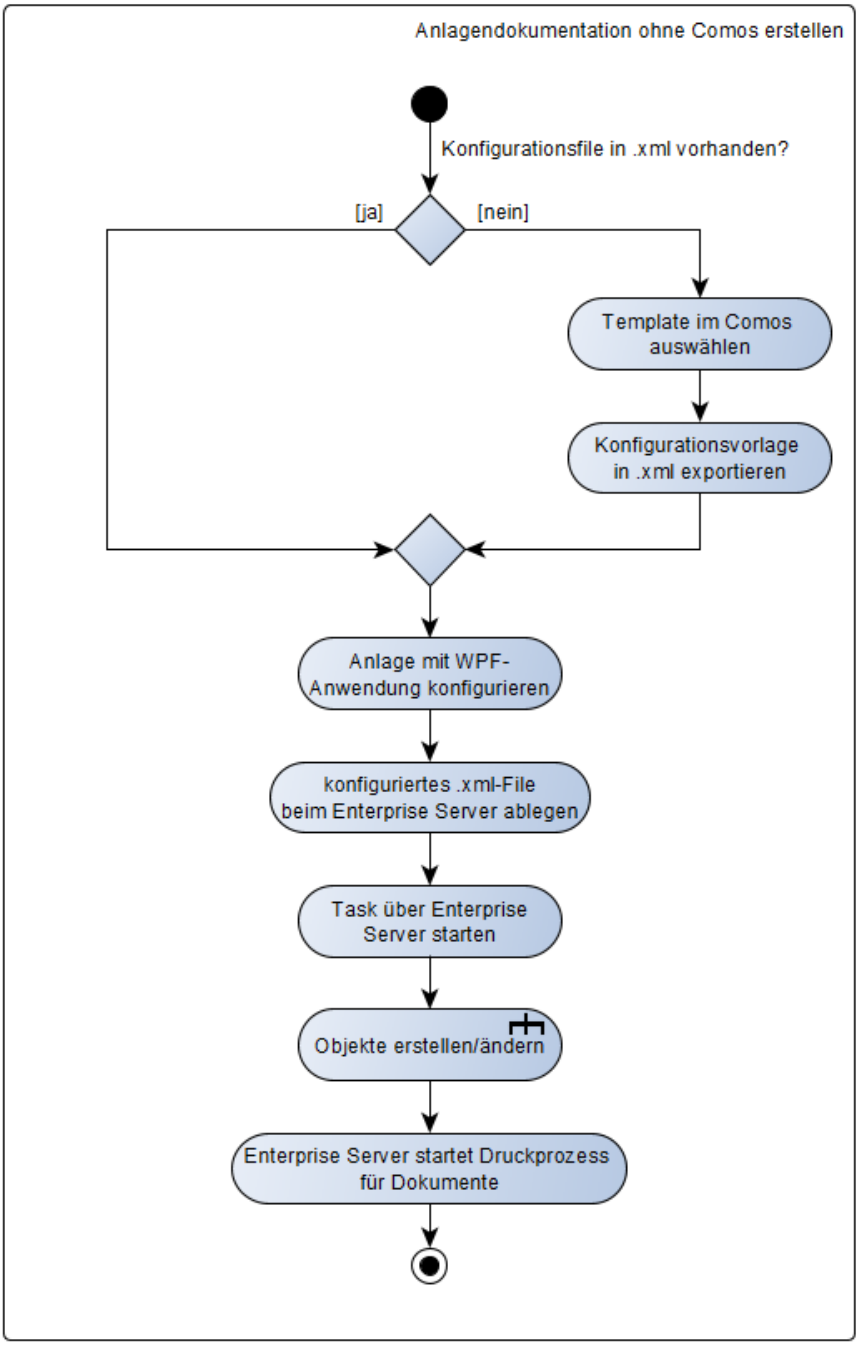


Abbildung 33: UML-Aktivitätsdiagramm *Anlagendokumentation ohne Comos erstellen*, Quelle: Eigene Darstellung.

In Abbildung 34 ist das UML-Aktivitätsdiagramm für die Funktion *Objekte erstellen/ändern*, welche durch die in diesem Kapitel beschriebenen UML-Aktivitätsdiagramme aufgerufen wird, dargestellt. Diese Funktion wird sowohl bei Erstellung einer neuen Anlage oder neuen Version einer Anlage, wie auch bei Änderung einer Anlagenkonfiguration ausgeführt. Als Erstes werden in diesem Prozess *die Attribute* der Objekte *der* Klasse *Conditions* mit Werten befüllt. Diese Parameter werden entweder durch die Konfiguration der Anlage im Comos oder durch den Import eines XML-Files befüllt. Durch die Verlinkung der Instanz der Klasse *Condition* mit einem Objekt der Klasse *eBlock* können die Parameter übergeben werden. Der *eBlock* benötigt beispielsweise diese Werte, um entscheiden zu können, welche Variation einkopiert werden soll.

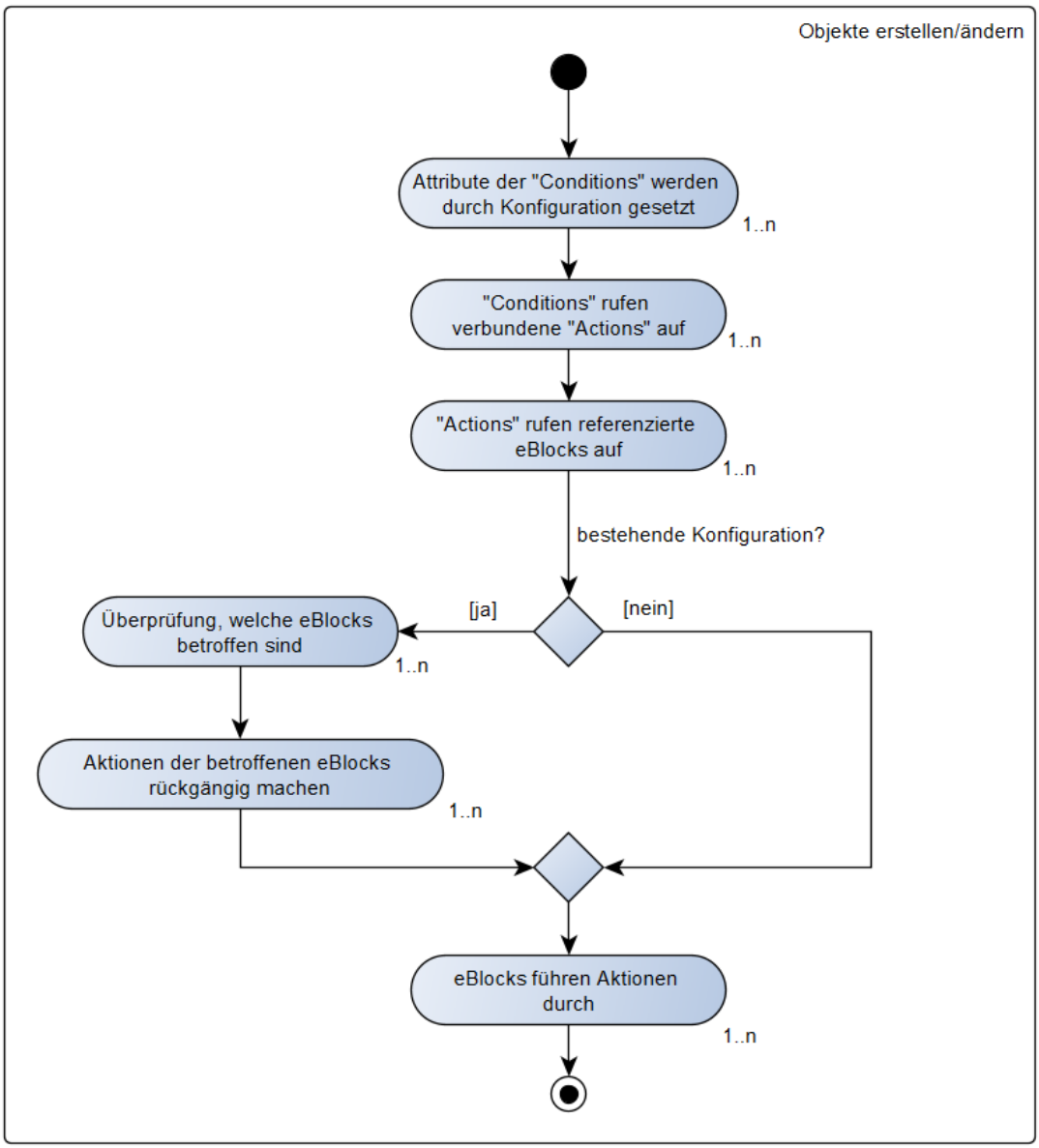


Abbildung 34: UML-Aktivitätsdiagramm *Objekte erstellen/ändern*, Quelle: Eigene Darstellung.

Bei der zweiten Aktivität im oben beschriebenen Diagramm, *rufen* die Objekte der Klasse *Conditions* die referenzierten Objekte der Klasse *Actions auf*. Diese Verlinkungen werden üblicherweise mit Hilfe des Verknüpfungsplans erstellt. Die Objekte der Klasse *Actions* rufen wiederum die verlinkten Objekte der Klasse *eBlocks auf*. Handelt es sich um eine bestehende Konfiguration, welche geändert werden soll, überprüft der jeweilige *eBlock*, ob er von dieser Änderung betroffen ist. Ist das der Fall, werden die Aktionen, die von diesem bei Erstellung der Ursprungskonfiguration ausgeführt wurden, rückabgewickelt. Danach führt das Objekt der Klasse *eBlock* die jeweilige Aktion aus. Diese kann z.B. aus Einkopieren von Objekten oder Implementieren von Klemmen bestehen. Handelt es sich um eine Konfiguration einer neuen Anlage oder neue Version einer bestehenden Anlage, entfallen die Aktionen für die Überprüfung der Betroffenheit bzw. die Rückabwicklung der Aktionen. Der jeweilige *eBlock* führt die definierte Aktion sofort aus. Da die Konfiguration einer Anlage üblicherweise mehrere Conditions und Actions beinhaltet, werden die Aktivitäten mehrmals ausgeführt. Das sagt die Multiplizität *1..n* bei der jeweiligen Aktivität aus.

5.3.3 Dynamische Verhalten der Software

Die dritte Perspektive, aus der die Anforderungen an den *Metris Engineering Configurator* beschrieben werden, ist das dynamische Verhalten. Zu diesem Zweck wurde das UML-Zustandsdiagramm erstellt. Wie der Name bereits sagt, beschreibt es die Zustände, in denen sich die Software während des Konfigurationsprozesses einer neuen Anlage befinden kann.

Der erste Zustand *Auswahl Template*, welcher in Abbildung 35 illustriert wird, deckt sich mit der ersten Aktion des in Abbildung 31 beschriebenen UML-Aktivitätsdiagrammes. Wurde das Template ausgewählt, befindet sich die Software im Zustand *Konfiguration Anlage*. Hier bieten sich dem Comos User drei Möglichkeiten. Erstens kann er die Anlage im Comos konfigurieren und mit der *Schaltfläche Execute* die Software in den nächsten Zustand *Objekte anlegen/ändern* überführen. In diesem Zustand wird die Aktivität *Objekte anlegen/ändern*, welche im vorhergehenden Unterkapitel beschrieben wird, ausgeführt. Bei der zweiten Möglichkeit zur Konfiguration einer Anlage exportiert ein Comos User die Konfigurationsvorlage in Form eines XML-Files. Nach erfolgter *Konfiguration* der *Anlage* mittels der *WPF-Anwendung* wird dieses wieder importiert und mit der *Schaltfläche Execute* wird wiederum der Zustand geändert.

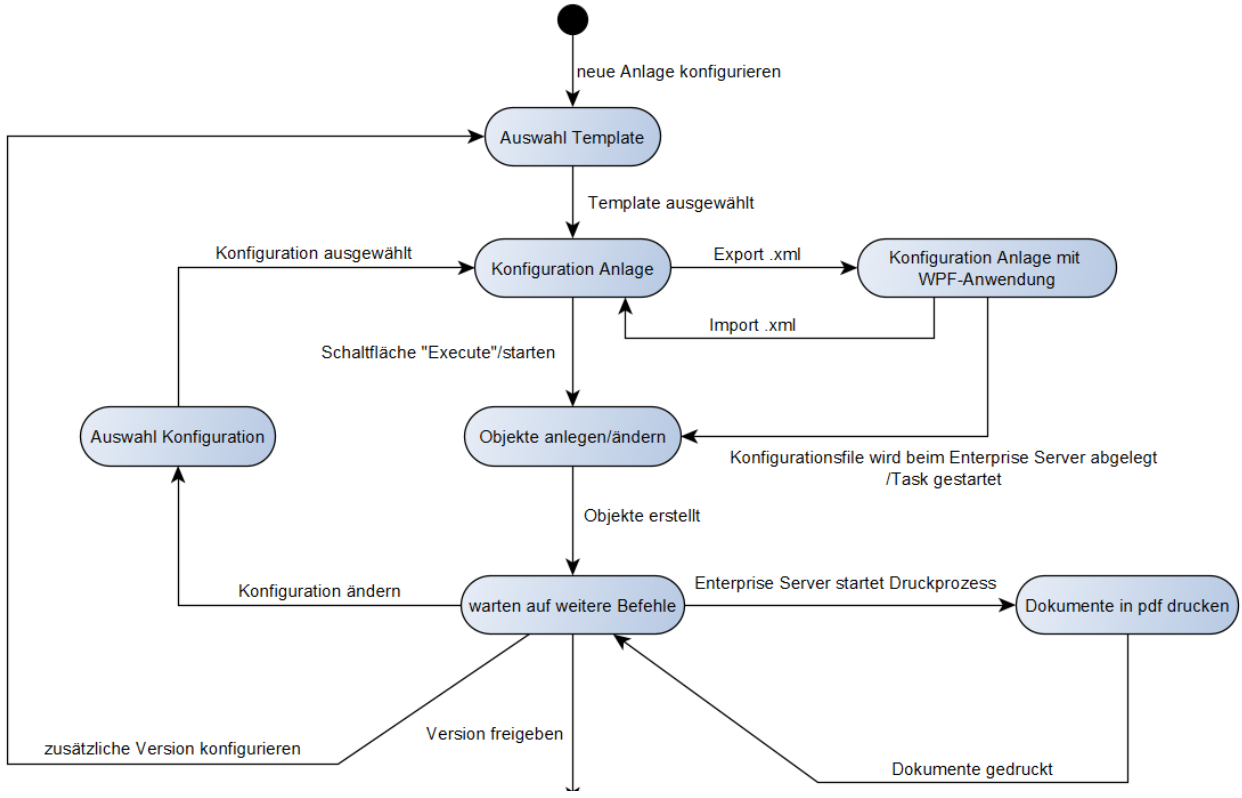


Abbildung 35: UML-Zustandsdiagramm – Teil 1, Quelle: Eigene Darstellung.

Die dritte Möglichkeit sieht anstatt des Imports des Konfigurationsfiles, die *Ablage des Konfigurationsfiles am Enterprise Server* vor. Dieser ändert den Zustand der Software zu *Objekte anlegen/ändern*. Die Funktionsweise des Enterprise Servers wird im Unterkapitel 5.3.2 detailliert erläutert. Wurden alle *Objekte erstellt*, geht die Software in einen Wartezustand über. Befindet sich die Software in diesem Zustand, bestehen mehrere Wahlmöglichkeiten zur weiteren Vorgangsweise. Möchte der Comos User eine bestehende *Konfiguration ändern*, muss die zu ändernde Konfiguration ausgewählt werden. Ist das geschehen, befindet sich die Software ein weiteres Mal in dem Zustand *Konfiguration Anlage*. Möchte ein Comos User eine

zusätzliche Konfigurationsvariante erstellen, befindet sich die Software wieder im Zustand *Auswahl Template*. Neben dem Starten des Objekterstellungsprozesses, kann der *Enterprise Server* auch *den Druckprozess starten*. Ist diese Aktion abgeschlossen, befindet sich die Software im selben Zustand wie davor.

Die unterschiedlichen Versionen einer Anlage werden dem Kunden üblicherweise im Verkaufsprozess vorgelegt. Entscheidet sich dieser für eine bestimmte Variante, wird diese in der Software freigegeben. Der Zustand dafür wird in Abbildung 36 als *Workinglayer freigeben* beschrieben. Nach diesem Schritt werden die *Grayboxen aufgelöst* und die Anlage für die *manuelle Nachbearbeitung* freigegeben. Nach der manuellen Nachbearbeitung ist die *Konfiguration abgeschlossen* und der Prozess beendet.

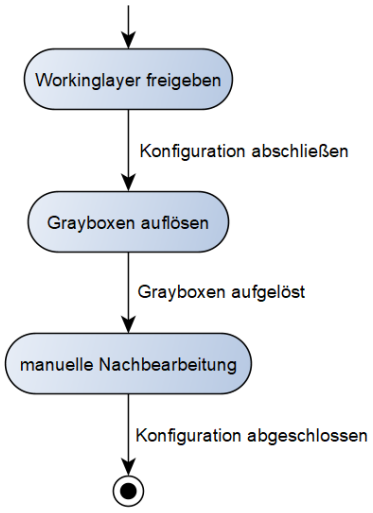


Abbildung 36: UML-Zustandsdiagramm – Teil 2, Quelle: Eigene Darstellung.

5.3.4 Präsentationsmodell

Die Modelle, welche in diesem Unterkapitel 5.3 bis jetzt behandelt werden, beziehen sich allesamt auf die Funktionalitäten einer Software und dessen Struktur. Einen weiteren wichtigen Aspekt einer erfolgreichen Software stellt die Bedienoberfläche dar. Warum es wichtig ist, auf die Bedienoberfläche Wert zu legen und die theoretischen Grundlagen zum Präsentationsmodell können im Unterkapitel 3.4.4.5 nachgeschlagen werden.

Das in Abbildung 37 dargestellte Präsentationsmodell gibt die Anforderungen an das Hauptfenster der grafischen Benutzeroberfläche, auch GUI (Graphical User Interface) genannt, des *Metris Engineering Configurators* vor. Dieses Modell wurde mit dem Programm *Pencil* im Style *Desktop – Sketchy GUI* erstellt. Wie hier zu sehen ist, ist diese Darstellung an kein Betriebssystem angelehnt, sondern erinnert an eine Handskizze. Die Vorteile dessen werden ebenfalls im Unterkapitel 3.4.4.5 erläutert. Da Benutzeroberflächen so einfach wie möglich gehalten werden sollen, kann die neue Software mit einem Hauptfenster und einem Popup vollständig bedient werden.

Das Hauptfenster wird in drei Bereiche unterteilt. Im oberen Bereich befinden sich die generellen Schaltflächen. Durch Klicken auf die Schaltfläche *select Template* können die Templates bzw. die Konfigurationen ausgewählt werden. Bei Auswahl eines Templates wird eine neue Konfiguration erstellt. Wird eine bestehende Konfiguration ausgewählt, kann diese geändert werden. *(1)* in Abbildung 37 zeigt an, dass damit das Popup mit der Nummer eins aufgerufen wird. Wurde das Template oder die jeweilige Konfiguration

ausgewählt, erscheinen abhängig vom Template im linken unteren Drittel des Hauptfensters unterschiedliche Objekte. Grundsätzlich wird zwischen obligatorischen und optionalen Objekten unterschieden. Wie in Abbildung 37 ersichtlich, wird bei optionalen Objekten, wie der *Option 1* zusätzlich eine Checkbox dargestellt. Das Feld *ausgewählte Variante* ist ebenfalls von dem jeweiligen Objekt abhängig. Beispielsweise besitzt die *Option 1* keine unterschiedlichen Varianten. Aus diesem Grund ist das Feld *ausgewählte Variante* bei dieser Komponente nicht sichtbar. Wird ein Anlagenteil markiert, erscheinen je nach Auswahlmöglichkeit der Varianten die Felder im rechten unteren Drittel des Hauptfensters. Hier können die unterschiedlichen Varianten der einzelnen Objekte für die zu erstellenden Anlagenteile ausgewählt werden. Soll eine Anlage nicht direkt im Comos, sondern über die XML-Schnittstelle konfiguriert werden, kann mit den Schaltflächen *Import* und *Export* das Konfigurationsfile zuerst in ein XML-File exportiert und nach erfolgter Konfiguration wieder importiert werden. Beim Import eines Konfigurations-Files werden die im XML-File befüllten Werte des Feldes *ausgewählte Variante* und die Checkboxen für die optionalen Anlagenteile übernommen. Durch Klicken der Schaltfläche *Apply* kann der Erstellungsprozess der Objekte und Dokumente gestartet werden. Wird auf *Close* geklickt, wird die Software ohne weitere Aktionen durchzuführen, geschlossen. Die Funktion der Schaltfläche *Execute* ist eine Kombination der Schaltflächen *Apply* und *Close*. Bei Klicken auf die Schaltfläche *Execute* wird zuerst der Erstellungsprozess gestartet und danach das Fenster geschlossen.

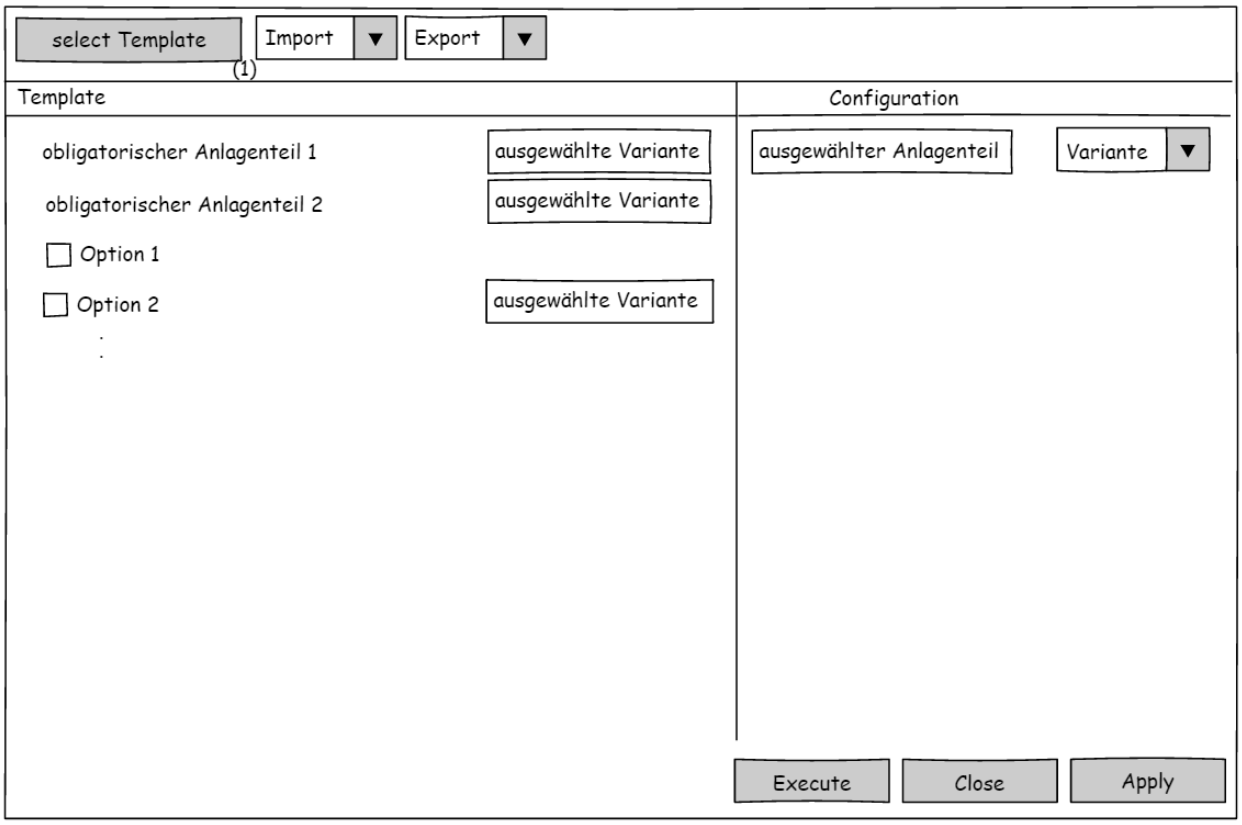


Abbildung 37: Präsentationsmodell – Hauptfenster, Quelle: Eigene Darstellung.

In Abbildung 38 ist das Popup ersichtlich, das erscheint, wenn die Schaltfläche *select Template* im Hauptfenster angeklickt wird. Mit diesem kann das Template für eine Konfiguration oder eine bestehende Konfiguration ausgewählt werden. Im linken Teil des Fensters wird das Comos Projekt, in dem das gewünschte Template bzw. die Konfiguration gespeichert ist, ausgewählt. In der rechten Hälfte erscheinen nach

Auswahl des Projektes die unterschiedlichen Templates und Konfigurationen. Durch Markieren des Templates bzw. der Konfiguration und Klicken auf die Schaltfläche *Select* wird das jeweilige Template oder die jeweilige Konfiguration übernommen und das Popup geschlossen. Wird statt *Select* die Schaltfläche *Cancel* benutzt, schließt sich das Popup ebenfalls, jedoch ohne die Auswahl zu übernehmen. Im Zuge dieser Masterarbeit wird ein Template für die Konfiguration einer Filterpressenanlage erstellt.

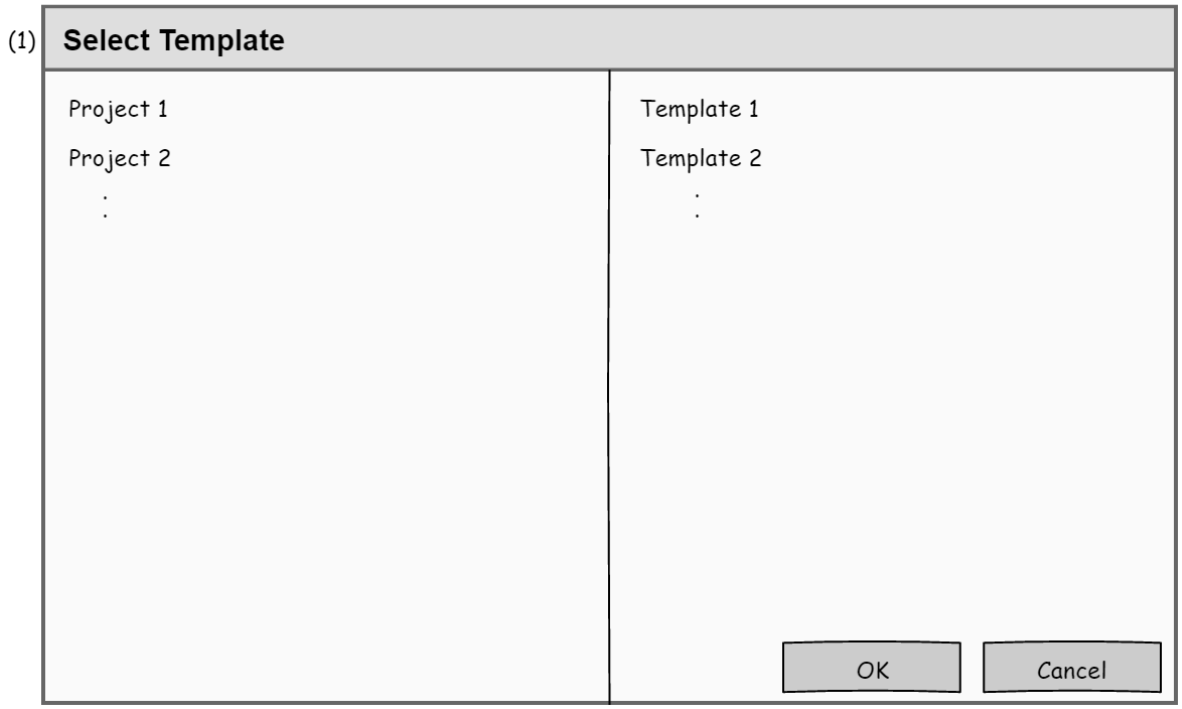


Abbildung 38: Präsentationsmodell – Popup Template auswählen, Quelle: Eigene Darstellung.

Soll die Anlage anstatt im Comos mittels der XML-Schnittstelle konfiguriert werden, werden die jeweiligen Varianten und Optionen nicht im Hauptfenster, sondern in einer eigenen WPF-Anwendung, die das XML-File grafisch darstellt, ausgewählt. Aufgrund der Durchgängigkeit ist für die GUI dieser Anwendung grundsätzlich das in Abbildung 37 beschriebene Hauptfenster ebenfalls gültig. Die einzigen Unterschiede betreffen die Schaltflächen im oberen und unteren Bereich. Die Schaltflächen *Import* und *Export* werden in der WPF-Anwendung nicht benötigt. Mit der Schaltfläche *select Template* erscheint ein Fenster, in dem das XML-File ausgewählt werden kann. Die Schaltfläche *Execute* fällt ebenfalls weg.

5.4 Anforderungen prüfen

Wie im Unterkapitel 3.5 - *Anforderungen prüfen* beschrieben, ist die konstruktive Qualitätssicherung der einfachste und kostengünstigste Weg, um Stakeholderlisten, Ziele und Anforderungen in guter Qualität zu erhalten. Aus diesem Grund wurde bereits bei Erstellung der Stakeholderanalyse, der Festlegung der Ziele und der Dokumentation der Anforderungen auf Qualitätskriterien Wert gelegt. Der zweite Schritt der Prüfung erfolgte nach Fertigstellung der Dokumentation der Anforderungen, aber vor Realisierung der neuen Software. Im Zuge dieser Prüfung wurde die Stakeholderliste mit den Anforderungen vom Unterkapitel 3.2.1 abgeglichen. Des Weiteren wurden die Ziele, welche im Projektauftrag dokumentiert sind, mit den Qualitätskriterien vom Unterkapitel 3.2.2 validiert. Ob die detaillierten Anforderungen den in Unterkapitel

3.1.4 angeführten Qualitätskriterien entsprechen, wurde ebenfalls überprüft. Diese Prüfungen wurden in Zusammenarbeit mit allen relevanten Stakeholdern durchgeführt.

Zusätzlich zu den beschriebenen Prüfungsschritten wurden im Rahmen der Erstellung der Modelle mehrere Testiterationen durchgeführt. Der Hauptgrund für diese Prüfungen ist die durchgängige Konsistenz und Vollständigkeit von der Stakeholderliste bis zu den einzelnen Modellen. So wurde beispielsweise nach Erstellung des UML-Aktivitätsdiagrammes das Use-Case-Diagramm geändert. Davor konnte auch das System Excel Konfigurationen ändern. Im Zuge der Erstellung des UML-Aktivitätsdiagrammes kam das Problem der technischen Realisierung dieser Anforderung auf. Aus diesem Grund wurde das Use-Case-Diagramm nach Absprache mit den Stakeholdern angepasst. Eine weitere Änderung betraf sowohl die Anforderungsliste, wie auch alle Modelle. Aufgrund der steigenden Komplexität der grafischen Benutzeroberfläche wurde das Excel-Interface durch ein XML-Interface ersetzt. Im Gegensatz zum Excel ist eine .NET basierte WPF-Anwendung objektorientiert aufgebaut. Dieser Umstand erleichtert die Erstellung einer grafischen Benutzeroberfläche wesentlich. Zumindest nach der Fertigstellung eines jeden Modells wurde eine Testiteration durchgeführt, bei der die neu erstellten Elemente gegen die bereits vorhandene Dokumentation validiert wurden.

6 MODULARISIERUNG DER FILTERPRESSENANLAGE

Um die neue Software im realen Projektumfeld testen zu können, wird ein konkreter Anwendungsfall betrachtet. Die Filterpressenanlage wurde aus zwei Hauptgründen ausgewählt. Erstens hat eine Filterpressenanlage im Gegensatz zu anderen von ANDRITZ verkauften Anlagen, wie z.B. einer Anlage zur Papierherstellung, für diese Anforderung den optimalen Umfang. Zweitens muss, um den *Metris Engineering Configurator* betreiben zu können, die Anlage konfigurierbar sein, das bedeutet, dass die möglichen Varianten einer Anlage eingeschränkt werden müssen. Auch dieser Umstand ist bei einer Filterpressenanlage gegeben. In diesem Kapitel wird zuerst die generelle Funktion einer Filterpressenanlage beschrieben. Danach wird auf die Modellierung dieser, inklusive der Beschreibung der verwendeten Modellierungssprache, eingegangen. Im letzten Unterkapitel wird die Erstellung der Templates für die Verwendung durch den *Metris Engineering Configurator* im Engineering Programm Comos detailliert erläutert.

6.1 Funktionalität der Filterpressenanlage

Eine Filterpressenanlage dient zur mechanischen Trennung von Fest- und Flüssigstoffen, üblicherweise in diskontinuierlicher Betriebsweise. Eine Filterpresse arbeitet nach der Filtrationsmethode. Das bedeutet, dass die Fest- von den Flüssigstoffen durch die Wirkung von Trennkräften mittels eines flüssigkeitsdurchlässigen Filtermediums getrennt werden. Die von den Feststoffen befreite Flüssigkeit wird als Filtrat und die auf dem Filtermedium zurückbleibenden Feststoffe, sofern in großer Menge vorhanden, als Filterkuchen bezeichnet. [107]

Vorläufer der heutigen Filterpressen wurden vor allem zur Gewinnung von Ölen und Säften aus pflanzlichen Rohstoffen eingesetzt. Mitte des zwanzigsten Jahrhunderts begann die industrielle Fertigung von Filterpressen, was die weite Verbreitung dieser Filtrationsgeräte nach sich zog. Durch die Erfindung von anderen Trenngeräten wie z.B. Anschwemmfilter, Vakuumdrehfilter oder Dekanter wurde die Filterpresse, aufgrund des hohen Personalaufwands, zurückgedrängt. Durch die Automatisierung der Filterpresse wurde der Personalaufwand stark reduziert und somit werden sie bis heute für verschiedenste Anwendungen eingesetzt. Die von ANDRITZ verkauften Filterpressenanlagen kommen in der Bergbau-, Lebensmittel-, Umwelt- oder Chemieindustrie zum Einsatz. [108] Dabei müssen Teilchen von 0,1 mm (z.B. Pigmente) bis hinauf in den mm-Bereich (z.B. polymergeflockter Klärschlamm) abgetrennt werden. [109]

6.1.1 Pressentypen

Filterpressen sind Druckfilter, bei denen ein durch Filterplatten und Rahmenplatten oder Kammerplatten gebildetes Filterpaket in einem Gestell zwischen einem festen und einem beweglichen Deckel angeordnet ist. Die Unterschiede zwischen den Filter-, Rahmen- und Kammerplatten werden im dritten Absatz dieses Unterkapitels beschrieben. Die Platten bestehen üblicherweise aus Polymeren oder Stahl mit

[107] Vgl. Gasper/Oechsle/Pongratz (Hrsg.) (2000), S. 1.

[108] Vgl. ANDRITZ (2018b), Online-Quelle [27.November.2018], S. 4.

[109] Vgl. Gasper/Oechsle/Pongratz (Hrsg.) (2000), S. 119.

Polymerbeschichtung. [110] Der feste Deckel ist über Holme mit der Brücke verbunden. Gemeinsam bilden sie das Pressengestell. Werden die Filterpressen nach deren Aufbau kategorisiert, wird daher von Seitenholm- und Brückenholmfilterpressen gesprochen. Wie in Abbildung 39 veranschaulicht, liegen bei der Seitenholmfilterpresse die Filter-, Rahmen- oder Kammerplatten auf den seitlich angebrachten Holmen auf. Somit ist der Raum über den Platten für diverse Vorrichtungen, wie z.B. Abstreifer frei. Ein weiterer Vorteil gegenüber der Brückenholmfilterpresse ist die geringere Masse. Im Gegensatz zur Seitenholmfilterpresse werden bei der Brückenholmfilterpresse die Platten von dem Brückenholm, der über den Platten angebracht ist, getragen. Sie werden vor allem für aggressive Medien und hohe Durchsatzraten eingesetzt. Der uneingeschränkte seitliche Zugang ist ein wesentlicher Vorteil gegenüber der Seitenholmfilterpresse. [111]

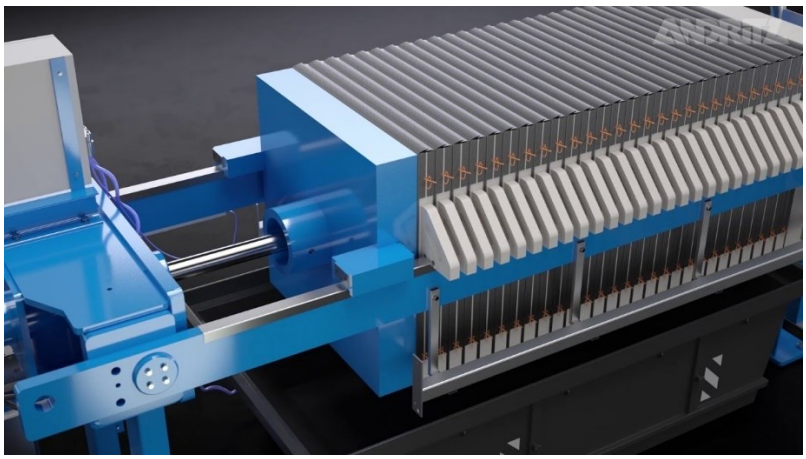


Abbildung 39: Seitenholmfilterpresse, Quelle: ANDRITZ (2017b), Online-Quelle [27.November.2018].

Ein weiteres wichtiges Bauteil beider Filterpressenarten ist die Anpressvorrichtung, welche in obiger Abbildung illustriert ist. Diese presst das Filterpaket beim Start des Filtrationsvorganges zusammen. Die Verrohrung für die Zu- und Ableitung der Produktströme ist im festen Deckel integriert. Zwischen den einzelnen Platten sind Gewebe oder Vliese angeordnet, die unter dem angebrachten Druck, neben der Filtration auch Dichtfunktion nach außen besitzen. In die von den Filterelementen gebildeten Hohlräume wird die Suspension mit Pumpen gefördert. Die Suspension passiert das Filtermedium und verlässt die Filterpresse über Ablaufkanäle als Filtrat. In den Kammern der Filterpresse bildet sich aus den Feststoffen ein Filterkuchen, der bei Bedarf gewaschen oder trocken geblasen werden kann. Nach Abschluss des Filtervorganges wird der Filterkuchen durch Öffnen der Filterpresse ausgetragen. Die Bezeichnung Filterpresse bezieht sich auf das Zusammenpressen der Filterplatten, das jedoch nicht für den eigentlichen Filtervorgang verantwortlich ist. Der benötigte Pressdruck entsteht nach Aufbau des Filterkuchens durch den Eintrag weiteren Materials. [112]

[110] Vgl. Tarleton/Wakeman (2007), S. 42.

[111] Vgl. ANDRITZ (2018b), Online-Quelle [27.November.2018], S. 6 ff.

[112] Vgl. Gasper/Oechsle/Pongratz (Hrsg.) (2000), S. 121.

Die zweite Möglichkeit der Kategorisierung der Filterpressen richtet sich nach der Ausführung des Plattenpakets. Hier wird zwischen Rahmen-, Kammer- und Membranfilterpresse unterschieden: [113]

Rahmenfilterpresse:

Die Rahmenfilterpresse ist die älteste und einfachste Form der Filterpresse. Wie in Abbildung 40 ersichtlich, besteht sie aus meist quadratischen Filter- und Rahmenplatten, welche alternierend angeordnet sind. Der Zulauf erfolgt über horizontale Kanäle im Dichtrand und tritt über vertikale Bohrungen oder Schlitze in die Hohlräume zwischen den Platten ein (dunkelblau dargestellt). Die Suspension passiert das Filtermedium, wird über Noppen oder Kanäle auf der Oberfläche der Platte abgeleitet und tritt im unteren Dichtrand wieder aus (hellblau dargestellt).

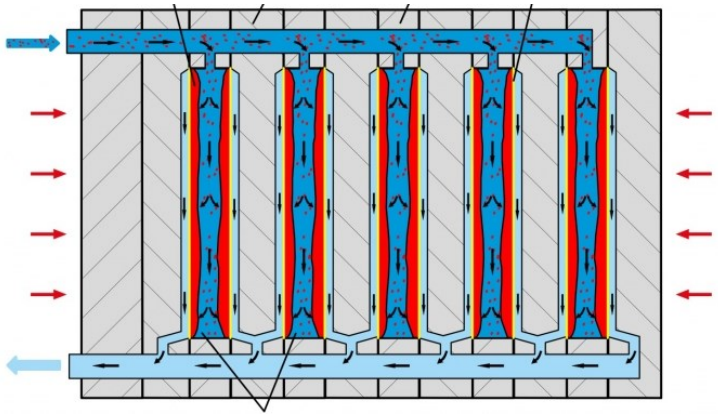


Abbildung 40: Rahmenfilterpresse, Quelle: G.U.N.T. Gerätebau (2018), Online-Quelle [27.November.2018].

Die Vorteile der Rahmenfilterpresse gegenüber der Kammer- und der Membranfilterpresse ergeben sich hauptsächlich durch die einfache konstruktive Gestaltung. So kann die Dicke der Kammer und somit auch die Kuchendicke allein durch den Austausch der Rahmenplatten einfach und kostengünstig geändert werden. Des Weiteren stellt die Vielfältigkeit der anwendbaren Filtermaterialien und deren einfacher Austausch einen wesentlichen Vorteil dar. Beispielsweise werden Filtertücher über die Filterplatten gehängt und müssen nicht zusätzlich verschraubt werden. Ein eklatanter Nachteil der Rahmenfilterpresse ist der oftmals schwierige Austrag des Filterkuchens. Insbesondere bei kompakten oder haftenden Materialen sitzt er auf dem unteren Rand des Rahmens auf und fällt nur unvollständig aus der Presse. Ein weiterer Nachteil ist der mit 600 – 800 kPa begrenzte Filtrationsdruck aufgrund der Rahmenkonstruktion.

Kammerfilterpresse:

Die Kammerfilterpresse stellt eine Weiterentwicklung der Rahmenfilterpresse dar. Bei der Kammerfilterpresse sind alle Kammerplatten identisch ausgeführt. Der Rahmen ist in jeder Platte integriert. Am äußeren Dichtrand sind die Platten planparallel konstruiert. Jede Platte besitzt im inneren Bereich auf beiden Seiten eine Vertiefung. Dieser Bereich wird auch Spiegel genannt. Dadurch entstehen die Kammern, in denen die Suspension meistens durch einen Kanal im Zentrum der Platten eingeführt wird. Über die Plattenoberflächen ist das Filtertuch gespannt. Der Ablauf des Filtrats erfolgt meist über Kanäle in den Ecken der Kammerplatten. Der verbesserte Kuchenaustrag ist der wesentliche Vorteil der Kammerfilterpresse gegenüber

[113] Vgl. Gasper/Oechsle/Pongratz (Hrsg.) (2000), S. 122 – 125.

der Rahmenfilterpresse. Aufgrund des schräg gestalteten Übergangs vom Dichtrand zum inneren Spiegel besitzt der Filterkuchen keine hohe Abstützmöglichkeit und fällt leichter aus der Kammerfilterpresse, als es bei der Rahmenfilterpresse der Fall ist. Ein weiterer Vorteil ist die höhere Druckstabilität. So werden Kammerfilterpressen oft mit 1600 kPa, Hochdruckfilterpressen mit 4000 kPa betrieben. Aus diesem Grund können kompaktere und homogenere Filterkuchen mit höherem Trockensubstanzgehalt erreicht werden. Für die Erreichung eines optimalen Ergebnisses muss die Kammerfilterpresse immer vollständig gefüllt sein. Sollen kleinere Mengen an Suspension gefiltert werden oder die Feststoffmenge in der Suspension sinkt, müssen einzelne Kammerplatten entfernt und so die Kapazität der Filterpresse verringert werden. Dieses Problem löst die Membranfilterpresse.

Membranfilterpresse:
Im Gegensatz zu den bereits beschriebenen Filterpressen, wird bei der Membranfilterpresse der Druck nicht nur durch die Suspension selbst, sondern auch durch die aufblasbaren Membranen, welche ein- oder beidseitig auf jeder oder jeder zweiten Filterplatte angebracht sind, aufgebaut. Ansonsten ist der Aufbau der Membranfilterpresse der Kammerfilterpresse sehr ähnlich. Die Membran wird nach Beendigung des Zuführvorganges der Suspension gegen den Filterkuchen gepresst. Durch diesen Vorgang muss die Membranfilterpresse für ein optimales Filterergebnis nicht vollständig gefüllt sein. Außerdem kann dadurch der Trockensubstanzgehalt gegenüber der Kammerfilterpresse um rund 5 – 10 Prozent gesteigert werden. Auch die Zykluszeit kann verringert werden. Außerdem entsteht bei der Membranfilterpresse ein besonders homogener und kompakter Filterkuchen, welcher für den möglichen anschließenden Waschvorgang essenziell ist. Der Waschvorgang wird im nächsten Unterkapitel detailliert erläutert.

6.1.2 Optionale Funktionen

Neben der Hauptfunktion einer Filterpresse, dem Trennen eines Fest-Flüssig-Gemisches, gibt es weitere Prozessschritte, die automatisiert ausgeführt werden können. Die bei ANDRITZ am häufigsten verkauften Optionen, welche auch in den Templates der Filterpressenanlage abgebildet werden, werden in diesem Unterkapitel behandelt.

Wie im vorherigen Unterkapitel beschrieben, müssen die einzelnen Filter-, Rahmen- oder Kammerplatten verschoben werden, um den Filterkuchen aus der Filterpresse entfernen zu können. Das kann manuell, halbautomatisiert oder vollautomatisiert ablaufen. Manuell bedeutet, dass die einzelnen Platten händisch, ohne etwaige Unterstützung, verschoben werden. Der halb- oder vollautomatische Plattentransport kann hydraulisch oder pneumatisch erfolgen. Beim halbautomatischen Betrieb werden die Bewegungen über Taster oder Touch Panel vom Bediener gesteuert. Im Gegensatz dazu ist beim vollautomatischen Betrieb kein Eingriff von Personen nötig. Die Steuerung, üblicherweise eine SPS (Speicherprogrammierbare Steuerung), steuert die Verschiebung der Platten nach Abschluss eines Filtrationsvorganges.

Um den Filterkuchen vollständig von deren Mutterlösung oder anderen Stoffen wie z.B. Salzen zu befreien, kann er gewaschen werden. Dabei wird nach dem Filtervorgang der Filterkuchen mit einer Waschflüssigkeit, wie z.B. Wasser, durchströmt. Generell gibt es zwei verschiedene Arten der Filterkuchenwäsche, die Gegenstrom- und die Gleichstromwäsche. Bei der Gleichstromwäsche wird die Waschflüssigkeit über denselben Weg, wie die Suspension, geleitet. Im Gegensatz dazu wird bei der Gegenstromwäsche die Waschflüssigkeit in die Filtratkanäle jeder zweiten Platte eingeführt. Die Waschflüssigkeit durchströmt den

Filterkuchen und fließt über die Rippenfelder der jeweils benachbarten Platten wieder zum Kopfstück zurück. Wie im vorherigen Unterkapitel bereits beschrieben, kann die erfolgreiche Filterkuchenwäsche nur mit einem homogenen und kompakten Filterkuchen durchgeführt werden. Enthält der Kuchen Risse oder Löcher, werden diese Bahnen von der Waschlösung bevorzugt benutzt und so wird nicht der gesamte Filterkuchen durchflossen. Der Filterkuchenwäsche wird oftmals die Kuchentrocknung nachgeschaltet. Um die Restfeuchte im Kuchen zu entfernen, wird er mit Luft durchströmt. Die Kuchentrocknung kann auch ohne Wäsche zum Einsatz kommen. [114]

Eine weitere wichtige Funktion, die die Effizienz einer Filterpressenanlage steigert, ist die automatische Filtertuchreinigung. Die Filtertücher werden durch die Partikeln, die durch das Filtertuch strömen, verschmutzt. Je nach Prozessparameter kann es notwendig sein die Filtertücher nach einem Monat oder schon nach ein paar Tagen zu reinigen. Vor der Entwicklung der automatischen Filtertuchreinigung mussten die Filtertücher manuell von den Platten abgenommen, in der Waschschleuder gereinigt und wieder auf die Platten montiert werden. Heutzutage können die Filtertücher durch sogenannte integrierte Filtertuchreinigungsanlagen direkt auf den Platten gereinigt werden. Wie in Abbildung 41 illustriert, besteht diese Anlage im Wesentlichen aus einem U-Profil mit Düsen auf beiden Plattenseiten, welche vertikal verfahrbar sind. Bei Beginn des Waschvorganges werden die Düsen am oberen Rand der Platten platziert. Während des Waschvorganges werden sie üblicherweise von einem Motor nach unten und dann wieder zurück in die Ausgangslage bewegt. Dadurch, dass die Platten von beiden Seiten gleichzeitig bespritzt werden, bleiben sie zwischen den Düsen zentriert. Die Filtertuchreinigungsanlage arbeitet mit einem Druck von ca. 10 MPa. [115]

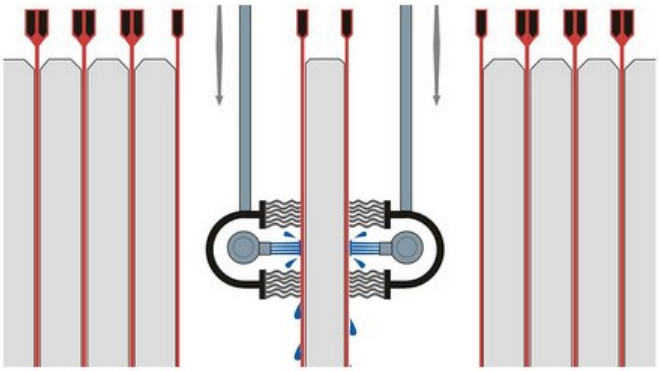


Abbildung 41: Filtertuchreinigung, Quelle: ANDRITZ (2018c), Online-Quelle [27.November.2018].

Um Feststoffe in einer Filterpresse abtrennen zu können, müssen sie eine gewisse Größe haben. Kolloidale Partikel (ca. <10 µm) wie Plankton, Bakterien oder Viren können mit einer Filterpresse üblicherweise nicht gefiltert werden. Außerdem verhindert die Brownsche Molekularbewegung ein Absinken der Teilchen und somit eine einfache Abtrennung durch Sedimentation. Des Weiteren bilden sich aufgrund der meist negativen Oberflächenladung ohne die Zugabe eines Hilfsmittels keine Flocken. Sollen diese Stoffe trotzdem abgetrennt werden, ist eine Vorbehandlung notwendig, welche die Partikel zu größeren Agglomerationen

[114] Vgl. Svarovsky (Hrsg.) (2000), S. 335 ff.

[115] Vgl. Kirchner (2018), Online-Quelle [27.November.2018].

zusammenfügt. Eine Methode ist die Koagulation und Flockung. Bei der Koagulation werden durch Zugabe eines Flockungsmittels in Form von Kationen, wie z.B. Aluminiumchlorid, Calciumhydroxid oder auch kationischen Polyelektrolyten, die Mikroteilchen destabilisiert. Das Ergebnis ist eine Agglomeration zu Teilchen im Millimeter-Bereich. Damit die Agglomerate eine stabile Verbindung eingehen, werden zusätzlich Flockungshilfsmittel (hochmolekulare Polyelektrolyte), meist Copolymere, hinzugegeben. [116]

6.2 Modellierung der Filterpressenanlage

Dieses Unterkapitel befasst sich mit der Modellierung der Filterpressenanlage. Im ersten Abschnitt werden die theoretischen Grundlagen zur verwendeten Modellierungssprache erläutert. Darauffolgend wird das Modell der Filterpressenanlage beschrieben.

6.2.1 Theoretische Grundlagen zur Modellierungssprache

Da in dieser Masterarbeit hauptsächlich die Modellierungssprache UML verwendet wird, liegt der Einsatz für die Modellierung der Konfigurationen der Filterpressenanlage auf der Hand. Jedoch können mit UML keine Abhängigkeitsbeziehungen, welche für die Modellierung der Filterpressenanlage benötigt werden, definiert werden. Des Weiteren stellt UML keine Formalitäten zur Darstellung von Varianten oder optionalen Komponenten bereit. Aus diesen Gründen muss für die Modellierung der Konfigurationsmöglichkeiten einer Filterpressenanlage auf eine alternative Notationsform zurückgegriffen werden. Im Folgenden werden unterschiedliche Modellierungssprachen verglichen und die Auswahl der Modellierungssprache OVM (Orthogonal Variability Modeling) begründet.

Als sich Anfang der 90er Jahre Kang, Cohen und andere am Carnegie Mellon Software Engineering Institute (SEI) Gedanken über die Wiederverwendung von Software im Rahmen von Domänenmodellen machten, entwickelten sie infolgedessen die Feature Oriented Domain Analysis (FODA). In diesem Variantenmodell können optionale, verpflichtende und sich gegenseitig ausschließende Varianten definiert werden. Das Problem bei FODA ist, dass die Aussage, was sich in den Varianten ändert, nur implizit dargestellt wird. In der Notation gibt es dafür keine Symbole. Eine neuere Methode zur Modellierung von Varianten ist die Modellierungssprache Orthogonal Variability Modeling (OVM). Der wesentliche Vorteil gegenüber der FODA besteht darin, dass die Variabilität eines Features als eigenständiges Modellierungselement behandelt wird. [117]

Neben der OVM gibt es eine Vielzahl weiterer Modellierungssprachen zur Darstellung von Varianten. Das Forschungsprojekt mecPro² verglich unter anderem unterschiedliche Modellierungssprachen nach sechs Hauptkriterien. Die einheitliche Darstellung, die Nachverfolgbarkeit, die Darstellung der Variabilität, die Bindezeit der Variabilität, die Darstellung der Varianten und die Darstellung der Restriktion wurden nach unterschiedlicher Gewichtung bewertet. In Abbildung 42 ist die Nutzwertanalyse der unterschiedlichen

[116] Vgl. Bratby (2016), S. 1 – 7.

[117] Vgl. Lippert/Graser (2016), Online-Quelle [27.November.2018], S. 2.

Modellierungssprachen illustriert. Die Kriterien werden mit *0 – 5* bewertet, wobei *0 nicht vorhanden* und 5 *vollständig vorhanden*, bedeutet. [118]

Kriterien	Gewichtung	OVM	CVL	Delta Modeling	SYSMOD	Feature Diagramme	COVAMOF
Einheitliche Darstellung	20%	5,00	5,00	5,00	2,50	5,00	5,00
über alle Entwicklungsphasen	50%	5	5	5	3	5	5
über alle Fachdisziplinen	50%	5	5	5	2	5	5
Nachverfolgbarkeit	20%	4,50	3,00	4,00	3,50	3,00	5,00
ins Basismodell	50%	5	5	5	5	5	5
über verschiedene Entwicklungsphasen	50%	4	1	3	2	1	5
Darstellung der Variabilität	17%	4,10	1,65	1,65	2,35	1,65	2,35
intern/ extern	35%	5	0	0	0	0	0
Zeit/ Raum	30%	2	2	2	2	2	2
optional/ erforderlich	35%	5	3	3	5	3	5
Bindezeit der Variabilität	2%	0,00	0,00	0,00	5,00	0,00	5,00
Bindezeit	100%	0	0	0	5	0	5
Darstellung der Varianten	25%	4,40	3,44	2,16	4,88	2,80	2,48
optional	24%	5	3	3	5	3	0
erforderlich	24%	5	3	3	5	3	0
alternativ	24%	5	3	3	5	3	5
Multiple Choice	16%	5	5	0	5	4	5
Variablen	12%	0	4	0	4	0	4
Darstellung der Restriktionen	16%	2,50	2,88	2,50	2,50	2,50	3,45
bedingend	25%	5	5	5	5	5	5
ausschließend	25%	5	5	5	5	5	5
mehrfach Abhängigkeiten	19%	0	2	0	0	0	5
negative Abhängigkeiten	19%	0	0	0	0	0	0
If-Abfrage	12%	0	0	0	0	0	0
Nutzwert	100%	4,10	3,20	3,02	3,32	2,98	3,67

Abbildung 42: Nutzwertanalyse der Modellierungssprachen zur Abbildung von Variabilität, Quelle: Schulte/Mayerhofer (2016), S. 20.

Die Anforderungen an eine Modellierungssprache zur Abbildung von Variabilität sind in dieser Masterarbeit den im vorherigen Absatz beschriebenen Anforderungen sehr ähnlich. Wie in obiger Abbildung zu sehen, schneidet die Notation OVM mit relativ klarem Abstand am besten ab. Aus diesem Grund wird in dieser Arbeit grundsätzlich OVM für die Darstellung der Variabilität einer Filterpressenanlage verwendet.

In nachfolgender Abbildung 43 sind die Hauptelemente der Notation OVM ersichtlich. Die Hauptobjekte der OVM-Notation sind Variation Points (VP) und Variants (V). Ein VP beschreibt ein variables Objekt und eine V die möglichen Instanzen dieses VPs. Jeder VP muss mindestens zu einer Variant in Relation stehen und umgekehrt. Sie werden als Dreieck und Rechteck dargestellt. Obligatorische VPs werden durch eine solide Linie, optionale durch eine strichlierte Linie gekennzeichnet. Weiters können mittels OVM alternative Auswahlmöglichkeiten mit Kardinalitäten versehen werden. Beispielsweise kann in Abbildung 43 der Variation Point *Messaging* aus einer der beiden Variants *SMS* und *MMS* oder aus beiden bestehen. Eine weitere wichtige Beziehung sind die *Requires*- und die *Excludes*-Beziehung. Die *Requires*-Beziehung wird durch einen gerichteten, strichlierten Graphen, die *Excludes*-Beziehung durch einen strichlierten Pfeil, welcher in beide Richtungen zeigt, dargestellt. Alternativ können diese Beziehungen mit dem jeweiligen Schlüsselwort versehen werden. Die *Requires*-Beziehung beschreibt die Abhängigkeit eine Variant oder eines VPs von einer anderen Variant oder eines anderen VPs. Im unteren Beispiel kann die Variant *Currency Exchange*

[118] Vgl. Schulte/Mayerhofer (2016), S. 16 – 20.

nur in Verbindung mit der Variant *Calculator* bestehen. Wird die *Requires*-Beziehung durch eine *Excludes*-Beziehung ersetzt, darf die jeweilige Variant nicht in Kombination mit der anderen Variant existieren. [119]

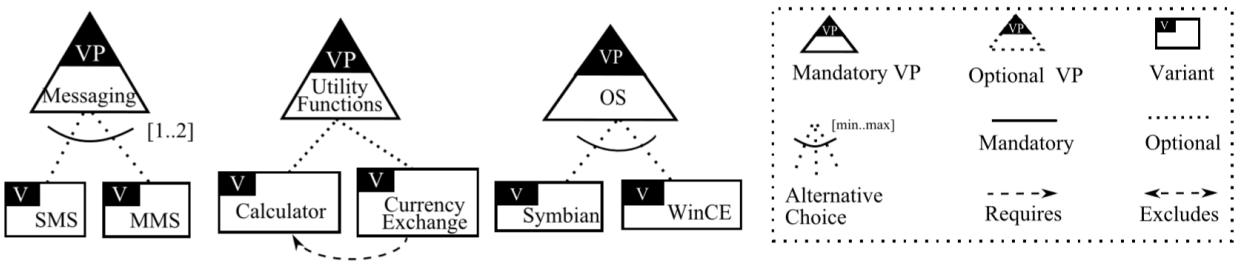


Abbildung 43: Notation OVM, Quelle: Roos-Frantz/Benavides/Ruiz-Cortés (2011), Online-Quelle [27.November.2018], S. 7.

Die Notation OVM stellt eine gute Basis für die Modellierung der Konfigurationsmöglichkeiten einer Filterpressenanlage dar, jedoch entspricht sie nicht exakt den Anforderungen. Aus diesem Grund wurden vier weitere Elemente, welche in Abbildung 44 ersichtlich sind, eingeführt. Das erste Element *System* entspricht einem gesamten System, wie z.B. einer Gesamtanlage. Um Komponenten, die keine Variabilität besitzen darstellen zu können, wurden die Elemente *obligatorische Komponente* und *optionale Komponente* eingeführt. In Anlehnung an OVM werden obligatorische Teile mit einer soliden, optionale mit einer strichlierten Linie dargestellt. Im Zuge der Modellierung der Filterpressenanlage kam die Notwendigkeit einer zusätzlichen Abfrage zur *Requires*-Beziehung auf. Deswegen wurde diese Beziehung, welche auch bei OVM enthalten ist, durch das Schlüsselwort *if* und einem frei wählbaren Text ergänzt. Dieser Text muss natürlich aussagekräftig und eindeutig sein.

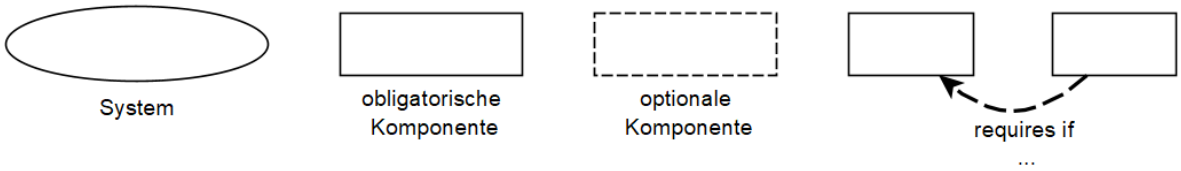


Abbildung 44: ergänzende Elemente zur OVM, Quelle: Eigene Darstellung.

6.2.2 Beschreibung des Modells

In diesem Unterkapitel wird die Konfiguration einer Filterpressenanlage modellbasiert dargestellt. Wie im vorherigen Unterkapitel beschrieben, dient als Basis die Notation Orthogonal Variability Modeling, die um einige Elemente erweitert wurde. Die Bedeutung der grafischen Elemente, sowie allgemeine theoretische Grundlagen zu dieser Modellierungssprache können ebenfalls im Unterkapitel 6.2.1 nachgelesen werden.

In Abbildung 45 ist das System *Filter press*, welches die gesamte Filterpresse darstellt, illustriert. *Line* und *Common* sind Komponenten des Systems *Filter press* und sind dem somit hierarchisch untergeordnet. Weiters ist in dieser Abbildung ersichtlich, dass eine Filterpresse aus 1 bis 10 Linien (*Line*) und einem optionalen allgemeinen Teil (*Common*) besteht. Im allgemeinen Teil werden all jene Komponenten

[119] Vgl. Shahin (2014), S. 40 f.

zusammengefasst, die eine oder mehrere Filterpressenlinien bedienen und somit nicht einer Linie zugeordnet werden können.

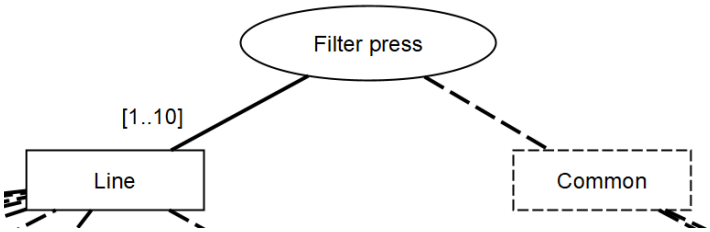


Abbildung 45: Modell der Filterpressenanlage – Gesamtsystem, Quelle: Eigene Darstellung.

Eine Filterpressenlinie (*Line*) besteht aus einem obligatorischen Teil Presse (*Press*) und aus mehreren optionalen Komponenten, wie dem VP *Cake transport*. Außerdem ist in Abbildung 46 ersichtlich, dass sich eine Filterpresse *(Press)* aus dem Hauptteil (*Press Main*) und aus Optionen (*Press Options*) zusammensetzt. Die Variations des Teiles *Press Main* unterscheiden sich durch deren elektrische Antriebsleistung. Generell werden in dieser Masterarbeit Seitenholmpressen (*SB*- side bar) behandelt. Unabhängig davon kann das Plattenpaket in Form von Rahmen-, Kammer-, oder Membranfilterpressen ausgeführt sein. Die Theorie zu den unterschiedlichen Pressentypen wird im Unterkapitel 6.1.1 beschrieben. Da es aus Sicht der Automatisierung keinen großen Unterschied macht, welche Form bzw. welchen Aufbau die Platten haben, kann für alle drei Typen ein Template verwendet werden. Alle Variations des VP *Press* können im Anhang 3 nachgeschlagen werden.

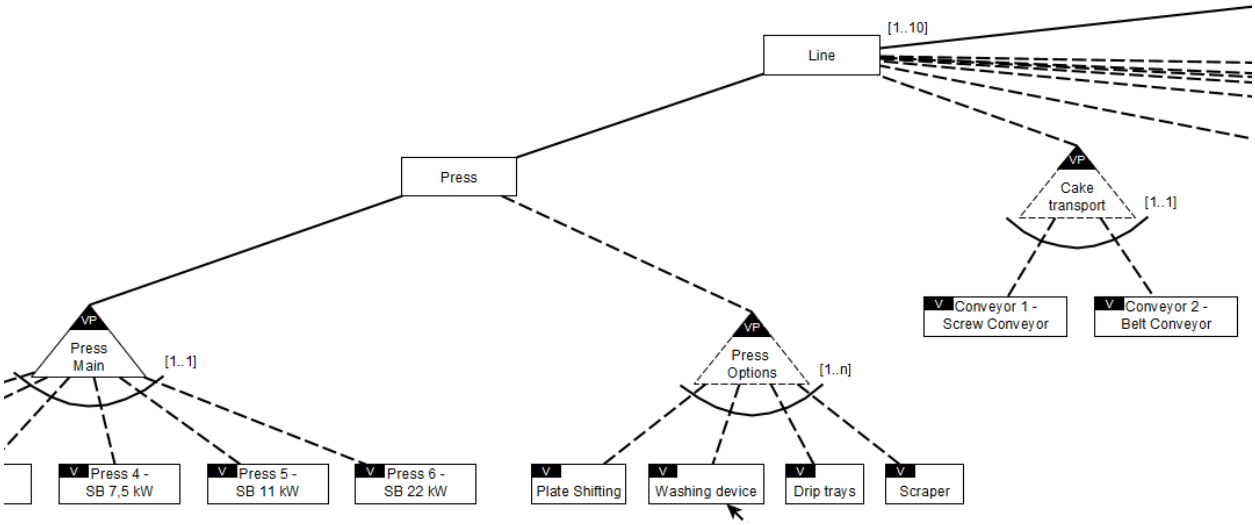


Abbildung 46: Modell der Filterpressenanlage – Line Teil 1, Quelle: Eigene Darstellung.

Der optionale VP *Press Options* kann sich aus mindestens einer und maximal allen Variations zusammensetzen. Die Variation *Plate Shifting* steht für die automatische Plattenverstellung, welche im Unterkapitel 6.1.2 näher erläutert wird. Die zweite Variant *Washing device* ist ein Teil der automatischen Filtertuchreinigung, nämlich das U-Profil inklusive der Düsen. Die Funktion der automatischen Filtertuchreinigung wird ebenfalls im Unterkapitel 6.1.2 beschrieben. Weitere optionale Variations der optionalen Teile der Filterpresse sind *Drip trays* und *Scraper*. *Drip trays* sind Tropfbleche, welche unter der Filterpresse montiert werden und z.B. das Waschwasser, welches beim Reinigen der Filtertücher aus der Filterpresse fließt, auffangen. An dem Tropfblech kann beispielsweise ein Rohr montiert werden, sodass die Flüssigkeiten

gezielt abgeführt werden können. Die letzte Variation des Variation Points *Press Options* beschreibt den Schaber (*Scraper*). Der Hauptteil des Schabers ist wie bei der automatischen Filterreinigung ein verfahrbarer Schlitten mit einem U-Profil. Anstatt den Düsen ist auf dem U-Profil ein Schaber montiert. Haftet der Filterkuchen besonders gut auf den Filtertüchern, kann er mittels des automatischen Schabers abgelöst werden. Eine optionale Komponente der Linie (*Line*) ist der VP *Cake transport*. Mit diesem kann der Filterkuchen entweder mittels eines Schneckenförderers (*Conveyor 1 – Screw Conveyor*) oder mittels eines Förderbandes (*Conveyor 2 – Belt Conveyor*) abtransportiert werden. Wie die Kardinalität *[1..1]* zeigt, schließen sich diese Variations gegenseitig aus.

In nachfolgender Abbildung 47 ist der zweite Teil der optionalen Komponenten einer Filterpressenlinie (*Line*) dargestellt. Damit der Filterkuchen bei Öffnen der Filterpresse nicht durch die im Zulaufkanal der Platten verbliebene Suspension verunreinigt wird, muss sie vorher entfernt werden. Aus diesem Grund wird vor dem Öffnen der Filterpresse Druckluft in entgegengesetzter Richtung der Suspensionszuführung im Zulaufkanal angelegt. Dadurch wird die verbliebene Suspension ausgeblasen (*Core blow*). Die Option *Core blow* besteht aus einem Einlass (*Inlet*) und einem Auslass (*Outlet*). Für die Komponente Auslass (*Outlet*) gibt es nur eine Variante, somit ist sie obligatorisch. Beim Einlass kann zwischen zwei Varianten, einer mit Druckluftaufbereitung (*Core blow 1 – with air supply*) und einer ohne (*Core blow 2 – without air supply*) gewählt werden. Der VP *Filtrate outlet* betrifft den Abtransport der gefilterten Suspension des Filtrates. Eine Filterpresse besitzt üblicherweise mehrere Filtratauslasskanäle. Bei der ersten Version *Filtrate 1* werden alle Auslasskanäle zu einem Rohr zusammengeführt. Bei der zweiten Variante *Filtrate 2* werden zusätzlich Ventile angebracht. So kann jeder Auslasskanal mit einem separaten Ventil geöffnet oder geschlossen werden.

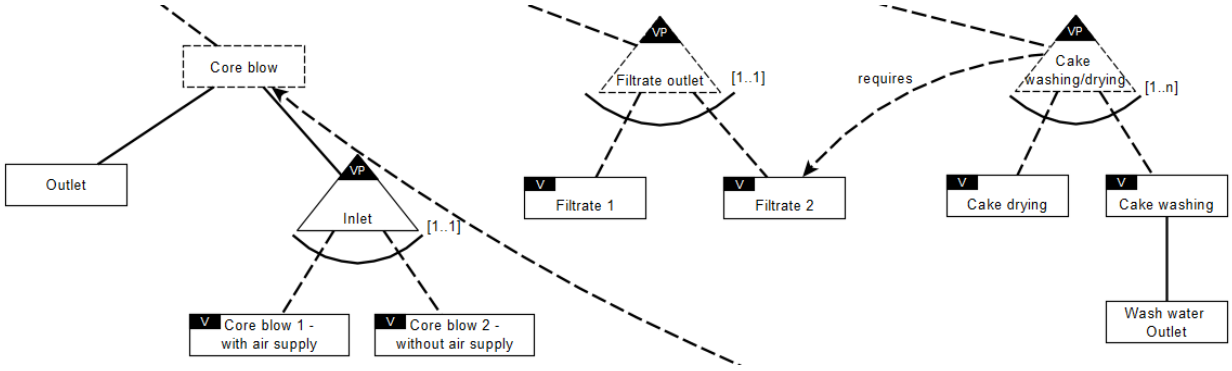


Abbildung 47: Modell der Filterpressenanlage – Line Teil 2, Quelle: Eigene Darstellung.

Wie in Abbildung 47 ebenfalls ersichtlich, wird die Variation *Filtrate 2* auch für die optionale Funktion Kuchenwaschung bzw. -trocknung (*Cake washing/drying*) benötigt. Der Grund dafür ist, dass das Waschwasser oder auch die Luft nicht durch alle Auslasskanäle aus der Filterpresse austreten darf. Die theoretischen Grundlagen zur Filterkuchenwäsche (*Cake washing*) und -trocknung (*Cake drying*) werden im Unterkapitel 6.1.2 beschrieben. Die Filterkuchenwäsche kann mit oder ohne der Filterkuchentrocknung zum Einsatz kommen. Dasselbe gilt umgekehrt. Jedoch ist für die Kuchenwäsche die Komponente *Wash water Outlet* obligatorisch. Diese Komponente besteht aus Rohrleitungen und Ventilen zur Abfuhr des Waschwassers.

Die Option *Squeezing, welche* in Abbildung 48 illustriert wird, kommt nur bei Membranfilterpressen zum Einsatz. Wie im Unterkapitel 6.1.1 beschrieben, wird durch die aufblasbaren Membranen zusätzlicher

Druck auf die Suspension und den Filterkuchen aufgebracht. Die Membranen können mittels Wasser (*Water*) oder Luft (*Air*) mit Druck beaufschlagt werden. Wird die Variation *Water* gewählt, besteht sie immer aus zwei Teilen. Einem Wassertank (*Water storage*) und einer Pumpe (*Pump*), um das Wasser in die Filterpresse zu pumpen. Je nach Anwendungsfall können drei verschiedene Pumpensysteme zum Einsatz kommen, entweder eine Exzenterschneckenpumpe (*Eccentric screw pump*) oder eine Kreiselpumpe mit (*Centrif. Pump active return*) oder ohne aktive Rückführung (*Centrif. Pump passive return*). Bei aktiver Rückführung kann die Pumpe mittels Ventile so geschalten werden, dass sie für den Abpumpvorgang des Wassers in entgegengesetzter Richtung wirkt. Bei passiver Rückführung wird die Pumpe einfach abgeschaltet und das Wasser fließt über denselben Weg zurück. Wird die Nachpressung mit Luft (*Air*) betrieben, ist die Komponente *Squeeze Air 1* obligatorisch. Soll die Luft, welche für das Aufblasen der Membranen verwendet wurde, auch für das Ausblasen des Einlasskanals eingesetzt werden (*Core blow*), wird die optionale Komponente *Squeeze Air 2* benötigt. Somit kann diese Komponente nur in Kombination mit der Komponente *Core blow* sinnvoll eingesetzt werden.

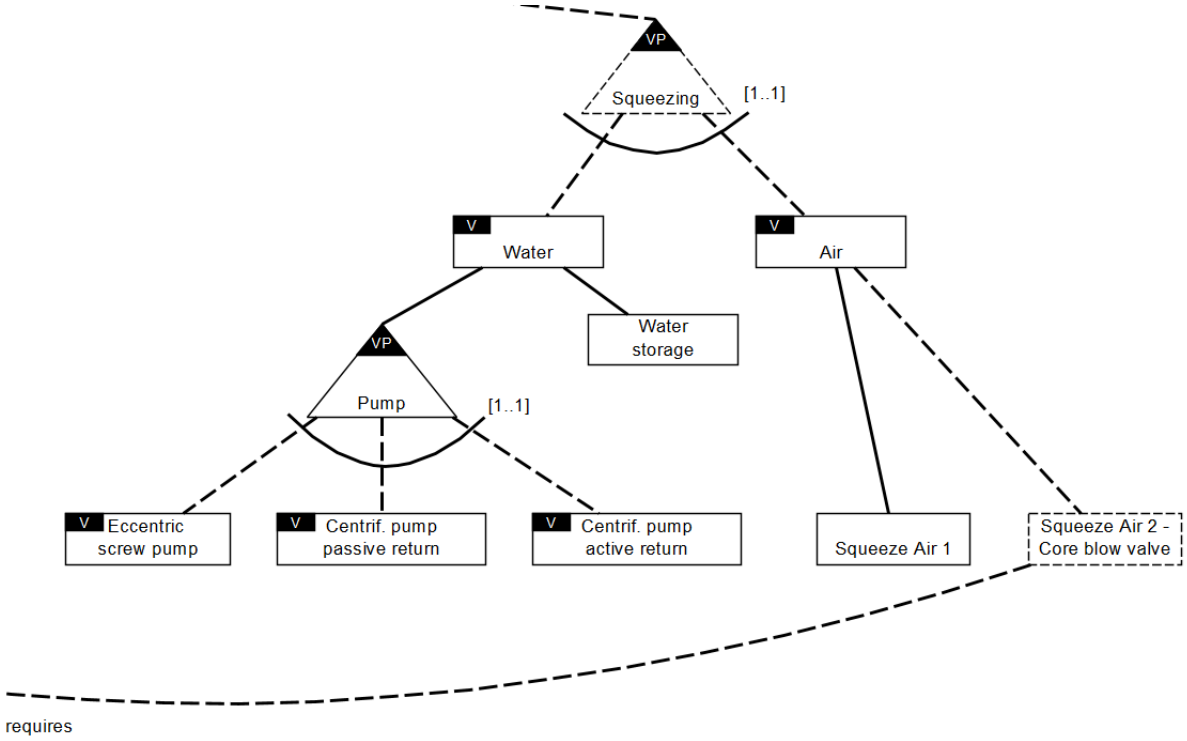


Abbildung 48: Modell der Filterpressenanlage – Line Teil 3, Quelle: Eigene Darstellung.

Die Option *Feed*, welche in Abbildung 49 dargestellt wird, ist für die automatisierte Zuführung der Suspension zur Filterpresse zuständig. Sie besteht generell aus zwei Teilen. Einem konstanten allgemeinen Teil (*Common*), welcher Rohrleitungen mit Messungen und Ventilen beinhaltet und einem variablen Teil *Feed Pump*. Wie der Name bereits aussagt, variieren hier die Typen der Pumpen. Grundsätzlich kann eine Kreisel- (*Feed 1*), Kolbenmembran- (*Feed 2*), Druckluftmembran- (*Feed 3*) oder Exzenterschneckenpumpe (*Feed 4*) zum Einsatz kommen. Weiters können verschiedene Kombinationen der beschriebenen Pumpentypen eingesetzt werden. Im Anhang 3 können alle Kombinationsmöglichkeiten nachgeschlagen werden.

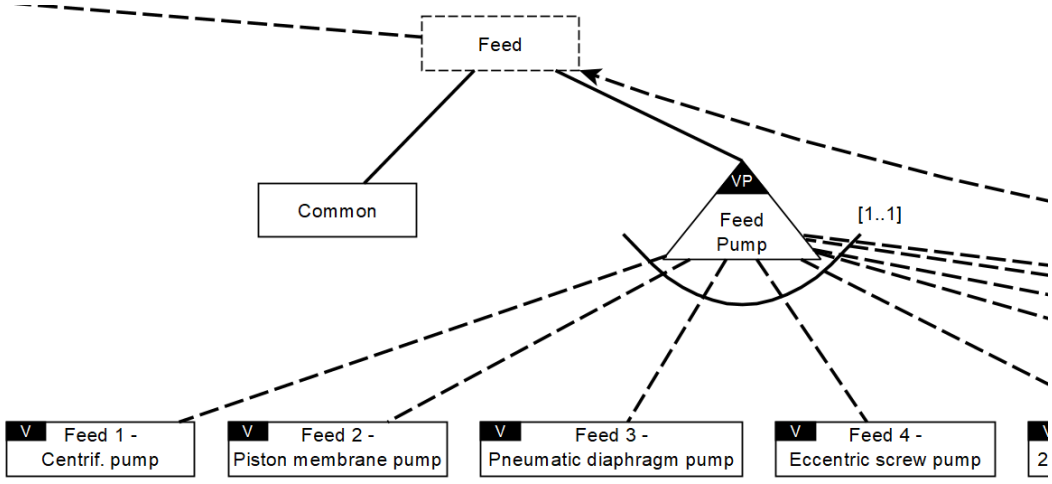


Abbildung 49: Modell der Filterpressenanlage – Line Teil 4, Quelle: Eigene Darstellung.

Wie am Anfang dieses Unterkapitels bereits beschrieben, werden unter dem Begriff *Common* all jene Anlagenteile zusammengefasst, die nicht direkt einer Filterpressenlinie zuordenbar sind. In Abbildung 50 ist zu sehen, dass dieser Teil und auch alle untergeordneten Komponenten optional sind. In der ersten Komponente Schlammtank (*Sludge tank*) wird die Suspension gelagert. Kommt ein Schlammtank zum Einsatz, wird ein Zuführsystem (*Feed*), welches im vorhergehenden Absatz beschrieben wird, für den Transport der Suspension vom Tank zur Filterpresse benötigt. Die zweite optionale Komponente *Polyelectrolyte station* stellt die Vorbehandlung der Suspension, welche im Unterkapitel 6.1.2 beschrieben wird, dar. Der Name *Polyelectrolyte station* kommt vom Einsatz des Flockungs- und Flockungshilfsmittels in Form von Polyelektrolyten. Sollen die Filtertücher in regelmäßigen Abständen von Kalk oder ähnlichen Rückständen befreit werden, kann die Station *Sour filtercloth cleaning* installiert werden. Die Filterpresse wird mit Säure gespült und somit werden die Filtertücher und die Filterpresse gereinigt.

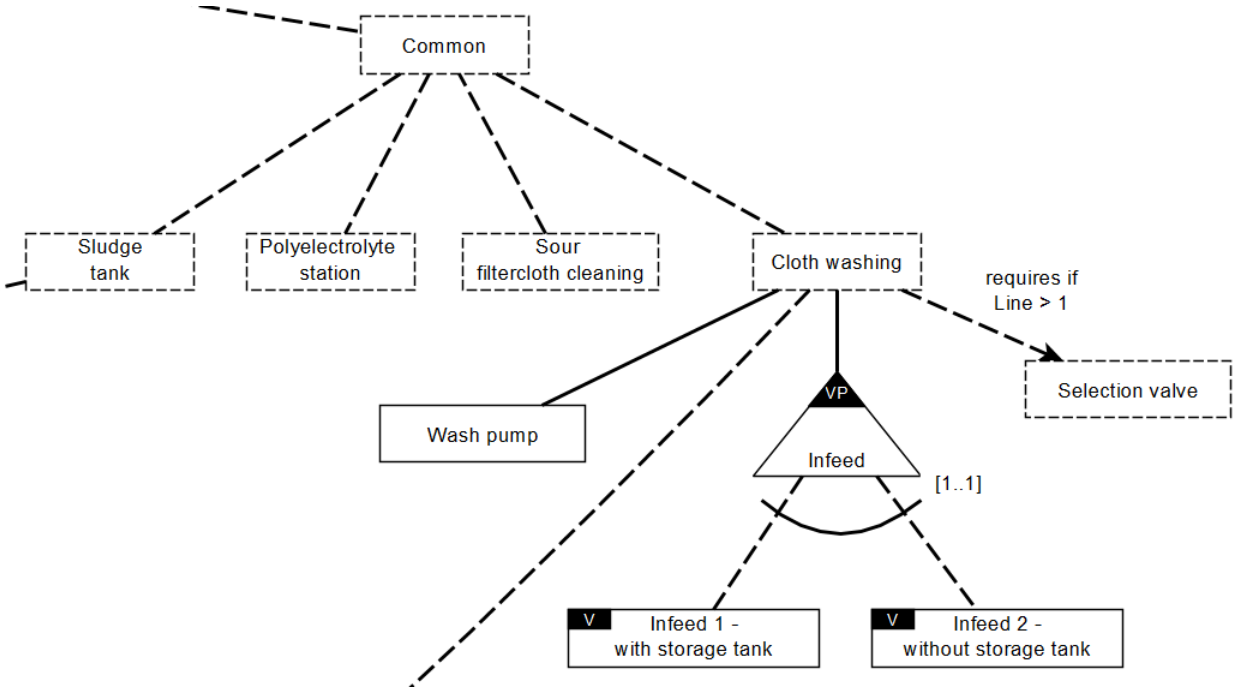


Abbildung 50: Modell der Filterpressenanlage – Common, Quelle: Eigene Darstellung.

Eine weitere Einrichtung zur Reinigung der Filtertücher ist die optionale Komponente *Cloth washing*. Sie ist der zweite Teil der automatischen Filtertuchreinigung, die im Unterkapitel 6.1.2 erläutert wird. Im Konkreten beinhaltet dieser Teil die Lagerung und Zuführung des Waschwassers zum *Washing device*, welches in Abbildung 46 dargestellt wird. Somit macht es keinen Sinn die Komponente *Cloth washing* ohne dem *Washing device* zu installieren. Die Zuführung des Waschwassers (*Wash pump*) muss bei Einsatz der automatischen Filtertuchreinigung immer vorhanden sein. Des Weiteren ist eine der beiden Versorgungsvarianten mit (*Infeed 1 – with storage tank*) oder ohne (*Infeed 2 – without storage tank*) Lagertank obligatorisch. Versorgt eine Komponente *Cloth washing* mehrere Filterpressen mit Waschwasser, wird die Option *Selection valve* benötigt. Diese beinhaltet Ventile zur Steuerung der Zuführung zu den unterschiedlichen Filterpressen.

6.3 Erstellung der Templates

Um die Filterpressenanlage mithilfe des *Metris Engineering Configurators* vollständig konfigurieren zu können, müssen Templates im Engineering Programm Comos erstellt werden. Als Vorlage dient das Modell, das im vorherigen Unterkapitel erläutert wird. Alle in diesem Unterkapitel beschriebenen Objekte und Klassen werden im Unterkapitel 5.3.1 beschrieben.

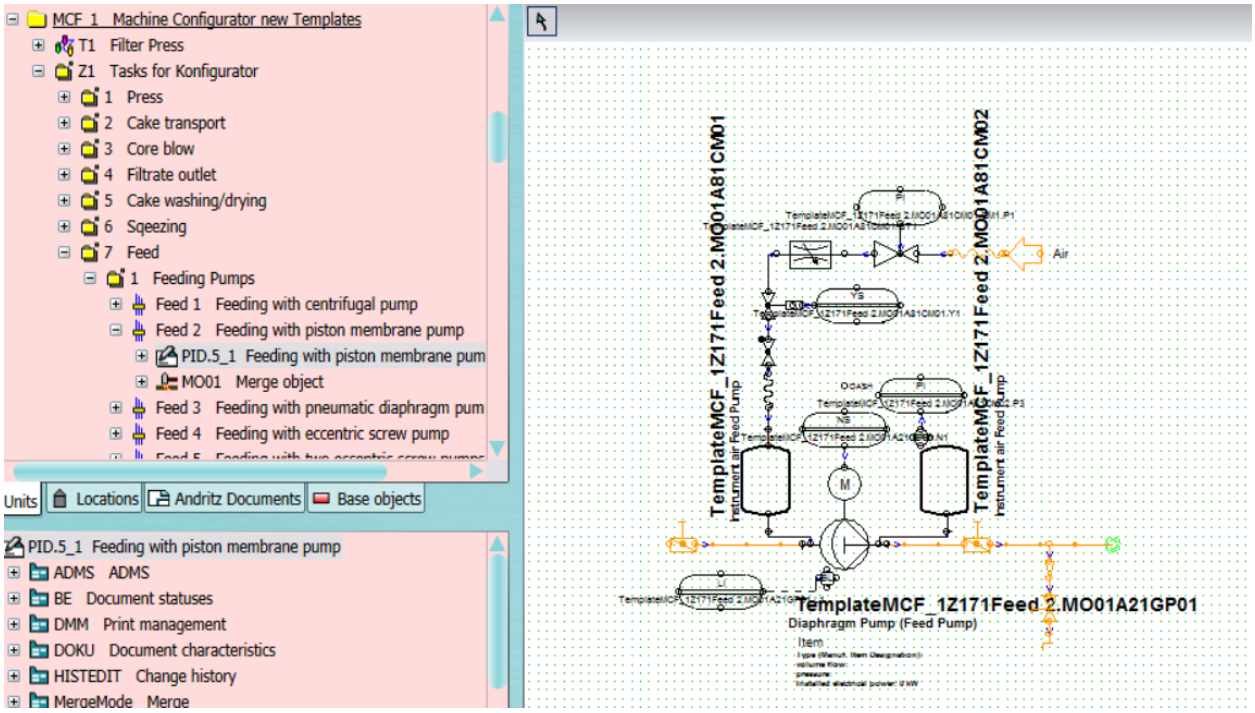


Abbildung 51: Templates – Filterpressenanlage, Quelle: Eigene Darstellung.

Wie in Abbildung 51 illustriert, sind die Templates im Comos hierarchisch aufgebaut. So entspricht z.B. das Vorlagenobjekt *7 – Feed* der optionalen Komponente *Feed*, welche in Abbildung 49 dargestellt ist. Das hierarchisch untergeordnete Objekt *1 – Feeding Pumps* ist äquivalent zum Variation Point *Feed Pump* des Modells. Die einzelnen Variations werden im Comos als Pfadmakros angelegt. Diese Variations haben im Comos und im Modell dieselbe Bezeichnung. Alle Objekte, die im Modell der Filterpressenanlage dargestellt sind, sind auch im Comos in Form von Vorlagenobjekten und Pfadmakros verfügbar. Ein weiteres wichtiges Objekt ist das *Merge-Objekt*. Unter dem in Abbildung 51 dargestellten *Merge-Objekt*,

welches unter dem Pfadmakro *Feed 2 – Feeding with piston membrane pump* angelegt ist, sind alle Objekte, wie z.B. Pumpen, Motoren, Instrumente, etc., die der Variation *Feed 2* zugehörig sind, verfügbar. In der rechten Hälfte der Abbildung 51 ist das P&ID dieser Variation ersichtlich. Jede Komponente, die im Modell nicht weiter zerlegt wurde, besitzt im Comos ihr eigenes Modul-P&ID, auf dem die zugehörigen Objekte platziert sind.

Der in Abbildung 52 dargestellte Verknüpfungsplan dient als Grundlage für die Anzeige der grafischen Benutzeroberfläche. Alle *Conditions*, die am Verknüpfungsplan platziert sind, werden auch auf der grafischen Benutzeroberfläche angezeigt. Auf dem Auszug des in Abbildung 52 dargestellten Verknüpfungsplans sind zwei *Conditions*, zwei *Actions* und ein *Dependency*-Objekt zu sehen. Die Condition *Feed System* (rot markiert) ist eine optionale Komponente und mit der Action *Feed System* (blau markiert) verbunden. Diese Verbindung bedeutet, dass bei Auswahl der Condition *Feed System*, die Action *Feed System* ausgeführt wird. Wie im Modell des Unterkapitels 6.2.2 beschrieben, darf die Komponente *Sludge tank*, welche ebenfalls optional ist, nur zum Einsatz kommen, wenn das *Feed System* in Verwendung ist. Diese requires-Beziehung wird durch das *Dependency*-Objekt (grün markiert) umgesetzt. Das *Dependency*-Objekt kann um beliebig viele Anschlüsse erweitert werden. Weiters besitzt das *Dependency*-Objekt eine Option *Single Selection*. Wird diese Option ausgewählt, erscheint anstatt dem *M* im *Dependency*-Objekt eine *1*. Nun stehen alle auf der rechten Seite angeschlossenen Objekte in einer exclude-Beziehung.

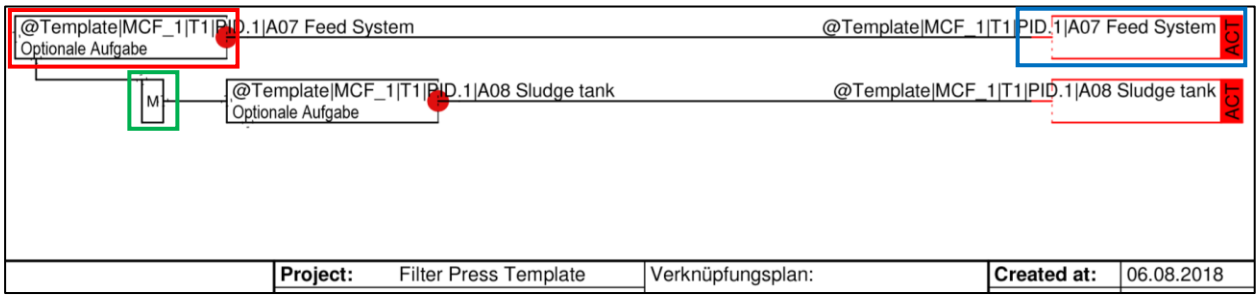


Abbildung 52: Verknüpfungsplan – Filterpressenanlage, Quelle: Eigene Darstellung.

Neben den in Abbildung 52 dargestellten logischen Operatoren, der *Condition*, der *Action* und dem *Dependency*-Objekt wurden drei weitere logische Operatoren entwickelt. Das *UND*, das *ODER* und die *Negation* haben dieselbe grafische Ausprägung wie das *Dependency*-Objekt. Der einzige Unterschied ist das Zeichen im Objekt. So wird z.B. beim *UND*-Objekt anstatt dem *M*, ein *&* angezeigt. Die *UND*- und *ODER*-Objekte sind ebenfalls flexibel aufgebaut, sodass sie auf der linken Seite um beliebig viele Anschlüsse erweitert werden können.

In Abbildung 53 sind die Eigenschaften der im vorherigen Absatz beschriebenen Condition *Feed System* und der Action *Feed System* dargestellt. Mit der rot markierten Combo-Box kann ausgewählt werden, ob es sich um eine optionale oder um eine obligatorische Komponente handelt. In den blau markierten Bereichen ist die Verlinkung der Condition und der Action zur Instanz *Feed System* der Klasse *eine Baugruppe einkopieren* ersichtlich.

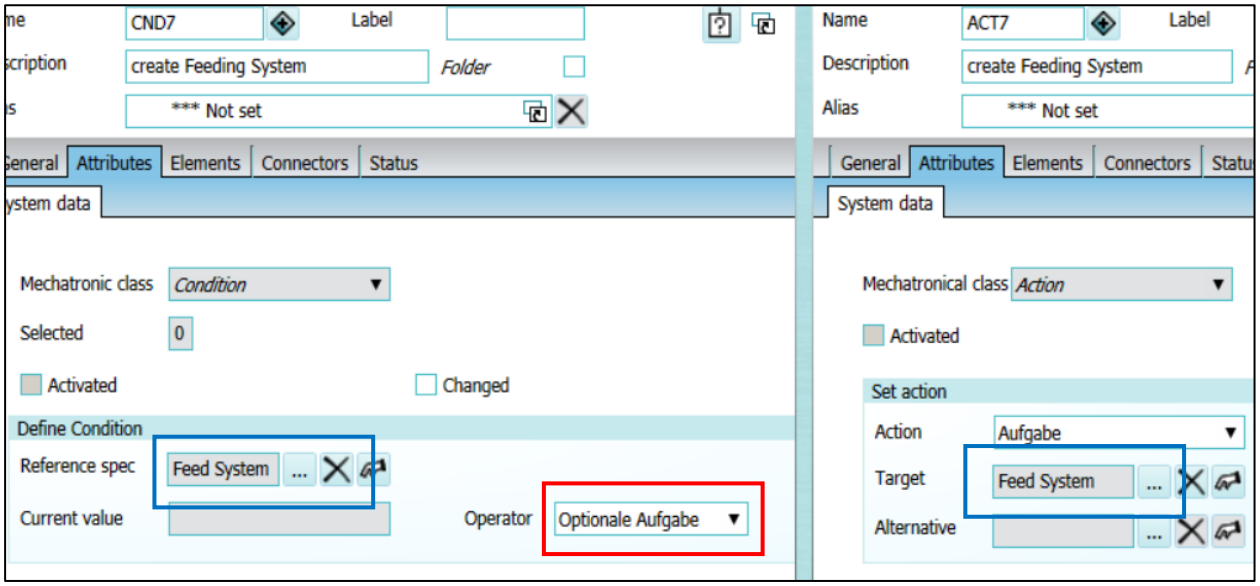


Abbildung 53: Condition, Action *Feed System*, Quelle: Eigene Darstellung.

Die Eigenschaften der Instanz *Feed System* der Klasse *eine Baugruppe einkopieren,* vereinfacht eBlock *Feed System* genannt, sind in Abbildung 54 illustriert. Das grün markierte Attribut *Root object for target search* entspricht dem in Abbildung 28 dargestellten *Startobjekt für Auswahlliste* und verlinkt das Vorlagenobjekt *Feeding Pumps*, welches in Abbildung 51 ersichtlich ist, mit dem eBlock *Feed System*. Somit stehen alle unter dem Vorlagenobjekt *Feeding Pumps* angelegten Pfadmakros als Auswahl bei der grafischen Benutzeroberfläche zur Verfügung.

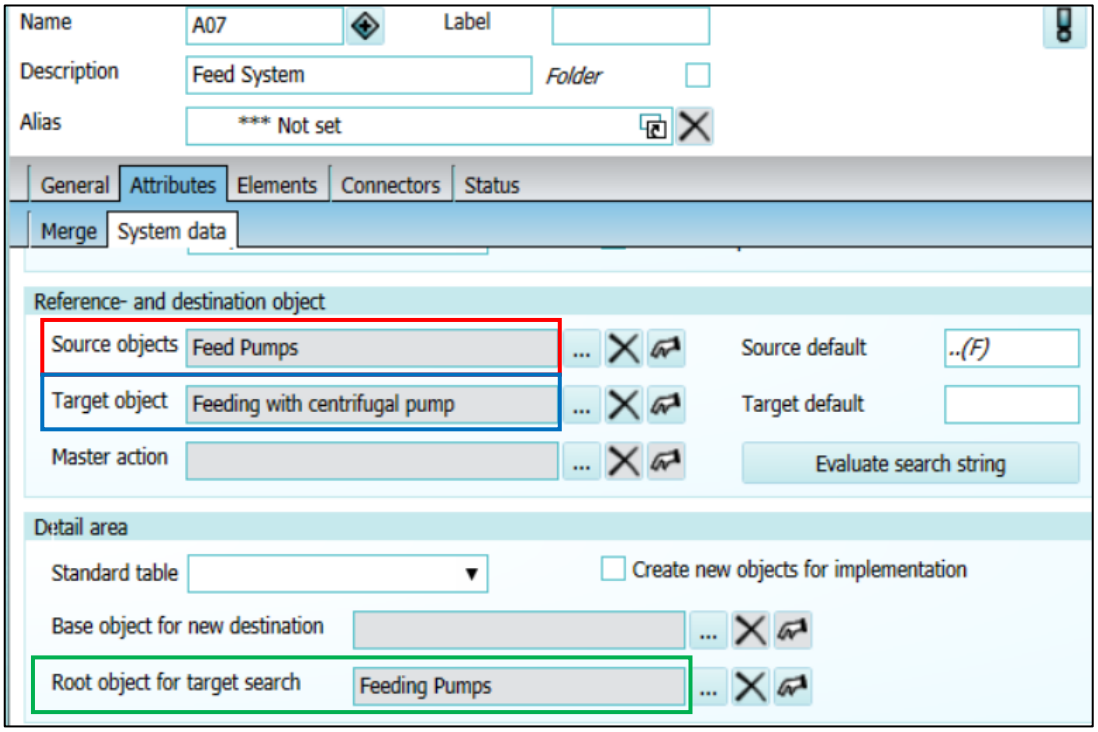


Abbildung 54: eBlock *Feed System*, Quelle: Eigene Darstellung.

Das Attribut *Target object* (in Abbildung 54 blau markiert) oder *Zielobjekt* definiert das Objekt, welches vom *Metris Engineering Configurator* angelegt wird. Ist kein Objekt im Attribut *Root object for target search*

verlinkt, werden immer diejenigen Objekte angelegt, die im blau markierten Feld verlinkt sind. Ist dieses Attribut jedoch befüllt, ändert sich durch die Auswahl des Benutzers auf der grafischen Benutzeroberfläche die Verlinkung. Wählt der Benutzer beispielsweise *Feed 3* aus, ändert sich die Verlinkung von *Feed 1* auf *Feed 3*.

Das Attribut *Source object* (rot markiert) oder in Abbildung 28 *Quellobjekt* genannt, definiert den Einfügepunkt der Objekte des Modul-P&IDs. In Abbildung 54 besitzt es eine Verlinkung zur Graybox (Platzhalter) *Feed Pumps*, welche in Abbildung 55 rot markiert ist. In dieser Abbildung ist ein Auszug des Haupt-P&IDs ersichtlich. Auf diesem sind außer einer Legende und den Grayboxen, welche die Einfügepunkte der Objekte der einzelnen Modul-P&IDs definieren, keine Objekte platziert.

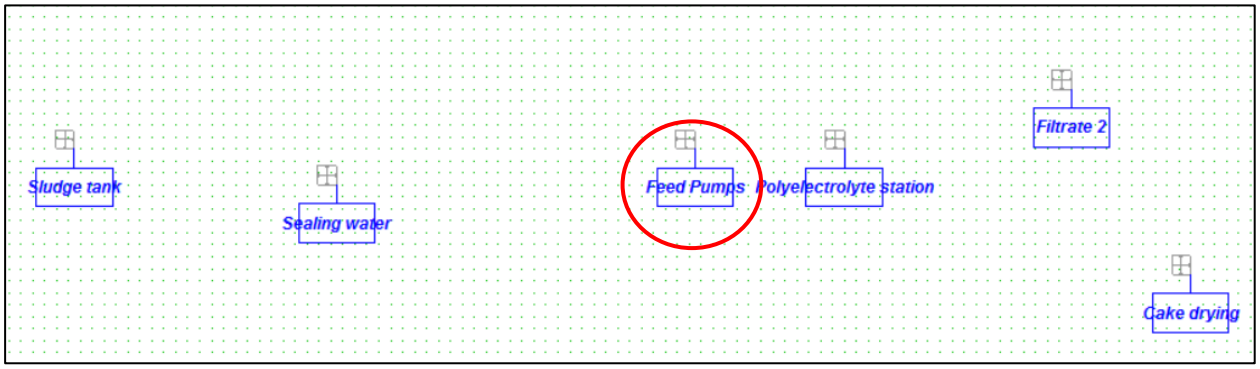


Abbildung 55: Haupt-P&ID – Filterpressenanlage, Quelle: Eigene Darstellung.

In Abbildung 56 sind weitere wichtige eBlocks, wie die Instanz *Merge Feed Pumps* der Klasse *Objekte in Anlagenstruktur verschieben (Merge)* dargestellt. Diese merged (verschiebt und vergleicht) die Objekte, die vom eBlock *A07 – Feed System* unter der Graybox angelegt wurden mit der bestehenden Anlagenstruktur. Im Template kann dem eBlock *Merge Feed Pumps* das zu verschiebende Objekt nicht zugewiesen werden, da es erst vom *Metris Engineering Configurator erstellt wird* und somit im Template nicht existiert. Aus diesem Grund wird ein Suchtest verwendet (rot markiert). Der in Abbildung 56 dargestellte Suchtext sucht nach dem Merge Objekt mit dem Namen *MO1*, welches unter der Graybox mit dem Namen *GB07.1* angelegt wird. Alle Objekte, welche dem Objekt *MO1* hierarchisch untergeordnet sind, werden mit der bereits bestehenden Anlagenstruktur gemerged. Der Ort der Anlagenstruktur wird mit dem Attribut *Target object* (blau markiert) definiert.

Wird der eBlock *A07 – Feed System* ausgeführt, werden alle darunterliegenden eBlocks ebenfalls ausgeführt. Im Modell der Filterpressenanlage im Unterkapitel 6.2.2 ist ebenfalls beschrieben, dass die optionale Komponente *Feed* immer aus einem Teil *Common* und einem Teil *Feed Pump* besteht. Der eBlock *A07 – Feed System* legt eine Variant des VP *Feed Pump* an. Dadurch, dass die Komponente *Common* bei Verwendung der Option *Feed* immer zum Einsatz kommt und es davon nur eine Variante gibt, wird diese Komponente mittels dem eBlock *EB10 – Feed common Part* bei Ausführung des eBlocks *A07 – Feed System*, automatisch angelegt. Der eBlock *EB01 – Merge Feed common Part* merged danach die Objekte dieser Komponente mit der bestehenden Anlagenstruktur.

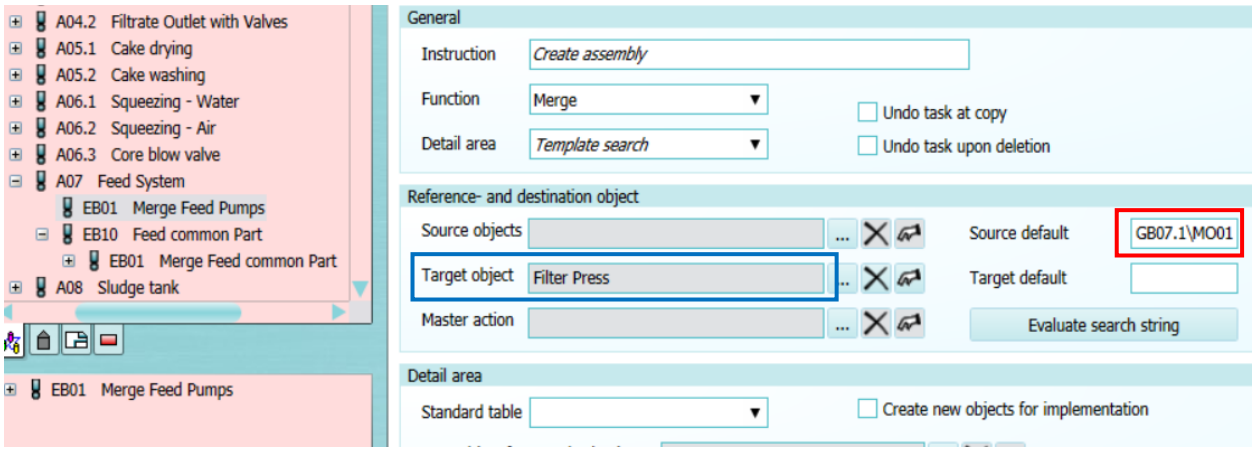


Abbildung 56: eBlock *Merge Feed Pumps*, Quelle: Eigene Darstellung.

Zusätzlich zu den in diesem Unterkapitel bereits beschriebenen eBlocks, wurden drei weitere eBlocks entwickelt. Das Eigenschaftsfenster eines Objektes der Klasse *Objekte löschen* ist mit dem in Abbildung 56 dargestellten Fenster identisch. Der einzige Unterschied ist, dass nur das Attribut *Target object* (Zielobjekt) verwendet wird. Beim Ausführen einer Instanz dieser Klasse wird das Objekt, dass im Attribut *Target object* verlinkt ist, und dessen hierarchisch untergeordneten Objekte, gelöscht. Die Klasse *Grayboxen auflösen* verwendet ebenfalls nur das Attribut *Target object (Zielobjekt)*. Wird ein Objekt dieser Klasse ausgeführt, löst es alle *Grayboxen (Platzhalter)*, die auf einem Plan, welcher mit dem Attribut *Target object* verlinkt ist, auf. Da nach der Ausführung dieses eBlocks die Objekte der Klasse *eine Baugruppe einkopieren* nicht mehr geändert werden können, wird der eBlock *Grayboxen auflösen* immer erst am Schluss des Konfigurationsprozesses ausgeführt. Auf der grafischen Benutzeroberfläche kann der User mittels einer Checkbox auswählen, ob dieser Vorgang durchgeführt werden soll.

Im Zuge der Entwicklung des *Metris Engineering Configurators* wurde die Klasse *massenhaft implementieren* ebenfalls entwickelt. Wie im Unterkapitel 5.3.1 beschrieben, enthält ein Objekt dieser Klasse zusätzliche Attribute. Die Attribute *Request (Anforderung)* und *Implementation (Implementierung)* sind in einer eigenen Kapitelkarte *Implementation* zu finden. Es können beliebig viele Paare von Anforderungen und Implementierungen zugewiesen werden. Beim Ausführen eines Objektes dieser Klasse werden die verlinkten Objekte des Attributes *Request* mit dem jeweils zugeordneten, im Attribut *Implementation* verlinkten Objekten, implementiert.

7 TESTS DER SOFTWARE

In diesem Kapitel werden die Methodik und die Ergebnisse der Tests, die an der neu entwickelten Software durchgeführt wurden, beschrieben. Die Vorgehensweise der Tests orientiert sich am V-Modell. Anders als bei der Entwicklung, wird bei der Testung mit der kleinsten Komponente begonnen, bis zum Schluss das gesamte System vom Kunden abgenommen wird. Die Theorie zur Methodik kann im Kapitel 4 nachgeschlagen werden.

7.1 Komponententest

Beim Komponententest werden die kleinsten Einheiten einer Software auf vier Hauptaspekte geprüft. Wie im Unterkapitel 4.1 beschrieben, handelt es sich hierbei um die Funktionalität, Robustheit, Effizienz und Wartbarkeit. Die Überprüfung der Funktionalität und der Robustheit kann als Black-Box-Test, die Überprüfung der Wartbarkeit als White-Box-Test eingestuft werden. Da die Prüfung der Effizienz sich mit den benötigten Rechnerressourcen beschäftigt und damit sowohl externe Größen, wie auch die interne Struktur der jeweiligen Komponente ausschlaggebend sind, kann sie nicht eindeutig zugeordnet werden. Wie in Abbildung 57 ersichtlich, werden unterschiedliche Tests von unterschiedlichen Stakeholdern durchgeführt. Da die Wartung der Software vom Team, dass auch für die Wartung der Engineering Software Comos verantwortlich ist, durchgeführt werden soll, werden alle Komponenten vom *Test User 4* auf deren Wartbarkeit überprüft. Die Aufgabe und das Wissensgebiet aller Stakeholder können im Unterkapitel 5.1.1 nachgelesen werden.

Legende: ja/nein - Test bestanden ja/nein

ID	Komponente	getestet von	getestet am	Funktionalität	Robustheit	Effizienz	Wartbarkeit	Anmerkungen
1	Vorlagenobjekt	Test User 3	03.07.2018	ja	ja	ja	ja/Test User 4	
2	Pfadmakro	Test User 3	03.07.2018	ja	ja	ja	ja/Test User 4	
3	Merge Objekt	Test User 3	03.07.2018	ja	ja	ja	ja/Test User 4	
4	Verknüpfungsplan	Test User 3	04.07.2018	ja	ja	ja	ja/Test User 4	
5	UND	Test User 3	04.07.2018	ja	ja	ja	ja/Test User 4	
6	ODER	Test User 3	04.07.2018	ja	ja	ja	ja/Test User 4	
7	Negation	Test User 3	04.07.2018	ja	ja	ja	ja/Test User 4	
8	Dependency	Test User 3	10.09.2018	ja	ja	ja	ja/Test User 4	
9	Condition	Test User 3	04.07.2018	ja	ja	ja	ja/Test User 4	
10	Action	Test User 3	04.07.2018	ja	ja	ja	ja/Test User 4	
11	Objekte löschen	PM / Entwickler	10.09.2018	ja	ja	ja/Test User 4	ja/Test User 4	
12	Objekte in Anlagenstruktur verschieben (Merge)	PM / Entwickler	04.07.2018	ja	ja	ja/Test User 4	ja/Test User 4	
13	massenhaft implementieren	PM / Entwickler	23.10.2018	ja	ja	ja/Test User 4	ja/Test User 4	
14	Grayboxen auflösen	PM / Entwickler	04.07.2018	ja	ja	ja/Test User 4	ja/Test User 4	
15	Platzhalter (Graybox)	Test User 3	04.07.2018	ja	ja	ja	ja/Test User 4	
16	eine Baugruppe einkopieren	PM / Entwickler	04.07.2018	ja	ja	ja/Test User 4	ja/Test User 4	
17	graphische Benutzeroberfläche	PM / Entwickler	11.09.2018	ja	ja/Test User 1	ja/Test User 4	ja/Test User 4	
18	WPF-Anwendung	PM / Entwickler	12.11.2018	ja	ja/Test User 1	ja/Test User 4	ja/Test User 4	für graphische Darstellung der XML-Datei

Abbildung 57: Komponententest, Quelle: Eigene Darstellung.

Die kleinste Einheit der neuen Software wird als Klasse bezeichnet. Somit entsprechen fast alle Komponenten, die in obiger Abbildung zu sehen sind, einer im UML-Klassendiagramm dargestellten Klasse. Dieses UML-Klassendiagramm wird im Unterkapitel 5.3.1 beschrieben. Die Bedeutung der einzelnen Klassen

wird im selben Unterkapitel erläutert. Nur die Komponenten mit der *ID 17* und *ID 18* sind nicht im Klassendiagramm zu finden, da sie die GUI betreffen. Die grafische Benutzeroberfläche im Comos (Komponente *17*) und die WPF-Anwendung (Komponente *18*) wurden im Zuge der Prüfung der Funktionalität gegenüber dem Präsentationsmodell validiert. Das Präsentationsmodell wird im Unterkapitel 5.3.4 erläutert. Die dritte Spalte *getestet von* gibt an, welcher Stakeholder für die Testung der jeweiligen Komponente verantwortlich ist. Ist in den Spalten *Funktionalität* bis *Wartbarkeit* ein weiterer Stakeholder notiert, bedeutet das, dass der Test von diesem durchgeführt wurde. Sind diese Spalten mit *ja* befüllt, bestand die Komponente den jeweiligen Test. In der vierten Spalte *getestet am* ist das Durchführungsdatum der Tests angegeben. Die Prüfung der Komponenten *1 – 10* und *15* wurde vom Test User 3 verantwortet. In Abbildung 57 ist ersichtlich, dass dieser all diese Komponenten auf deren Funktionalität, Robustheit und Effizienz testete. Außerdem ist ersichtlich, dass all diese Komponenten sämtliche Tests bestanden. Die Tests der restlichen Komponenten wurden vom *Projekt Manager/Entwickler* verantwortet. Die Funktionalität der Komponenten wurden von diesem auch selbst getestet. Test User 4 führte die Effizienz- und Wartbarkeitstests all dieser Komponenten durch. Vor allem Robustheitstests von Benutzeroberflächen sollten von Stakeholdern durchgeführt werden, die mit der Software nicht all zu gut vertraut sind. Da sie im Gegensatz zu Stakeholdern, die sich mit der Software intensiv beschäftigt hatten, andere Fehleingaben tätigen. Aus diesem Grund wurden die Komponenten *17* und *18* vom *Test User 1* auf deren *Robustheit* getestet.

Der Vergleich zwischen den in Abbildung 57 dargestellten Komponenten und dem UML-Klassendiagramm, welches im Unterkapitel 5.3.1 erläutert wird, zeigt, dass die Komponente *Baugruppen einkopieren – Vervielfältiger* nicht im Komponententest enthalten ist. Der Zweck der Klasse *Baugruppen einkopieren – Vervielfältiger* ist die Erfüllung der Anforderung *14 – Ein Projekt kann mehrere Pressen (Linien) enthalten* der in Abbildung 23 dargestellten Anforderungsliste. Wie im Unterkapitel 5.2.1 beschrieben, wird aufgrund der Komplexität und des späten Erstelldatums die Entwicklung dieser Anforderung und somit der erwähnten Klasse auf den Release 1.2 verschoben. Alle anderen Klassen wurden im Zuge des Komponententests überprüft. Die interne Fehlerfreiheit der einzelnen Komponenten wird mit dem Komponententest gewährleistet. Wie in Abbildung 57 illustriert, haben alle Komponenten diesen Test bestanden und so kann zur nächsten Teststufe übergegangen werden.

7.2 Integrationstest

Wie im Unterkapitel 4.2 beschrieben, ist der Integrationstest der zweite Test im V-Modell. Er stellt die reibungslose Interaktion der Komponenten untereinander, wie auch mit deren Umwelt sicher. Voraussetzung für den Integrationstest ist ein abgeschlossener Komponententest. So können interne Komponentenfehler von vornherein ausgeschlossen werden. Als Methodik kam die Ad-hoc Strategie, welche ebenfalls im Unterkapitel 4.2 erläutert wird, zum Einsatz.

In Abbildung 58 ist ersichtlich, dass die im Zuge des Integrationstests getesteten Komponenten den Komponenten des Komponententests sehr ähnlich sind. Die Komponenten *1 – 17* entsprechen den Komponenten *1 – 17* des Komponententests. Die Tests wurden auch von denselben Stakeholdern durchgeführt. Zeitlich wurde der Integrationstest kurz nach dem Komponententest der jeweiligen Komponente vollzogen. Die zweite Gruppe der Testobjekte, welche die Komponenten *18* und *19* umfasst, betrifft die externen Schnittstellen des Systems. Wie bei der Komponente *Excel-Schnittstelle* in der Spalte *Anmerkungen* ersichtlich,

wurde die *Excel-Schnittstelle* durch die *XML-Schnittstelle* ersetzt. Aufgrund der Nachvollziehbarkeit wurde dieser Punkt nicht gelöscht. Der Test der Komponente *19* wurde vom *Projekt Manager/Entwickler* verantwortet und durchgeführt. Die Schnittstellen der WPF-Anwendung (Komponente *20*) wurden auch vom *Projekt Manager/Entwickler* überprüft.

Legende: ja/nein - Test bestanden ja/nein

ID	Komponente	getestet von	getestet am	Test bestanden	Anmerkungen
1	Vorlagenobjekt	Test User 3	05.07.2018	ja	
2	Pfadmakro	Test User 3	05.07.2018	ja	
3	Merge Objekt	Test User 3	05.07.2018	ja	
4	Verknüpfungsplan	Test User 3	05.07.2018	ja	
5	UND	Test User 3	05.07.2018	ja	
6	ODER	Test User 3	05.07.2018	ja	
7	Negation	Test User 3	05.07.2018	ja	
8	Dependency	Test User 3	11.09.2018	ja	
9	Condition	Test User 3	05.07.2018	ja	
10	Action	Test User 3	05.07.2018	ja	
11	Objekte löschen	PM / Entwickler	11.09.2018	ja	
12	Objekte in Anlagenstruktur verschieben (Merge)	PM / Entwickler	05.07.2018	ja	
13	massenhaft implementieren	PM / Entwickler	23.10.2018	ja	
14	Grayboxen auflösen	PM / Entwickler	05.07.2018	ja	
15	Platzhalter (Graybox)	Test User 3	05.07.2018	ja	
16	eine Baugruppe einkopieren	PM / Entwickler	05.07.2018	ja	
17	graphische Benutzeroberfläche	PM / Entwickler	11.09.2018	ja	
18	Excel-Schnittstelle				ersetzt durch XML-Schnittstelle
19	XML-Schnittstelle	PM / Entwickler	23.10.2018	ja	
20	WPF-Anwendung	PM / Entwickler	12.11.2018	ja	für graphische Darstellung der XML-Datei
21	Software entspricht Aktivitätsdiagramm	Test User 3	12.11.2018	teilweise	*Anlagendokumentation ohne Comos erstellen* nicht möglich
22	Software entspricht Zustandsdiagramm	Test User 3	12.11.2018	teilweise	*Anlagendokumentation ohne Comos erstellen* nicht möglich
23	Software entspricht Use Case Diagramm	Test User 3	12.11.2018	ja	

Abbildung 58: Integrationstest, Quelle: Eigene Darstellung.

Eine weitere wichtige Grundlage für den Integrationstest sind die Aktivitäts-, Zustands- und Use-Case-Diagramme, welche in den Unterkapiteln 5.2.2, sowie 5.3.2 und 5.3.3 ersichtlich sind. Die Validierung der Software gegen diese drei UML-Diagramme betrifft die Komponenten *21, 22* und *23*. Die Komponenten *21* und *22* haben im Gegensatz zu allen anderen den Integrationstest nur *teilweise* bestanden. Grund dafür ist die Verschiebung der Entwicklung der *Erstellung der Anlagendokumentation ohne Comos zu öffnen*. Der Test dieser zwei Komponenten wird mit *Release 1.1* und *1.2* wiederholt. Weitere Informationen zu diesem Thema können im Unterkapitel 5.2.1 nachgelesen werden.

7.3 Systemtest

Der Systemtest ist der erste Test, der die neue Software aus Sicht des Kunden betrachtet. Die Theorie zum Systemtest wird im Unterkapitel 4.3 erläutert. Wie in nachfolgender Abbildung 59 illustriert, dient zum einen die Anforderungsliste als Grundlage für den Systemtest. Dieser Teil wird im folgenden Absatz beschrieben. Zum anderen wurde die Liste um die Anforderungen *16* bis *21* erweitert. Details zu diesen Anforderungen können ebenfalls dem Unterkapitel 4.3 entnommen werden. Sie werden im übernächsten Absatz erläutert.

Legende: ja/nein - Test bestanden ja/nein

ID	Anforderung	getestet von	getestet am	Test bestanden	Anmerkungen
1	Erstellung der Anlagendokumentation soll vollautomatisiert ablaufen (keine Nachbearbeitung nötig)	Test User 1	25.10.2018	teilweise	mit Filterpressenmodell vergleichen, mehrere Linien können nicht erstellt werden
2	einfache, fehlerfreie Bedienung -> Software muss Userfehler ausschließen	Test User 2	25.10.2018/ 13.11.2018	ja	im Comos und WPF-Anwendung, siehe Filterpressenmodell
3	Erstellung / Änderung der Templates soll einfach sein (von Key User nach kurzer Einschulung bedienbar)	Test User 3	13.11.2018	ja	
4	Konfiguration der Anlage über Excel-Interface oder ähnlich weit verbreitetes Tool	Test User 2	13.11.2018	ja	mit XML und WPF-Anwendung gelöst
5	Anlagendokumentation soll erstellt werden können, ohne Comos zu öffnen				über Enterprise Server, Umsetzung geplant in Release 1.1
6	Software soll für andere Anlagen/Produkte einfach erweiterbar sein	Test User 3	13.11.2018	ja	
7	soll mit Comos V10.1 lauffähig sein	PM / Entwickler	22.06.2018	ja	
8	automatisches Updaten + Revisionieren der Dokumente bei Änderung der Konfiguration				
9	Prozessmodell der Software soll erstellt werden	Auftraggeber	08.08.2018	ja	siehe Aktivitätsdiagramm
10	offenes Interface (XML)	Auftraggeber	23.10.2018	ja	
11	mit Software erstellte Anlagen müssen wie manuell erstellte Projekte im Comos modifizierbar sein (kopierbar, einzelne Teile verschieben löschen etc.)	Test User 2	25.10.2018	ja	
12	Aktionen rückgängig machen	PM / Entwickler	25.10.2018	ja	
13	verschiedene Versionen in Verkaufsphase (Working Layer), in Engineering Phase wird ein Working Layer freigegeben	Test User 1	25.10.2018	ja	
14	Ein Projekt kann mehrere Pressen (Linien) enthalten - muss im Konfigurator auswählbar sein				Umsetzung geplant in Release 1.2
15	Lieferumfang muss im Excel oder ähnlichem Tool auswählbar sein (eigene Gruppen) -> Kapitelkarte "Scope of Supply" und Farbe am P&ID ändern sich				Umsetzung geplant in Release 1.1
16	Lasttest	Test User 2	13.11.2018	ja	
17	Zugriffsberechtigungen	PM / Entwickler	13.11.2018	ja	
18	Robustheit	Test User 2	13.11.2018	ja	
19	Kompatibilität (GUI, Interaktion mit anderen Systemen)	Test User 3	13.11.2018	ja	
20	Dokumentation	Test User 2	14.11.2018	ja	
21	Änderbarkeit	Test User 4	12.11.2018	ja	

Abbildung 59: Systemtest, Quelle: Eigene Darstellung.

Die Anforderung mit der ID *1* wurde vom *Test User 1* am *25.10.2018* getestet und aufgrund der teilweisen Übereinstimmung mit dem Filterpressenmodell (*mehrere Linien können nicht erstellt werden*), welche im Unterkapitel 6.2.2 erläutert wird, nur teilweise erfüllt. Die Prüfung aller Anforderungen, bei denen der Prüfer mit der Software und einer Filterpressenanlage nicht so sehr vertraut sein soll, wurden vom *Test User 2* durchgeführt. Das betrifft die Anforderungen *2, 4* und *11*. Die *einfache und fehlerfreie Bedienung* wurde sowohl bei der grafischen Benutzeroberfläche im Comos, wie auch bei der WPF-Anwendung getestet. Wie bereits im Unterkapitel 5.2.2 beschrieben, wurde die Excel-Schnittstelle durch eine XML-Schnittstelle und eine WPF-Anwendung ersetzt. Das ist in den *Anmerkungen* der Anforderung *4* dokumentiert. Die Anforderungen *3* und *6* betreffen die Änderung der Templates und die Erweiterung des Einsatzbereiches der Software. Soll die Software für andere Produkte/Anlagen verwendet werden, müssen Templates geändert bzw. neu erstellt werden. Laut Use-Case-Diagramm kann ein Comos Key-User diese Tätigkeit durchführen. Somit macht es Sinn, diese Anforderungen vom *Test User 3* prüfen zu lassen. In der Stakeholderliste ist ersichtlich, dass dieser einem Comos Key-User entspricht. Da die Anforderung *10* vom *Auftraggeber* eingestellt wurde und er die nötigen Kompetenzen im Bereich von XML besitzt, wurde sie auch von diesem überprüft. Das *Prozessmodell der Software* wurde vom *Auftraggeber* als sehr nützlich empfunden und somit auch von diesem getestet. Das UML-Aktivitätsdiagramm, welches im Unterkapitel 5.3.2 beschrieben wird, entspricht einem Prozessmodell. Die Prüfung der allgemeinen Anforderungen *7* und *12*, stellt keine speziellen Ansprüche. Somit wurde sie vom *Projekt Manager/Entwickler* durchgeführt. Alle beschriebenen Anforderungen, außer der Anforderung *1,* haben die Tests bestanden.

Die Anforderungen *16* bis *21* sind nicht in der Anforderungsliste enthalten, sondern stellen allgemeine Anforderungen an eine Software. Beim *Lasttest* wird die Software an ihre Leistungsgrenzen gebracht. Die Prüfung der Reaktion auf Falscheingaben erfolgt im Zuge der Testung der *Robustheit*. Diese Tests können aufgrund der fehlenden Vertrautheit des *Test Users 2* mit der Software von diesem am besten durchgeführt werden. Auch die Prüfung der Dokumentation wurde aus diesem Grund vom *Test User 2* durchgeführt. Da die Software nicht im Dauerbetrieb zum Einsatz kommt, entfällt der Test auf Stabilität des Systems. Ein weiterer wichtiger Aspekt sind die *Zugriffsberechtigungen* der unterschiedlichen User. So soll beispielsweise ein *Comos User* den Source Code nicht verändern können. Diese Anforderung wurde vom *Projekt Manager/Entwickler* erfolgreich getestet. Bei der Testung der Anforderung *19* wurde neben der Testung der *Kompatibilität* gegenüber benachbarten Systemen auch die GUI nochmals überprüft. Die letzte Anforderung der Abbildung 59 kann nur von einer Person mit einem fundierten Wissen im Bereich Comos und IT-Landschaft durchgeführt werden. Aufgrund dessen wurde dieser Test vom Test *User 4* durchgeführt. Wie in Abbildung 59 ersichtlich, entspricht die Software den Anforderungen *16* bis *21*.

7.4 Abnahmetest

Der Abnahmetest ist der letzte Test im V-Modell. Er wird unmittelbar vor der Inbetriebnahme der neuen Software vom Kunden bzw. Endanwender durchgeführt. Wie im Unterkapitel 4.4 erläutert, gibt es üblicherweise vier verschiedene Formen von Abnahmetests. Drei davon werden in nachfolgender Abbildung 60 behandelt. Der Alpha-Test wird nicht explizit erwähnt, da die Erstellung der Dokumentation einer Filterpressenanlage bereits Teil der unterschiedlichen Testschritte war und somit die Software bereits unter realen Bedingungen getestet wurde. Da die Anzahl der User vorerst begrenzt ist, kann die Phase zwischen dem Release 1.0 und dem Release 1.1 als Beta-Testphase gesehen werden.

Legende: ja/nein - Test bestanden ja/nein

ID	Anforderung	getestet von	getestet am	Test bestanden	Anmerkungen
1	Test auf Akzeptanz des Kunden	Kunde	15.11.2018	teilweise	Anforderungen 5, 8, 14, 15 müssen im Zuge des Release 1.1 bzw. 1.2 erfüllt werden
2	Test auf Benutzerakzeptanz	Test User 1	15.11.2018	teilweise	Anforderungen 14, 15 müssen im Zuge des Release 1.1 bzw. 1.2 erfüllt werden
3	Test auf Akzeptanz der Systembetreiber	Test User 4	15.11.2018	ja	

Abbildung 60: Abnahmetest, Quelle: Eigene Darstellung.

Verläuft der *Test auf Akzeptanz des Kunden* positiv, bedeutet das, dass der Kunde die Software in dieser Form akzeptiert und alle vertraglichen Rahmenbedingungen, soweit vorhanden, erfüllt sind. Im Falle dieser Masterarbeit ersetzt die Anforderungsliste den Vertrag. Da die *Anforderungen 5, 8, 14* und *15* im Zuge des *Release 1.1* bzw. *1.2* erfüllt werden, wurde die Software vom Kunden teilweise abgenommen. Dasselbe gilt für den *Test auf Benutzerakzeptanz*. Dieser wurde vom *Test User 1*, welcher später einer der Hauptanwender der neuen Software sein wird, durchgeführt. Für ihn spielen die *Anforderungen 5* und *8* keine wesentliche Rolle, daher würde er die Software nach Erfüllung der *Anforderungen 14* und *15* vollständig akzeptieren. Der Test auf *Akzeptanz der Systembetreiber* wurde vom *Test User 4*, welcher die Funktion *Comos Administrator* inne hat und die Software auch warten wird, erfolgreich durchgeführt. Zusätzlich zu

der in Abbildung 60 ersichtlichen Tabelle wurde pro Akzeptanztest ein Protokoll erstellt. Neben den oben angeführten Punkten enthält es Unterschriften der Stakeholder, die die Tests durchführten.

7.5 Zusammenfassung der Tests

Im Zuge des Komponenten-, Integrations-, System- und Abnahmetests wurde zum einen die Software *Metris Engineering Configurator* an sich getestet. An dieser Stelle wird überprüft ob die Software dem UML-Klassendiagramm und der Anforderungsliste entspricht. Zum anderen wurden die Templates der Filterpressenanlage, welche im Unterkapitel 6.3 beschrieben sind, mit dem Modell der Filterpressenanlage verglichen, welches im Unterkapitel 6.2.2 erläutert wird. Der Großteil der Tests wurde jedoch mit dem *Metris Engineering Configurator* in Kombination mit der Filterpressenanlage durchgeführt. Diese dritte Testart betrifft vor allem die Verifizierung gegenüber dem UML-Aktivitätsdiagramm, dem UML-Zustandsdiagramm und dem Präsentationsmodell. In einem weiteren Testschritt wurden die neue Software und die Templates der Filterpressenanlage dem Use-Case-Diagramm gegenübergestellt. Wie in den Unterkapiteln 7.1 bis 7.3 erläutert, entspricht sowohl der *Metris Engineering Configurator*, wie auch die Templates der Filterpressenanlage dem Großteil der unterschiedlichen Modelle. Ebenso werden die meisten aller Anforderungen, welche in der Anforderungsliste im Unterkapitel 5.2.1 angeführt sind, erfüllt. Vor dem Abnahmetest wurde überprüft, ob die im Projektauftrag angeführten Ziele erreicht wurden. Der Projektauftrag wird im Unterkapitel 5.1.2 beschrieben. Außer dem Ziel *3 – die Anlage kann im Comos oder über ein Excel-Interface vollständig konfiguriert werden*, wurden alle Ziele vollständig erreicht. Das Excel-Interface wurde durch eine XML-Schnittstelle und eine WPF-Anwendung ersetzt. Somit wurde dieses Ziel zwar nicht explizit erreicht, jedoch kann die Filterpressenanlage auch von Personen ohne Comos Zugang konfiguriert werden.

8 ERSTELLUNG DER DOKUMENTATION EINER FILTERPRESSENANLAGE

Um die Vor- und Nachteile bei der Erstellung einer Anlagendokumentation zwischen der konventionellen Vorgangsweise und unter Einsatz des *Metris Engineering Configurators* ermitteln zu können, werden in diesem Kapitel die Unterschiede beim Beispiel einer Filterpressenanlage aufgezeigt. Damit die neue Software die Anlagendokumentation voll automatisiert erstellen kann, musste die Filterpressenanlage vollständig modularisiert und für den Einsatz des *Metris Engineering Configurators* aufbereitet werden. Die Ergebnisse der Modularisierung und dessen Umsetzung im Engineering Programm Comos werden im Kapitel 6 beschrieben.

8.1 Konventionelle Vorgangsweise

Bei der konventionellen Vorgehensweise der Erstellung der Anlagendokumentation gibt es im Wesentlichen drei unterschiedliche Arten. Bei der ersten Vorgehensweise wird ein Projekt mit allen Objekten und Dokumenten neu erstellt. Typischerweise wird das von mehreren Abteilungen gemeinsam erledigt. Als Erstes werden die Struktur-Objekte inklusive der Funktionen von der Abteilung *Process Engineering* erstellt. Diese erstellt auch die P&IDs. Die Abteilung *Basic Engineering - Automation* versorgt danach die Sensoren und Aktoren mit technischen Daten und definiert eventuell den Hersteller und die Type des jeweiligen Gerätes. In dieser Phase werden unter anderem Motor-, Ventil- und Instrumentenlisten erzeugt. Im Anschluss werden vom *Detail Engineering - Automation* Stromlaufpläne erstellt. Die jeweilige Abteilung kann erst mit der Arbeit beginnen, wenn die vorgelagerten Dokumente nahezu vollständig sind. Die Abteilung *Process Engineering* oder die Abteilung *Basic Engineering - Automation* ist auch für die Erstellung der Funktionspläne zuständig. Diese Art des Engineerings ist am Aufwendigsten und wird nur verwendet, wenn noch kein ähnliches Projekt vorhanden ist.

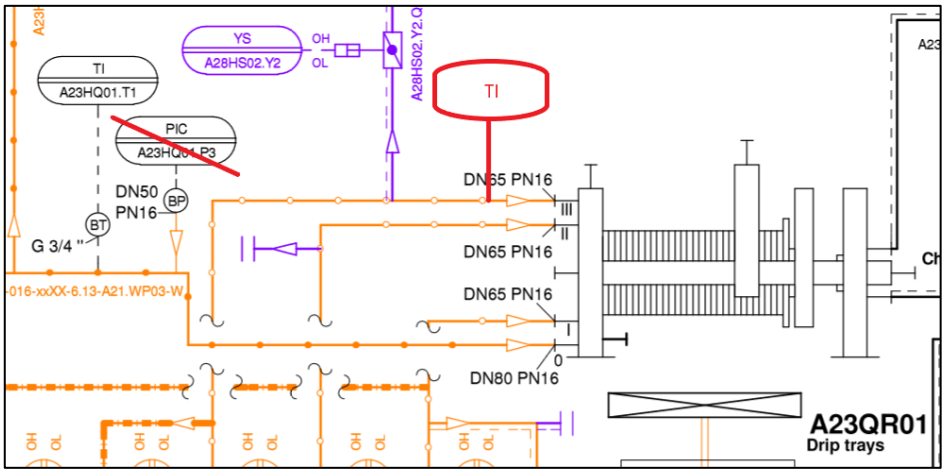


Abbildung 61: bearbeitetes P&ID, Quelle: Eigene Darstellung.

Bei der zweiten Methode wird ein ähnliches Projekt kopiert und abgeändert. Es sind wiederum dieselben Abteilungen, wie bei der ersten Methode beteiligt. Jeder ändert die für ihn relevanten Objekte und Dokumente. In Abbildung 61 ist eine typische Änderung skizziert. Eine Funktion wird gelöscht und eine andere wird hinzugefügt. Wiederum sind die Abteilungen voneinander abhängig. Beispielsweise können die

Motoren von der Abteilung *Basic Engineering - Automation* nicht ausgelegt werden, wenn noch nicht definiert ist, welche eigentlich benötigt werden. Die dritte Vorgehensweise kommt vor allem bei standardisierten und modularisierten Anlagen zum Einsatz. Die Module sind in einem Template-Projekt gespeichert und können von diesem in das Engineering-Projekt kopiert werden. Da diese Methode außer bei sehr kleinen Maschinen eine Nachbearbeitung erfordert und die Wartung der Templates auch ein erheblicher Aufwand ist, wird auch für standardisierte Anlagen öfters die zweite Methode gewählt. Filterpressenanlagen werden bis jetzt mit der dritten Methode erstellt.

8.2 Verwendung der neuen Software

Die Vorgehensweise bei der Erstellung der Anlagendokumentation einer Filterpressenanlage mithilfe der neu entwickelten Software wird im Folgenden erläutert. Wurde das Engineering Programm Comos gestartet und das Projekt, in dem die Dokumente erstellt werden sollen, geöffnet, kann der *Metris Engineering Configurator* durch Klicken auf die Schaltfläche, welche in Abbildung 62 ersichtlich ist, gestartet werden.

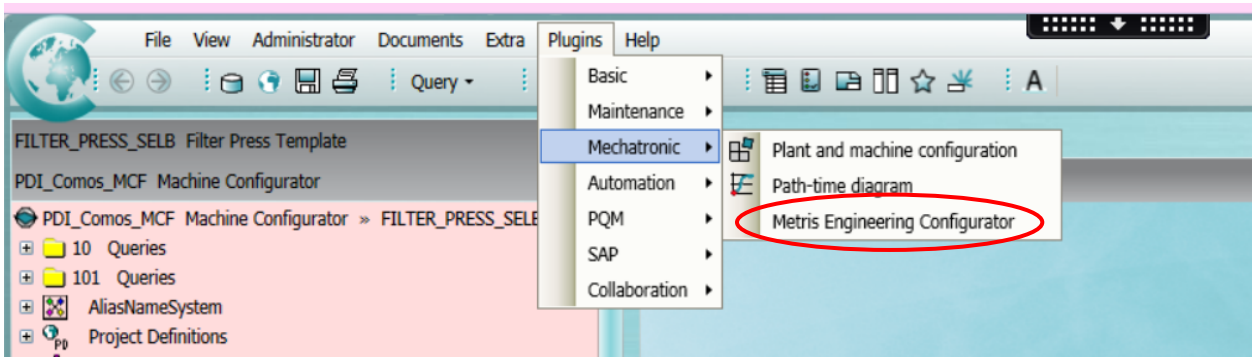


Abbildung 62: Schaltfläche *Metris Engineering Configurator* starten, Quelle: Eigene Darstellung.

Im nächsten Schritt kann durch Klicken auf die Schaltfläche *Select template*, welche in Abbildung 65 zu sehen ist, das Fenster zur Auswahl des Templates geöffnet werden. Neben einem Template kann auch eine bestehende Konfiguration ausgewählt werden. Wird ein Template ausgewählt, erstellt der *Metris Engineering Configurator* eine neue Anlage. Bei Auswahl einer bestehenden Konfiguration wird diese aufgerufen und kann danach verändert werden.

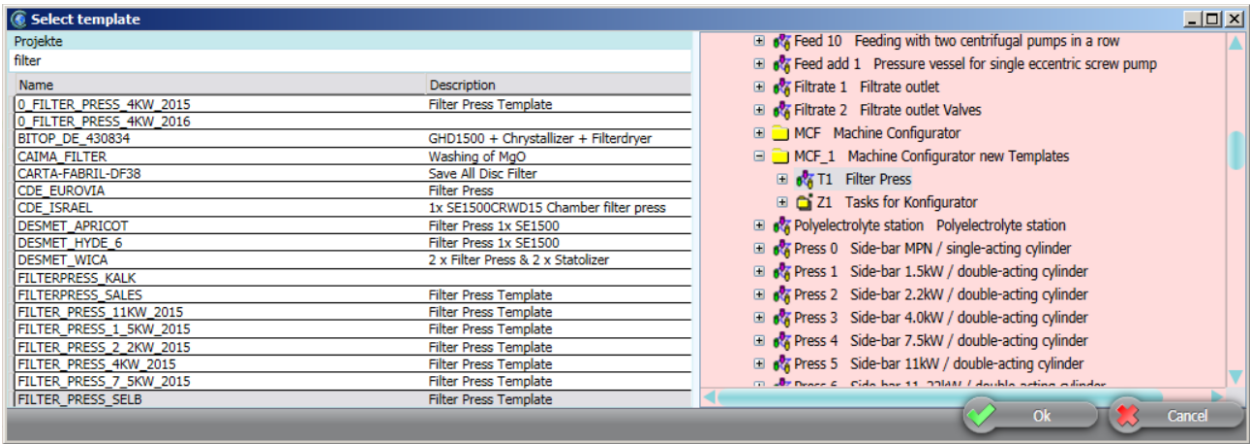


Abbildung 63: Fenster *Select template*, Quelle: Eigene Darstellung.

In Abbildung 63 ist das Fenster, in dem die Templates und Konfigurationen ausgewählt werden können, illustriert. In diesem Beispiel wird ein Template markiert. Mit Klicken auf die Schaltfläche *OK* wird die Auswahl übernommen und das Fenster schließt sich. Vergleicht man dieses Fenster mit dem Präsentationsmodell, ist zu sehen, dass ein Suchfeld im linken oberen Eck hinzugefügt wurde. Mit diesem wird die Auswahl des gewünschten Projektes oder Template Projektes erleichtert. In obiger Abbildung wurde nach dem Wort *filter* gesucht.

Nach Auswahl eines Templates oder einer Konfiguration erscheinen die zu konfigurierenden Objekte im Hauptfenster. In Abbildung 64 ist eine Beispielkonfiguration einer Filterpressenanlage dargestellt. Im rot markierten Bereich sind die obligatorischen und optionalen Anlagenteile ersichtlich. Die optionalen Anlagenteile können durch die Checkboxen ausgewählt werden. Alle Anlagenteile sind im Modell der Filterpressenanlage, welche im Unterkapitel 6.2.2 beschrieben wird, zu finden. Die Anlagenteile, welche versetzt angezeigt werden, wie z.B. *Cake drying* können nur ausgewählt werden, wenn der übergeordnete Anlagenteil, in diesem Fall *Filtrate Outlet with Valves,* selektiert wurde. In diesem Beispiel ist das nicht Fall, daher sind die versetzten Anlagenteile *Cake drying* und *Cake washing* grau dargestellt. Somit wurde die *requires*-Beziehung umgesetzt. Wird ein optionaler Anlagenteil, welcher in einer *exclude*-Beziehung zu einem anderen optionalen, selektierten Anlagenteil steht, ausgewählt, wird der bereits selektierte Anlagenteil automatisch abgewählt. Mit dieser Methode wird die Auswahl zweier sich ausschließender Anlagenteile verhindert.

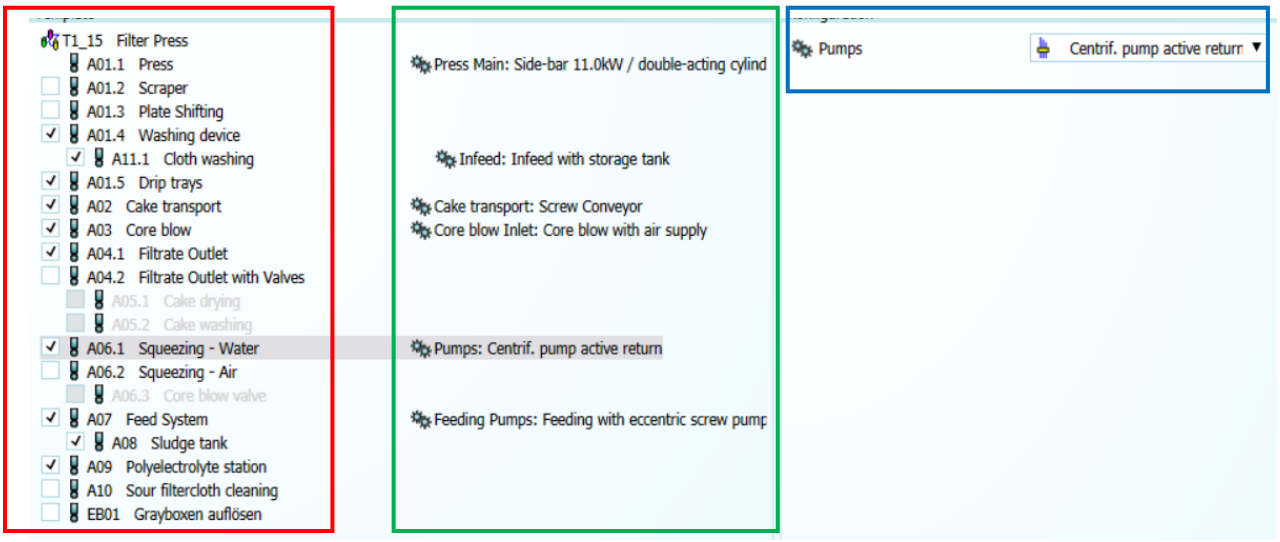


Abbildung 64: Konfiguration der Anlage im Comos, Quelle: Eigene Darstellung.

Im blau markierten Bereich können die unterschiedlichen Varianten der Variation Points ausgewählt werden. Die selektierten Varianten werden zugehörig zu jedem Anlagenteil im grün markierten Bereich übersichtlich dargestellt. Beispielsweise wurde der Anlagenteil *Squeezing – Water markiert*. Im blau markierten Bereich wurde die Variante *Centrif. pump active return* gewählt. Die obige Abbildung entspricht dem Präsentationsmodell, welches in Abbildung 37 ersichtlich ist. Durch Klicken auf die Schaltflächen *Execute* oder *Apply*, welche in Abbildung 64 nicht ersichtlich sind, wird der automatisierte Prozess zur Erstellung der Anlagendokumentation gestartet.

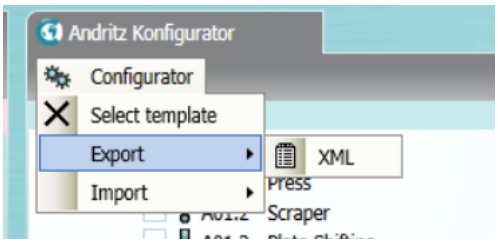


Abbildung 65: *Metris Engineering Configurater* – XML-Export/-Import, Quelle: Eigene Darstellung.

Soll die Anlage nicht im Comos direkt, sondern extern konfiguriert werden, muss das Konfigurationsfile nach Auswahl des Templates bzw. der Konfiguration, in Form eines XML-Files, exportiert werden. Die benötigten Schaltflächen sind in Abbildung 65 dargestellt. Wurde die WPF-Anwendung gestartet und das XML-File über die in Abbildung 66 rot markierte Schaltfläche geladen, wird das Konfigurationsfenster geöffnet. Wie in Abbildung 66 zu sehen, ist es dem Konfigurationsfenster des Comos sehr ähnlich. Die Optionen können wieder mittels Checkboxen selektiert und die unterschiedlichen Varianten im rechten Bereich des Fensters ausgewählt werden. Wurde die Anlage vollständig konfiguriert, werden mit Klicken auf die Schaltfläche *Apply* die Daten im XML-File gespeichert. Die Schaltfläche *Close* schließt die Anwendung, ohne die Auswahl in das XML-File zu übernehmen. Nach durchgeführter Konfiguration mittels der WPF-Anwendung, wird das XML-File wieder in das Comos importiert. Die Schaltfläche für den Import des XML-Files ist in Abbildung 65 ersichtlich.

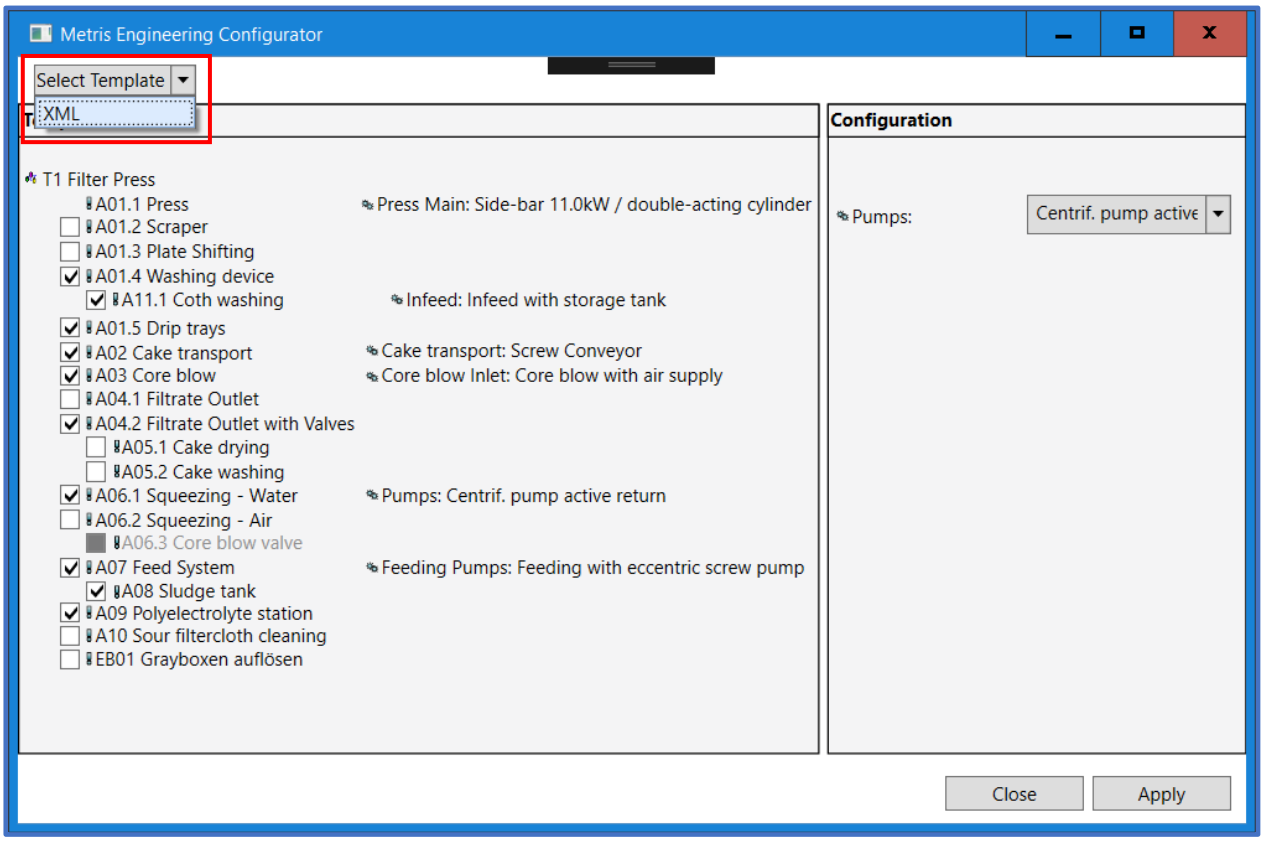


Abbildung 66: Konfiguration der Anlage mittels WPF-Anwendung, Quelle: Eigene Darstellung.

Die nachfolgende Abbildung zeigt einen Auszug der Objekte, die vom *Metris Engineering Configurator* erstellt wurden. Zum einen ist hier das erstellte P&ID dargestellt, zum anderen unterschiedliche

Strukturobjekte und Equipments. Unter den Equipments sind weitere Objekte wie Funktionen, Motoren und Instrumente zu finden. Die Funktionspläne und die Stromlaufpläne wurden unter den Funktionen erstellt.

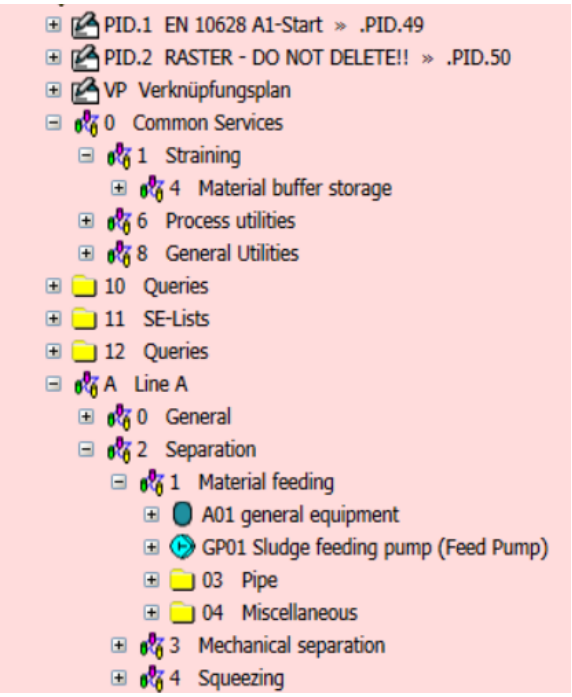


Abbildung 67: vom *Metris Engineering Configurator* erstellte Objekte, Quelle: Eigene Darstellung.

Das wichtigste Dokument, welches im Comos erstellt wird, ist das P&ID. Auf diesem baut das weitere Engineering und somit auch alle weiteren Dokumente auf. Auf dem P&ID sind alle Rohrleitungen und die wichtigsten Instrumente und Aktuatoren dargestellt. Des Weiteren sind auch Tanks, Pumpen und ähnliche Geräte ersichtlich. In Abbildung 68 ist das P&ID, welches vom *Metris Engineering Configurator* mit obiger Konfiguration erstellt wurde, illustriert. Die Kernkomponente für den Filterprozess umfasst die Filterpresse mit den optionalen Komponenten *Washing device* und *Drip trays* (rot markiert). Das *Washing device* wird von der Komponente *Cloth washing,* die auf diesem Auszug nicht ersichtlich ist, mit Waschwasser versorgt. Für den Abtransport des Filterkuchens wurde ein *Screw Conveyor* gewählt. Dieser ist in Abbildung 68 blau markiert. Der grün markierte Bereich besteht aus zwei Equipments, dem Wassertank (*Water storage*) und der Kreiselpumpe für die automatische Rückführung (*Centrif. pump active return*). Diese zwei Equipments wurden als *Squeezing – Water* bezeichnet und sind für das Nachdrücken der Membranen mit Hilfe von Wasser zuständig.

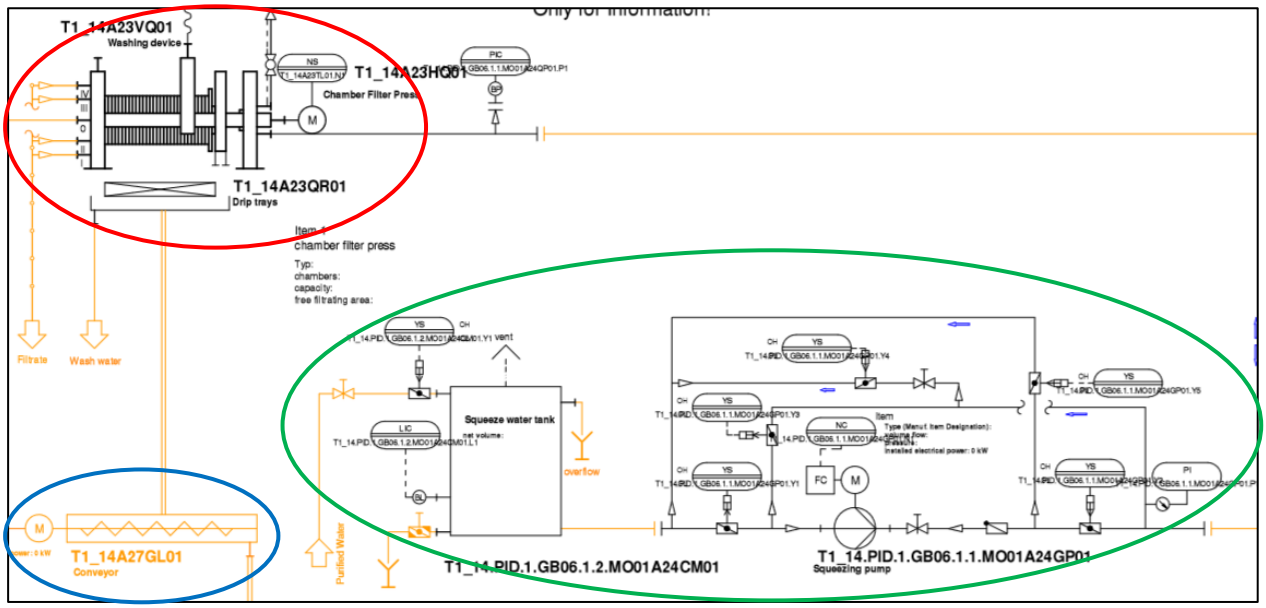


Abbildung 68: vom *Metris Engineering Configurator* erstelltes P&ID – Auszug, Quelle: Eigene Darstellung.

In Abbildung 69 ist der Stromlaufplan des Motors der Hydraulikpumpe, die auch im P&ID in Abbildung 68 ersichtlich ist, dargestellt. Wie zu sehen, wurde er vom *Metris Engineering Configurator* mit allen notwendigen Informationen angelegt. So ist zu sehen, dass der Motor über ein Schütz *-QA1* ein- und ausgeschaltet und vom Motorschutzschalter *-FC1* gegen Überlast und Kurzschluss geschützt wird. Darüber hinaus ist das Versorgungskabel für den Motor zu sehen.

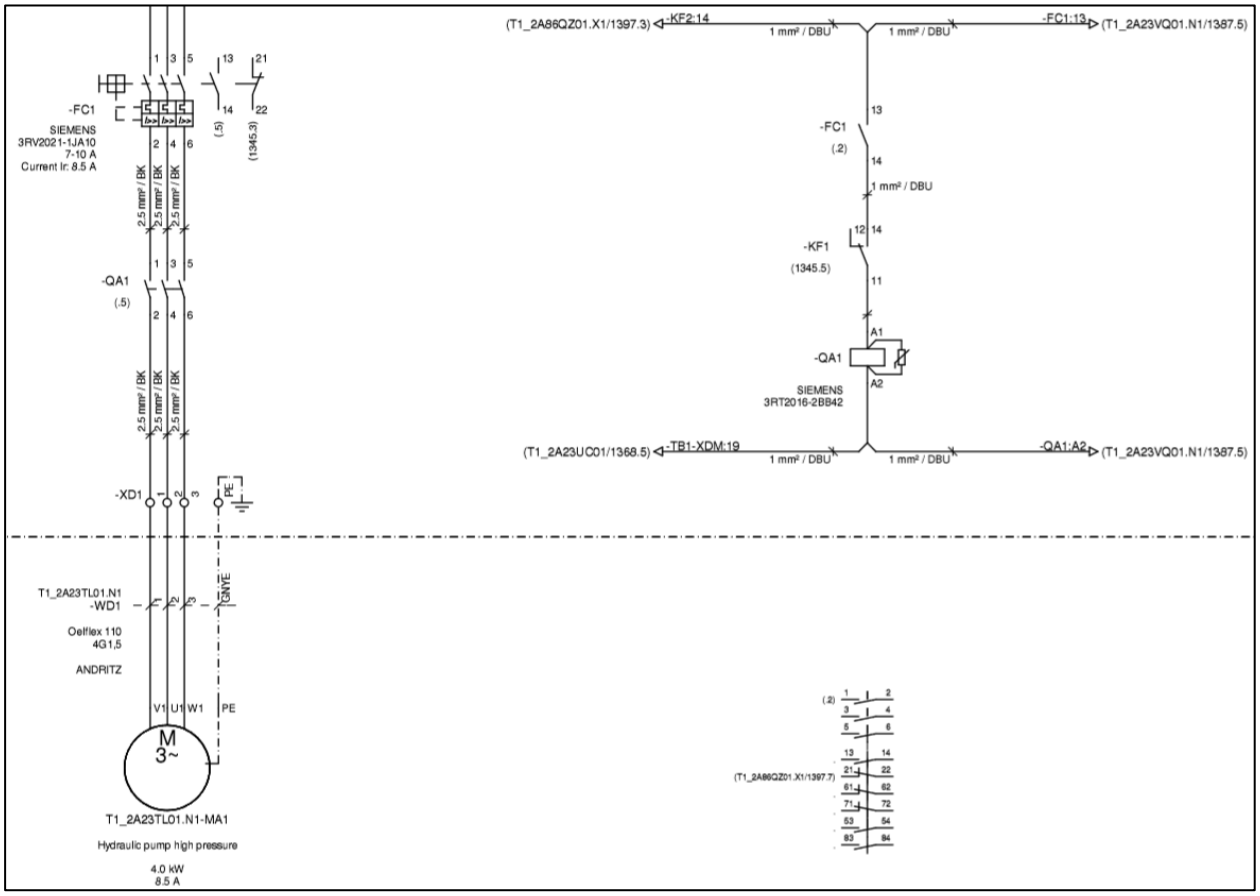


Abbildung 69: vom *Metris Engineering Configurator* erstellter Stromlaufplan – Auszug, Quelle: Eigene Darstellung.

Um den Fluss von Hydraulikflüssigkeiten im Engineering Programm Comos darstellen zu können, werden Hydraulikpläne, wie in Abbildung 70 ersichtlich, erstellt. Auf diesem werden alle für das Hydrauliksystem relevanten Komponenten wie z.B. Wegeventile (rot markiert), Überdruckventile (blau markiert), Filter (grün markiert) und Pumpen (orange markiert) dargestellt. Der Motor dieser Pumpe entspricht dem in Abbildung 69 illustriertem Motor. Der unten dargestellte Hydraulikplan wurde ebenfalls vom *Metris Engineering Configurator* vollständig erstellt und muss nicht nachbearbeitet werden.

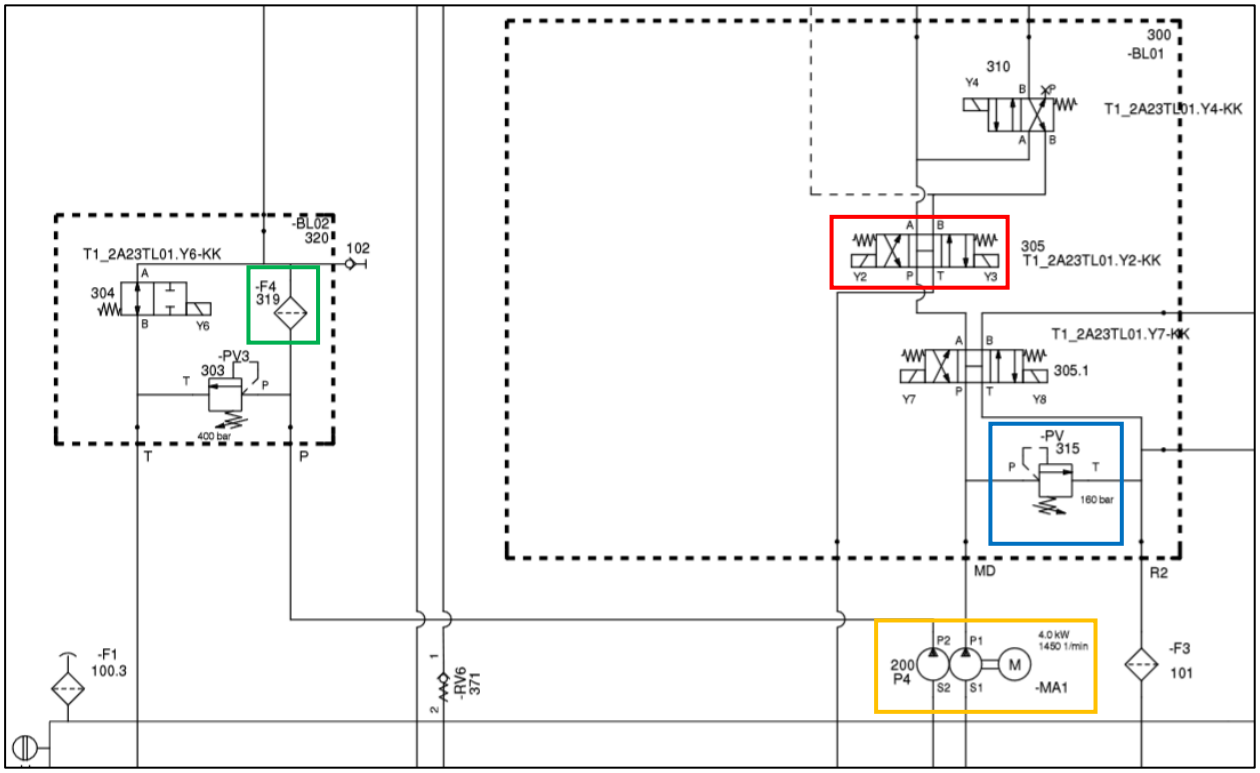


Abbildung 70: vom *Metris Engineering Configurator* erstellter Hydraulikplan – Auszug, Quelle: Eigene Darstellung.

Mit der neuen Software *Metris Engineering Configurator* wurde die Erstellung der Anlagendokumentation einer Filterpressenanlage automatisiert. Die Vorteile, die sich daraus ergeben, werden im folgenden Kapitel näher erläutert.

9 FAZIT

In diesem Kapitel werden die bereits im Detail beschriebenen Ergebnisse zusammengefasst und die konventionelle Erstellung der Dokumentation einer Filterpressenanlage mit dem Ergebnis unter Einsatz der neuen Software verglichen. An dem im Kapitel 8 gezeigten Beispiel werden die Vorteile, die sich durch die Verwendung des *Metris Engineering Configurators* ergeben, beschrieben. Ebenso wird in diesem Kapitel 9.1 ein Resümee über die gesamte Masterarbeit gezogen. Zum Schluss werden die daraus abgeleiteten Handlungsempfehlungen präsentiert.

Der erste Teil des Ergebnisses dieser Masterarbeit erstreckt sich von der Erhebung der Anforderungen, über die modellbasierte Detaillierung dieser, bis hin zu den Tests an der neu entwickelten Software. Dieser gesamte Entwicklungsprozess orientiert sich am V-Modell und wird hauptsächlich im Kapitel 5 und Kapitel 7 behandelt. Der zweite Teil des Ergebnisses betrifft die Anwendung der neu entwickelten Software an einem konkreten Beispiel, der Filterpressenanlage. Somit kann zum einen die praktische Anwendbarkeit des *Metris Engineering Configurators* getestet und zum anderen die Vor- und Nachteile gegenüber der konventionellen Vorgehensweise bei der Erstellung der Anlagendokumentation erhoben werden. Dieser zweite Teil des Ergebnisses erstreckt sich über das Kapitel 6 und 8. Im Kapitel 6 wird die Modularisierung der Filterpressenanlage und deren Umsetzung im Engineering Programm Comos beschrieben. Die Erstellung der Dokumentation der Filterpressenanlage mit dem *Metris Engineering Configurator* und mit der konventionellen Vorgangsweise wird im Kapitel 8 erläutert. Die Ergebnisse des Vergleiches der beiden Vorgangsweisen wird im Folgenden beschrieben.

9.1 Conclusio

In diesem Unterkapitel wird die Schlussfolgerung aus dieser Masterarbeit gezogen. Im vorherigen Kapitel werden die konventionelle Vorgangsweise und die Vorgangsweise unter Einsatz der im Zuge dieser Masterarbeit erstellten Software zur Erstellung der Anlagendokumentation, erläutert. Wie diesem Ergebnis zu entnehmen ist, wurde somit die manuelle Erstellung der Dokumente automatisiert. Daraus ergeben sich drei wesentliche Vorteile. Erstens kann die Zeit, die für das Engineering einer konfigurierbaren Anlage benötigt wird, auf wenige Stunden oder Minuten reduziert werden. Die einzige verbleibende manuelle Arbeit ist die Konfiguration der Anlage mithilfe der grafischen Oberfläche bzw. der Import und Export des XML-Files. Im Gegensatz dazu muss bei allen drei konventionellen Vorgangsweisen, die im Unterkapitel 8.1 beschrieben werden, zumindest ein Teil des Engineerings manuell ausgeführt werden.

Bis jetzt wurde eine Filterpressenanlage mit der dritten konventionellen Vorgehensweise geplant. Mit dieser Methode, bei der die Anlage vollständig modularisiert ist und Templates vorhanden sind, benötigt die Erstellung der Dokumente wie P&ID, Stromlaufpläne, Funktionspläne und diverser Listen für eine Filterpressenanlage ca. zwei bis drei Wochen. Unter Einsatz des *Metris Engineering Configurators* ist die fertige Dokumentation innerhalb weniger Stunden verfügbar. Somit ergibt sich eine Einsparung von ca. 90% des Engineering Budgets. Ein zusätzlicher einmaliger Aufwand ergibt sich durch die Erstellung der Templates, welche zur neu entwickelten Software kompatibel sein müssen. Für die Erstellung dieser Templates für die Filterpressenanlage wurden ca. zwei Wochen benötigt. Aufgrund dieses hohen Aufwandes ist die Verwendung des *Metris Engineering Configurators* für einmalig verkaufte Anlagen nicht wirtschaftlich. Für Anlagen, die in ähnlicher Weise mehrmals verkauft werden, kann die neue Software zielbringend eingesetzt werden.

Dafür müssen die Anlagen modularisiert und standardisiert werden. Auch wenn aufgrund der großen Variabilität nicht alle Varianten in den Templates abgebildet werden können, wird die benötigte Planungszeit für den standardisierten Teil um ein Vielfaches verringert. Die nicht abgebildeten Teile können nach Anwendung des *Metris Engineering Configurators* nach wie vor manuell hinzugefügt werden.

Neben der Verringerung der Planungskosten ist die Verringerung der Durchlaufzeiten ein wesentlicher Vorteil. Vor allem bei Maschinen und kleinen Anlagen, die in hohem Maße standardisiert und modularisiert sind, erwartet sich der Kunde eine kurze Lieferzeit. Wie im Unterkapitel 8.1 erwähnt, sind üblicherweise mehrere Abteilungen an der Planung einer Maschine oder Anlage beteiligt. Meistens kann eine bestimmte Abteilung erst mit der Planung beginnen, wenn die im Planungsprozess vorgereihte Abteilung die Arbeit großteils abgeschlossen hat. Unter Verwendung der neu entwickelten Software, wird dieser Umstand eliminiert. Sind mehrere Abteilungen in der Planung involviert, entstehen zwangsläufig Schnittstellen. Schnittstellen bergen die Gefahr des Informationsverlustes. Beispielsweise erhält ein Mitarbeiter, der in einer späteren Phase der Planung tätig ist, eine für ihn wichtige Information nicht, da sie von einer im Planungsprozess vorgereihten Abteilung nicht weitergegeben wurde. Der dritte wesentliche Vorteil, der sich durch den Einsatz des *Metris Engineering Configurators* ergibt, ist der Wegfall dieses Risikos. Mithilfe der XML-Schnittstelle ist es möglich, dass der Verkäufer / die Verkäuferin bereits im Verkaufsgespräch eine Anlage konfiguriert und das P&ID als Gesprächsbasis dient.

Die Vorgehensweise in dieser Masterarbeit hat sich insofern bewährt, da mithilfe der unterschiedlichen Modelle frühzeitig Inkonsistenzen in den Anforderungen beseitigt werden konnten. Ebenso wurden fehlende Anforderungen frühzeitig entdeckt. Auch die strukturierte Testung der neuen Software war zielführend. So konnten Fehler, die üblicherweise erst in der Betriebsphase der Software entdeckt werden, bereits in der Testphase korrigiert werden. Bei der Entwicklung ähnlicher Produkte wird in der Praxis das Requirements Engineering oftmals vernachlässigt. Jedoch kann mit einem gut ausgeführten Requirements Engineering das Risiko der Auslieferung einer Software, die nicht den Anforderungen der Stakeholder entspricht, signifikant verringert werden. Die Entwicklungskosten werden vom Requirements Engineering ebenfalls maßgeblich beeinflusst. In einer späten Projektphase haben Änderungen größere Auswirkungen und verursachen somit weitaus höhere Kosten, als in einer frühen Projektphase. Im Nachhinein gesehen könnte man einen Punkt verbessern. Wäre das Modell der Filterpressenanlage zu einem früheren Zeitpunkt als Diskussionsbasis zur Verfügung gestanden, wäre die Anforderung *14 – ein Projekt kann mehrere Pressen (Linien) enthalten* vermutlich früher aufgekommen. Dann hätte sie eventuell bereits in der Grundstruktur der Software berücksichtigt werden können und wäre somit zum Auslieferungszeitpunkt erfüllt gewesen.

9.2 Ausblick

Mit dem 19.11.2018 wurde die erste Version des *Metris Engineering Configurators* im Unternehmen ANDRITZ freigegeben. Sie entspricht bereits einem Großteil der erhobenen Anforderungen. Im nächsten Schritt sollen die Funktionen, deren Entwicklung zeitlich verschoben wurden, durchgeführt werden. Mit dem ersten Update am 21.12.2018 (Release 1.1) soll hauptsächlich die Dokumentation der Filterpressenanlage erstellt werden können, ohne Comos zu öffnen. Das zweite Update, welches bereits mit 24.01.2019 geplant ist (Release 1.2), soll eine zusätzliche Funktionalität enthalten. Es soll möglich sein, mehrere Filterpressenlinien zu konfigurieren. Diese Anforderungen werden im Kapitel 5 detailliert beschrieben. Des Weiteren soll

der *Metris Engineering Configurator* zukünftig für die Konfiguration anderer Anlagen oder Ausrüstungen wie z.B. Anlagen zur Erzeugung von Tissue-Papier oder auch elektrische Schaltschränke zum Einsatz kommen. Vor allem die Verwendung für die Erstellung der elektrischen Dokumentation eines RIO-Schrankes (Remote-I/O-Schrank) ist bereits in Planung. Um die Benutzerfreundlichkeit zu erhöhen, könnte zusätzlich zur WPF-Anwendung eine webbasierte Lösung z.B. in Form einer ASP.NET-Anwendung entwickelt werden. Das hätte den Vorteil, dass die neu entwickelte Software von jedem Gerät mit einem Browser über eine Website aufgerufen werden könnte. Außerdem müssten keine Updates an die Benutzer verteilt werden. Eine weitere bzw. zusätzliche Lösung zur Erhöhung der Benutzerfreundlichkeit wäre die Entwicklung einer App, mit welcher ebenfalls Anlagen konfiguriert werden könnten.

Das Engineering Programm Comos wird bei ANDRITZ von ca. 800 Mitarbeitern auf vier Kontinenten verwendet. Da es bis jetzt keine mit dem *Metris Engineering Configurator* vergleichbare Software gibt, soll sie auch, zumindest konzernintern, vermarktet werden. Ein erster Schritt wurde mit der Definition eines Produktnamens bereits erledigt.

Ein interessanter Aspekt für die Zukunft wäre die weitere Vernetzung des *Metris Engineering Configurators* mit den im Abwicklungsprozess vor- und nachgelagerten Systemen. Beispielsweise könnte die neu entwickelte Software mit einer Schnittstelle zum ERP-System direkt die Bestellungen auslösen.

LITERATURVERZEICHNIS

Gedruckte Werke (35)

Andresen, Andreas (2003): *Komponentenbasierte Softwareentwicklung*, Carl Hanser Verlag, München Wien

Baker, Paul; u.a. (2008): *Model-Driven Testing*, Springer-Verlag, Berlin Heidelberg

Brandt-Pook, Hans; Kollmeier, Rainer (2015): *Softwareentwicklung kompakt und verständlich*, 2. Auflage, Springer Vieweg, Wiesbaden

Bratby, John (2016): *Coagulation and Flocculation in Water and Wastewater Treatment*, 3. Auflage, IWA Publishing, London

Charette, Robert (1990): *Applications Strategies for Risk Analysis*, McGraw-Hill, New York

COLLABNET; VERSIONONE (2017): *12th Annual State of Agile Report*, 12. Auflage

Davis, Alan (1993): *Software Requirements - Objects, Functions and States*, Prentice Hall PTR, Upper Saddle River

Galitz, Wilbert (2007): *The Essential Guide to User Interface Design*, 3. Auflage, Wiley Publishing, Indianapolis

Gasper, Horst; Oechsle, Dietmar; Pongratz, Elmar (Hrsg.) (2000): *Handbuch der industriellen Fest/Flüssig-Filtration*, 2. Auflage, WILEY-VCH Verlag, Weinheim

Gernert, Christiane; Ahrend, Norbert (2000): *IT-Management: System statt Chaos*, Oldenbourg Verlag, München, Wien

Goll, Joachim; Hommel, Daniel (2015): *Mit Scrum zum gewünschten System*, Springer-Vieweg, Esslingen

Henrich, Andreas (2002): *Management von Softwareprojekten*, Oldenbourg-Verlag, München

Höhn, Reinhard; Höppner, Stephan (2008): *Das V-Modell XT*, Springer-Verlag, Berlin Heidelberg, New York

Jochem, Roland (2010): *Was kostet Qualität?*, Carl Hanser Verlag, München

Kleuker, Stephan (2018): *Grundkurs Software-Engineering mit UML*, 4. Auflage, Springer Vieweg, Wiesbaden

Krallmann, Achim; Dockter, Diana; Ritter, Alexander (2017): *Modellbasiertes Requirements Engineering*, entwickler.press, Frankfurt am Main

Kuhrmann, Marco; Ternité, Thomas; Friedrich, Jan (2011): *Das V-Modell XT anpassen*, Springer-Verlag, Berlin Heidelberg

Leffingwell, Dean (2011): *Agile Software Requirements*, Addison-Wesley, Boston

Ludewig, Jochen; Lichter, Horst (2013): *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*, 3. Auflage, dpunkt.Verlag, Heidelberg

Maximini, Dominik (2018): *Scrum - Einführung in der Unternehmenspraxis*, 2. Auflage, Springer Gabler, Hattenhofen

Partsch, Helmuth (2010): *Requirements-Engineering systematisch*, 2 Auflage, Springer-Verlag, Berlin Heidelberg

Poetzsch-Heffter, Arnd (2009): *Konzepte objektorientierter Programmierung*, 2. Auflage, Springer-Verlag, Berlin Heidelberg

Pohl, Klaus; Rupp, Chris (2015): *Basiswissen Requirements Engineering*, 4. Auflage, dpunkt.verlag GmbH, Heidelberg

Rumpe, Bernhard (2011): *Modellierung mit UML*, 2. Auflage, Springer-Verlag, Berlin Heidelberg

Rupp, Chris; SOPHIST GROUP (2007): *Requirements- Engineering und Management*, 4. Auflage, Hanser Verlag, Nürnberg

Sandhaus, Gregor; Berg, Björn; Knott, Philip (2014): *Hybride Softwareentwicklung*, Springer Vieweg, Berlin Heidelberg

Spillner, Andreas; Linz, Tilo (2012): *Basiswissen Softwaretest*, 5. Auflage, dpunkt.verlag, Heidelberg

Stachowiak, Herbert (1973): *Allgemeine Modelltheorie*, Springer-Verlag, Wien, New-York

Stahlknecht, Peter; Hasenkamp, Ulrich (2002): *Einführung in die Wirtschaftsinformatik*, 10. Auflage, Springer-Verlag, Berlin Heidelberg

Stephens, Matt; Rosenberg, Doug (2003): *Extreme Programming Refactored: The Case Against XP*, Apress, New York City

Svarovsky, Ladislav (Hrsg.) (2000): *Solid-Liquid Separation*, 4. Auflage, Butterworth-Heinemann, Oxford

Tarleton, Steve; Wakeman, Richard (2007): *Solid/Liquid Separation: Equipment Selection and Process Design*, Butterworth-Heinemann, Oxford

Unhelkar, Bhuvan (2018): *Software Engineering with UML*, CRC Press, Boca Raton

van Randen, Hendrik; Bercker, Christian; Fieml, Julian (2016): *Einführung in UML*, Springer-Vieweg, Wiesbaden

Witte, Frank (2016): *Testmanagement und Softwaretest*, Springer Vieweg, Wiesbaden

Wissenschaftliche Artikel (3)

Kano, Noriaki; u.a. (1984): *Attractive Quality and Must-be Quality*, in: Quality - The Journal of the Japanese Society for Quality Control, 14/1984, S. 39-44

Schulte, Tim; Mayerhofer, Lisa (2016): *Erweiterung des integrierten Konzeptes aus Prozessrahmenwerk und Beschreibungssystematik von mecPro² um ein modellbasiertes Variantenmanagement*

Shahin, Ashraf (2014): *Variability Modeling for Customizable SaaS Applications*, in: International Journal of Computer Science & Information Technology (IJCSIT), Vol. 6, No. 5/2014, AIRCC, S. 39-49

Konferenzbeiträge (4)

Buxton, John; Randell, Brian (1969): *Software Engineering Techniques*, in: *Report on a conference sponsored by the NATO SCIENCE COMMITTEE*, Rome

Naur, Peter; Randell, Brian (1968): *Software Engineering*, in: *Report on a conference sponsored by the NATO SCIENCE COMMITTEE*, Scientific Affairs Division, Garmisch

Royce, Winston (1970): *Managing the Development of Large Software Systems*, in: IEEE (Hrsg.): *ICSE '87 Proceedings of the 9th international conference on Software Engineering*, Wescon, S. 328-338

Wallin, C.; u.a. (2002): *Combining models for business decisions and software development*, in: Society, IEEE (Hrsg.): *28th EUROMICRO Conference*, Dortmund, S. 266-271

Online-Quellen (21)

ANDRITZ (2017a): *ANDRITZ präsentiert "Metris - Industrial IoT Solutions"* https://www.andritz.com/group-de/presse/news/2017-05-31-andritz-prasentiert-metris [Stand: 27.November.2018]

ANDRITZ (2017b): *ANDRITZ SEPARATION filter press sidebar SP – air over oil (trailer)* https://www.youtube.com/watch?v=Lhb4dtqcjA4 [Stand: 27.November.2018]

ANDRITZ (2018a): *Metris - ANDRITZ Group* https://www.andritz.com/metris-en [Stand: 27.November.2018]

ANDRITZ (2018b): *Breaking ground for the digitalized future* https://www.andritz.com/resource/blob/258372/63d5aa48a8d2f4b8e647adcc83d160f2/andritz-sidebar-and-overhead-filter-presses-data.pdf [Stand: 27.November.2018]

ANDRITZ (2018c): *ANDRITZ filter press* https://www.andritz.com/products-en/group/separation/filter-presses/filter-press-side-bar-overhead [Stand: 27.November.2018]

Angermeier, Daniel; u.a. (2018): *V-Modell-XT-Gesamt* http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/2.2/V-Modell-XT-Gesamt.pdf [Stand: 27.November.2018]

Beauftragte der Bundesregierung Deutschland für Informationstechnik, Vitt (2018): *V-Modell XT* https://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html [Stand: 27.November.2018]

Beck, Kent; u.a. (2001a): *Manifest für Agile Softwareentwicklung* http://agilemanifesto.org/iso/de/manifesto.html [Stand: 27.November.2018]

Beck, Kent; u.a. (2001b): *Prinzipien hinter dem Agilen Manifest* http://agilemanifesto.org/iso/de/principles.html [Stand: 27.November.2018]

Carl Hanser Verlag (2018): *QZ-online.de* https://www.qz-online.de/qualitaets-management/qm-basics/kunden/kundenmanagement/artikel/kundenanforderungen-kano-modell-168360.html [Stand: 27.November.2018]

Evolus (2017): *Pencil Project*
https://pencil.evolus.vn/ [Stand: 27.November.2018]

G.U.N.T. Gerätebau (2018): *Gunt - Produkte*
https://www.gunt.de/de/produkte/prozesstechnik/mechanische-verfahrenstechnik/trennverfahren-filtration/rahmenfilterpresse/083.28700/ce287/glct-1:pa-119:ca-234:pr-31 [Stand: 27.November.2018]

Glinz, Martin (2014): *A Glossary of Requirements Engineering Terminology*
https://www.ireb.org/content/downloads/1-cpre-glossary/ireb_cpre_glossary_16_en.pdf [Stand: 27.November.2018]

IREB (2018a): *IREB in Kurzform*
https://www.ireb.org/de/about/basics/ [Stand: 27.November.2018]

IREB (2018b): *Übersicht CPRE*
https://www.ireb.org/de/cpre/basics/ [Stand: 27.November.2018]

Kirchner, Bernd (2018): *filterpresse.de - Wissenswertes*
http://www.kammerfilterpresse.de/wissen.htm [Stand: 27.November.2018]

Lippert, Frank; Graser, Franz (2016): *Software-Variantenmanagement mit SysML*
https://www.embedded-software-engineering.de/software-variantenmanagement-mit-sysml-a-554157/ [Stand: 27.November.2018]

Roos-Frantz, Fabricia; Benavides, David; Ruiz-Cortés, Antonio (2011): *Feature Model to Orthogonal Variability Model Transformation towards Interoperability between Tools - Researchgate*
https://www.researchgate.net/publication/247935125_Feature_Model_to_Orthogonal_Variability_Model_Transformation_towards_Interoperability_between_Tools [Stand: 27.November.2018]

Siemens AG (2018a): *COMOS auf einen Blick*
https://w3.siemens.com/mcms/plant-engineering-software/de/comos-ueberblick/Seiten/Default.aspx [Stand: 27.November.2018]

Siemens AG (2018b): *Life Cycle Engineering und Anlagenmanagement mit Comos*
https://w3.siemens.com/mcms/plant-engineering-software/de/Seiten/Default.aspx [Stand: 27.November.2018]

Togar, Eyub (2017): *ResearchGate*
https://www.researchgate.net/publication/312294855_Implementierung_ausgewahlter_Car2X_Use_Cases/figures [Stand: 27.November.2018]

ABBILDUNGSVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

ANSI	American National Standard Institute
CPRE	Certified Professional for Requirements Engineering
DIN	Deutsches Institut für Normung
EIA	Electronic Industries Alliance
EN	European Normative, europäische Norm
ER	Entity Relationship
EER	erweiterte Entity-Relationship
FODA	Feature Oriented Domain Analysis
GUI	Graphical-User-Interface, Benutzeroberfläche
HMI	Human machine interface, Mensch-Maschine Schnittstelle
IEEE	Institute of Electrical and Electronics Engineers
IO	Input/Output, Eingang/Ausgang
IoT	Internet of Things
IREB	International Requirements Engineering Board
ISO	International Organization for Standardization, Internationale Organisation für Normung
KM	Konfigurationsmanagement
MSA	Modern Structured Analysis
OMT	Object Modeling Technique
OOA	Object Oriented Analysis
OOSE	Object Oriented Software Engineering
OVM	Orthogonal Variability Modeling
P&ID	Piping & Instrumentation Diagram, Rohrleitungs- und Instrumentenfließschema
RIO-Schrank	Remote-Input/Output-Schrank, dezentraler Schaltschrank mit Ein-/Ausgabebaugruppen
SA	Structured Analysis
SADT	Structured Analysis and Design Technique
SA/RT	Structured Analysis / Real-Time
SB	side bar, Seitenholm
SEI	Software Engineering Institute
SPS	Speicherprogrammierbare Steuerung
SysML	Systems Modeling Language

UML	Unified Modeling Language
V	Variant
VP	Variation Point
WPF	Windows Presentation Foundation

ANHANG 1: PROJEKTAUFTRAG

Automatisierte Erstellung der Anlagendokumentation	PROJEKT-AUFTRAG	ANDRITZ AG

Projektbeschreibung:

Mit der im Zuge der Masterarbeit entwickelten Software soll ein vollständig automatisierter Prozess zur Erstellung der Anlagendokumentation im bei ANDRITZ verwendeten Engineering Programm Comos gestartet werden. Mit dieser Software sollen Dokumente, wie z.B. P&ID's, Stromlaufpläne und Funktionspläne vollautomatisch erstellt werden. Weiters soll es ein Excel-Interface zur Software geben, mit dem die Anlage bereits von der Verkäuferin, dem Verkäufer vollständig konfiguriert werden kann. Die Software soll mittels einer Filterpressenanlage, die hierfür vollständig modularisiert werden muss, getestet werden.

Projektstartereignis: • Mündlicher Auftrag vom PAG (ANDRITZ AG)	**Projektstarttermin:** • 5.3.2018
Projektendereignis: • Abschlusspräsentation im Unternehmen	**Projektendtermin:** • 7.2.2019
Projektziele: Hauptziele: • (Z1) P&ID's, Stromlaufpläne, Funktionspläne und zugehörige Listen einer Filterpressenanlage können vollautomatisch erstellt werden • (Z2) Die Software ist einfach bedienbar (von Comos User) und wartbar (von Comos Key-User) • (Z3) Die Anlage kann im Comos oder über ein Excel-Interface vollständig konfiguriert werden Nebenziele: • Kompetenzen für die Abwicklung eines Entwicklungsprojektes werden erweitert (Requirement-, Test Engineering etc.)	**Nicht-Projektziele:** • (N1) Software ist für andere Anlagen außer Filterpressenanlagen verwendbar • (N2) Alle Eventualitäten einer Filterpressenanlage werden abgedeckt (Vergleich Kosten/Nutzen) • (N3) Software ist eigenständig oder mit anderen Programmen außer Comos einsetzbar
Hauptaufgaben (Projektphasen): • Anforderungen erheben – Projektauftraggeber, Endkunde etc. • Geeignetes Grundkonzept erstellen • Grundfunktionalitäten entwickeln –> Proof of Concept • Weitere Funktionalitäten entwickeln • Tests mit Software • Dokumentation erstellen	**Projektbudget:** Stunden: 300h Kosten für Entwickler etc.: €18.000,-
Projektauftraggeber: • PAG	**Projekt Manager:** • PM

Anlagen:
Anlage 1:

Seite 1/1 Datum: 19.03.2018

Abbildung 71: Projektauftrag, Quelle: Eigene Darstellung.

ANHANG 2: ERGÄNZUNG ZUM UML-KLASSENDIAGRAMM

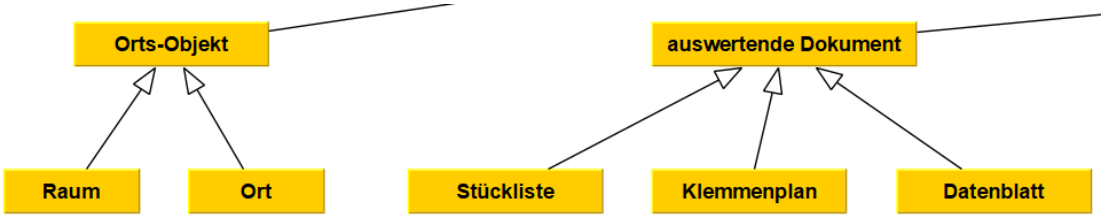


Abbildung 72: UML-Klassendiagramm – *Orts-Objekt*, *auswertende Dokument*, Quelle: Eigene Darstellung.

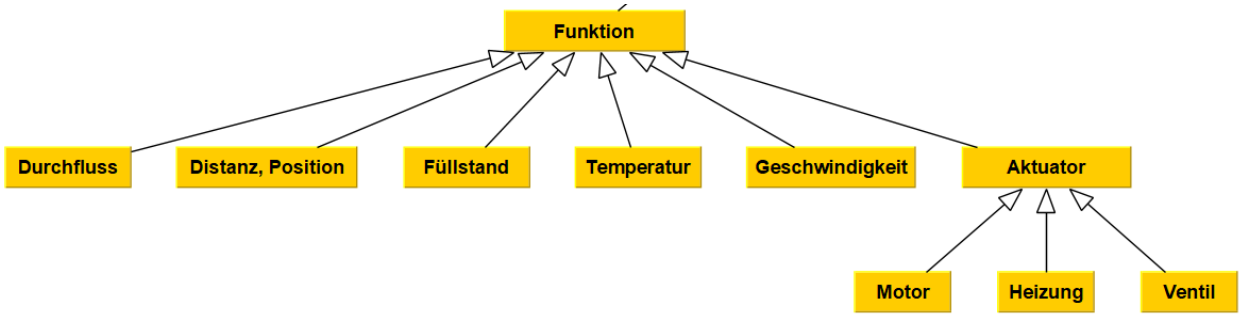


Abbildung 73: UML-Klassendiagramm – *Funktion*, Quelle: Eigene Darstellung.

ANHANG 3: ERGÄNZUNG ZUM MODELL DER FILTERPRESSENANLAGE

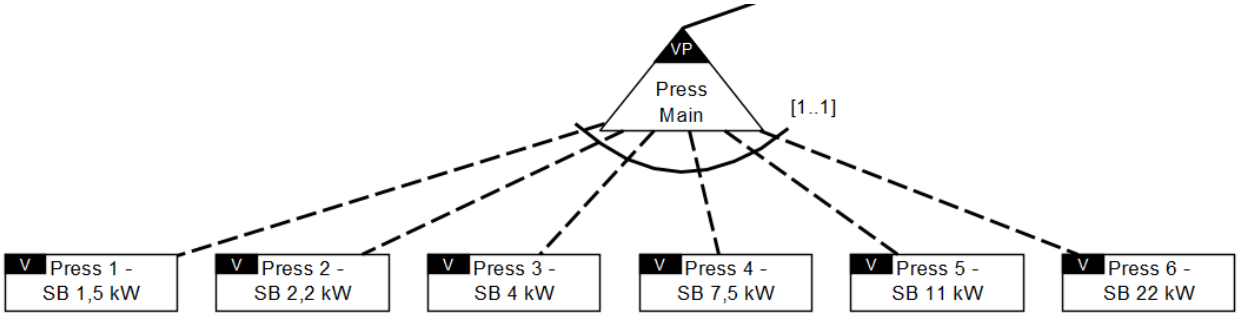


Abbildung 74: Modell der Filterpressenanlage – Press Main, Quelle: Eigene Darstellung.

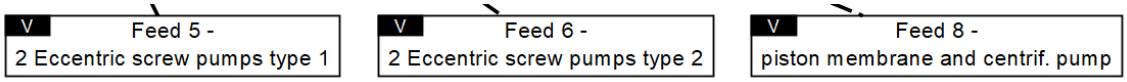


Abbildung 75: Modell der Filterpressenanlage – Feed Pumps Teil 1, Quelle: Eigene Darstellung.

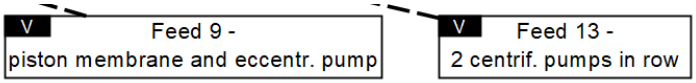


Abbildung 76: Modell der Filterpressenanlage – Feed Pumps Teil 2, Quelle: Eigene Darstellung.