Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Entwicklung eines kontextgestützten Editors für die Erstellung hybrider Prozesse

Bachelorarbeit

im Studiengang Technische Informatik

von

Daniel Morbach

Prüfer: Prof. Kurt Schneider
Zweitprüfer: Dr. Jil Klünder
Betreuer: Nils Prenner

Hannover, 26.08.2021

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 26.08.2021

Daniel Morbach

Zusammenfassung

Softwareunternehmen haben ein ständiges Interesse an der Verbesserung des internen Entwicklungsprozesses. Traditionelle Entwicklungsmethoden versprechen Planungssicherheit und ein guten Umgang mit großen Softwareprojekten. Jedoch haben sie Probleme während der Entwicklung auf Änderungen der Anforderungen einzugehen. Da Anforderungsänderungen je nach Projekt durchaus wahrscheinlich sein können, verwenden viele Unternehmen stattdessen agile Entwicklungsmethoden, welche jedoch Schwierigkeiten im Umgang mit großen Projekten haben. Um die Vorteile beider Ansätze nutzen zu können, kombinieren Unternehmen immer öfter traditionelle und agile Entwicklungsansätzen zu sogenannten hybriden Entwicklungsansätzen. Da die Anforderungen an einen solchen hybriden Entwicklungsansatz von Unternehmen zu Unternehmen und von Projekt zu Projekt sehr unterschiedlich sein können, gibt es keinen allgemeingültigen, für alle Situationen passenden, hybriden Entwicklungsansatz. Vielmehr muss er für jedes Unternehmen individuell strukturiert und organisiert werden. So muss entschieden werden, wann und in welchem Umfang bestimmte Phasen durchzuführen sind, welche Aktivitäten während den Phasen durchgeführt werden und welche Personen oder Teams daran beteiligt sein sollten.
Jede dieser Entscheidungen ist von einer Vielzahl an Einflussfaktoren abhängig, und stellt die Unternehmen somit vor die Herausforderung, die passendsten Entscheidungen für das eigene Unternehmen zu treffen. Ohne die nötige Expertise, die zu Beginn fehlt, ist dies nur durch langjähriges Ausprobieren und sammeln von Erfahrungen möglich.
Um diese Herausforderung zu adressieren, wurde im Rahmen dieser Arbeit ein Tool zur Erstellung und Bearbeitung von hybriden Prozessen als Webanwendung entwickelt und implementiert. Die Anwendung generiert aus den Kontextfaktoren einen passenden hybriden Entwicklungsansatz, der dann im Editor weiter angepasst und zugeschnitten werden kann.

Abstract

Development of a context-based editor for the creation of hybrid processes

Software companies have continuous interest in improving their own intern development process. Traditional (or plan-based) development methods promise reliable planning and a good handling of large projects. However, they usually have problems responding to changes in requirements during development. Since changes in requirements can be quite likely depending on the project, many companies use agile development methods instead, which, however, have difficulties in dealing with large projects. To make use of the advantages of both approaches, companies combine traditonal and agile development approaches to so called hybrid development approaches. Since the requirements for such a hybrid development approach can be very different from company to company and from project to project, there is no universal hybrid development approach that is suitable for all situations. It rather has to be structured and organized individually for each company. It has to be decided when and to what extent certain phases are executed, which activities are carried out during these phases and which people or teams should be involved.

Each of these decisions is dependent on a variety of influencing factors, and thus faces the company with the challenge of making the most appropriate decisions for their own company. Without the necessary expertise, which is missing at the beginning, this is only possible through many years of trying out and gaining experience.

In order to adress this challange, a tool for creating and editing hybrid development processes was developed and implemented as web application. The Tool generates based on the context factors a hybrid development process which can be further customized in the editor.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In der Theorie gibt es hauptsächlich zwei vorherrschende Entwicklungsansätze in der Softwareentwicklung. Auf der einen Seite steht der traditionelle (plan-basierte) Ansatz, welcher langfristige, gut strukturierte und sichere Planung in den Vordergrund stellt. Auf der anderen Seite steht der agile Ansatz. Dieser legt den Fokus vor allem auf Flexibilität, Kundenzufriedenheit und schnelle Anpassungsfähigkeit, falls sich die Anforderungen an das Projekt ändern sollten. Die Praxis hingegen zeichnet ein anderes Bild. Die Forschung belegt, dass Unternehmen bevorzugt Methoden der traditionellen Softwareentwicklung mit agilen Methoden vermischen und zusammen verwenden [12, 17]. Sogenannte hybride Entwicklungsansätze.
Kuhrmann et al. [6, p.2] definieren hybride Entwicklungsansätze wie folgt:

> *„any combination of agile and traditional (plan-driven or rich) approaches that an organizational unit adopts and customizes to its own context needs (e.g. application domain, culture, process, project, organizational structure, techniques, technologies, etc.)“*

Unternehmen versuchen damit ihren Entwicklungsansatz besser auf die eigenen Bedürfnisse zuzuschneiden. Es sollen die Vorteile aus beiden Entwicklungsansätzen nutzbar gemacht werden. Gleichzeitig versucht man die Nachteil zu eliminieren. Denn traditionelle Entwicklungsansätze wie das Wasserfaltmodell [13] eignen sich zwar gut um ein Projekt langfristig zu planen, jedoch fehlt es ihnen an regelmäßigen Feedback vom Kunden [4]. So werden Fehler oftmals erst spät durch den Kunden entdeckt, was hohe Fixkosten mit sich bringt. Außerdem können aufgrund der einmaligen Planung zu Beginn des Projekts Anforderungsänderungen nur schwer später noch berücksichtigt werden [2]. Agile Ansätze wie SCRUM [16] wurden entwickelt um mit diesen Problemen umzugehen. Die Entwicklung in agilen Ansätzen findet in Iterationen statt, was häufiges Feedback vom Kunden ermöglicht aber langfristige Planung unmöglich macht [4].

1.1 Problemstellung

Einen hybriden Entwicklungsansatz, welcher von jedem Unternehmen eingesetzt werden kann, gibt es nicht. Unternehmen müssen sich ihren eigenen hybriden Entwicklungsansatz erstellen [10]. Kündler et al. [5] beschreibt, dass die meisten existierenden hybriden Entwicklungsansätze durch evolutionären Prozesse, also durch langjähriges Austesten und sammeln von Erfahrungen, entstanden sind. In diesem Prozess müssen immer verschiedene Einflussfaktoren in Betracht gezogen werden, um zu entscheiden, ob und welche Änderungen am hybriden Entwicklungsansatzes vorgenommen werden sollten. Diese Änderungen betreffen vor allem die Organisation des Ansatzes. Die Organisation beschreibt, wann und in welchem Umfang bestimmte Phasen durchzuführen sind, welche Aktivitäten während den Phasen durchgeführt werden und welche Personen oder Teams daran beteiligt sein sollten. Doch die große Anzahl an zu berücksichtigenden Einflussfaktoren, wie unter anderem Teamgröße, Kritikalität des zu entwickelnden Produkts, Verfügbarkeit von Stakeholdern während der Entwicklung und vielen weiteren,macht es den Unternehmen schwer, die richtigen Entscheidungen im Bezug auf die Organisation des Ansatzes zu treffen [8, 11]. Den Unternehmen fehlt ein verlässliches Hilfsmittel, dass sie bei der Erstellung eines hybriden Entwicklungsansatzes unterstützt.

1.2 Lösungsansatz

Daher soll im Rahmen dieser Arbeit eine Webanwendung entwickelt werden, die Unternehmen einen, auf sie zugeschnittenen, hybriden Entwicklungsansatz erstellt. Als Ausgangspunkt für diesen Ansatz dient der generische hybride Entwicklungsansatz von Prenner et al. [11], welcher in Kapitel 2 genauer erläutert wird. Zunächst soll das Tool Aussagen über die Kontextfaktoren des Unternehmens vom Nutzer entgegennehmen. Aus den Antworten soll die Software mit Hilfe eines multikriterielle Einscheidungsprozesses den generischen Entwicklungsansatz so anpassen, dass er möglichst gut auf das Unternehmen des Nutzer abgestimmt ist. Die zu treffenden Entscheidungen aus dem Entscheidungsprozess stammen ebenfalls von Prenner et al. [11]. Der individuelle hybride Entwicklungsansatz soll nach dem Einscheidungsprozesses grafisch im Tool dargestellt werden. Da es bestimmte Kontextfaktoren gibt, die erst nach Beginn eines Projektes klar werden, soll es die Möglichkeit geben, den Entwicklungsansatz abzuspeichern und zu laden [11]. Diese Kontextfaktoren sollen dann zu einem späteren Zeitpunkt im Tool eingeben werden können.

1.3 Struktur der Arbeit

In Kapitel 2 werden wichtige Grundlagen, die zum Verständnis dieser Arbeit notwendig sind, erklärt. Dazu wird zunächst die Frage geklärt, was hybride Entwicklungsansätze sind und welche Muster es bei der Struktur solcher Prozesse gibt. Außerdem soll dem Leser notwendiges Wissen im Bereich der multikriteriellen Entscheidungsfindung vermittelt werden. Kapitel 3 beschäftigt sich darauf hin mit dem Konzept der zu entwerfenden Software. In Kapitel 4 wird drauf hin die Implementierung der Software beschrieben. Nach diesen Kapiteln sollte der Leser ein umfassendes Verständnis über das Tool und dessen Möglichkeiten haben. Kapitel 5 befasst sich dann noch mit den wichtigsten verwandten Arbeiten, bevor in Kapitel 6 die Arbeit noch einmal kurz zusammengefasst wird, und ein Ausblick für weitere Arbeiten auf Basis dieser Arbeit gegeben wird.

Kapitel 2

Grundlagen

In diesem Kapitel werden die wichtigsten Begriffe, die zum Verständnis dieser Arbeit notwendig sind, erläutert. Es ist von Vorteil, wenn der Leser mit den Publikationen von Prenner et al. [12, 10, 11] vertraut ist, da diese Arbeit wesentlich auf diesen Publikationen aufbaut. Zunächst werden in Abschnitt 2.1 hybride Entwicklungsprozesse allgemein erläutert. Darauf hin erhält der Leser einen Überblick über einen speziellen, hybriden Entwicklungsprozess, der für diese Arbeit sehr wichtig ist. In Abschnitt 2.2 werden dann anschließend multikriterielle Entscheidungsprobleme erklärt, und wie sie mithilfe eines analytischen Hierarchieprozesses gelöst werden können.

2.1 Hybride Entwicklungsansätze

Hybride Entwicklungsansätze sind Ansätze, die sich aus Methoden der traditionellen Softwareentwicklung und der agilen Softwareentwicklung zusammensetzten. Sie unterscheiden sich darin, welch Methoden jeweils aus beiden Bereichen verwendet werden, und wie der Entwicklungsansatz organisiert ist. Jeder Ansatz ist in Phasen aufgeteilt. Die Organisation bestimmt, wann und in welchem Umfang die Phasen durchgeführt werden, welche Aktivitäten innerhalb der Phasen stattzufinden haben, und welche Personen oder Teams an den Phasen beteiligt sind.

2.1.1 Arten

Im Rahmen einer systematischen Literaturrecherche von Prenner et al. [12] wurde eine Analyse, von in der Praxis existierenden hybriden Entwicklungsansätzen, durchgeführt. Diese Literaturrecherche hat ergeben, dass alle untersuchten hybriden Prozesse in irgendeiner Weise das Wasserfallmodell von Royce [13] verwenden. Sie unterscheiden sich aber im Umfang und der Anordnung der Phasen. Ebenfalls wurde festgestellt, dass

sich jeder Prozess in eine von drei Hauptansätzen einordnen lässt:

Die erste Gruppe an Prozessen folgt dem sogenannten **Waterfall-Agile-Approach (WAA)**, welcher (siehe Abbildung 2.1) dem Wasserfallmodell [13] sehr ähnlich ist. In den ersten Phasen werden die Anforderungen und die Architektur eher grob geplant und erstellt. Die Coding-Phase aus dem Wasserfallmodell wird zu einer iterativen Entwicklungsphase [11]. In dieser werden dann agile Methoden wie zum Beispiel SCRUM[16] genutzt. Dort werden Anforderungen und Designentscheidungen konkretisiert. Die in den ersten beiden Phasen erstellten Dokumente können für die agile Entwicklung in einen Product Backlog umgewandelt werden.

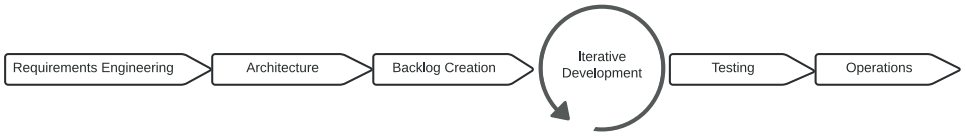

Abbildung 2.1: Waterfall-Agile-Approach (WAA)

Ein weitere Gruppe folgt dem **Waterfall-Iteration-Approach (WIA)**, in welchem das komplette Wasserfallmodell [13] iterativ durchgeführt (siehe Abbildung 2.2). Es werden die selben Phasen wie beim **WAA** durchlaufen. Lediglich die Backlog Creation Phase entfällt, da dies direkt in der Requirements Engineering Phase geschieht.

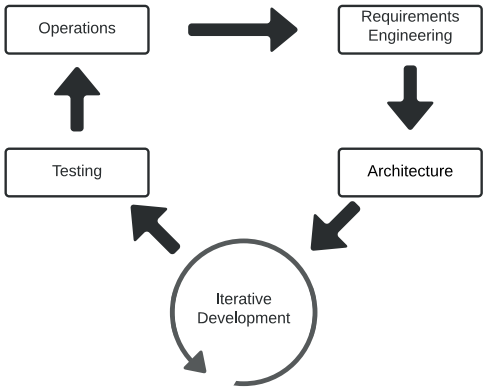

Abbildung 2.2: Waterfall-Iteration-Approach (WIA)

Anders als in den ersten beiden Gruppen werden im sogenannten **Pipeline-Approach (PA)** alle Phasen parallel durchgeführt, indem man die Entwicklung in Inkremente aufteilt. Wie in Abbildung 2.3 zu sehen ist, beschreibt N das sich gerade in Entwicklung befindliche Inkrement in Iteration I. Parallel dazu wird die Architektur für das Inkrement

$N + 1$ entwickelt und alle nötigen Tests werden für das Inkrement $N - 1$ durchgeführt. In der nächsten Iteration $(I + 1)$ rutschen alle Inkremente jeweils in die nächste Phase.

Requirements Engineering	N	N+1	N+2	N+3	N+4
Architecture	N-1	N	N+1	N+2	N+3
Iterative Development	N-2	N-1	N	N+1	N+2
Testing	N-3	N-2	N-1	N	N+1
Operations	N-4	N-3	N-2	N-1	N
Iteration	I-2	I-1	I	I+1	I+2

Abbildung 2.3: Pipeline-Approach nach Paige et al. [9]

2.1.2 Generischer hybrider Entwicklungsprozess

Auf Basis der Erkenntnisse aus dem letzten Abschnitt, hat Prenner et al. [11] einen generischen hybriden Entwicklungsprozess entwickelt. Wie in Abbildung 2.4 zu sehen, wird als Basis der Waterfall-Agile-Approach genutzt, welcher zu vor der iterativen Entwicklungsphase um eine parallele Planungsphase ergänzt wurde. Während der iterativen Entwicklung kommt eine Kombination aus Waterfall-Iteration-Approach und Pipeline-Approach zum Einsatz [12]. So ist es mit diesem generischen Prozess möglich, durch gezieltes Anpassen der Phasen, für jedes Unternehmen einen passenden hybriden Entwicklungsprozess zu erstellen [11]. Dieser Anpassungsprozess ist wesentlicher Bestandteil dieser Arbeit und wird im Kapitel 3 erläutert.

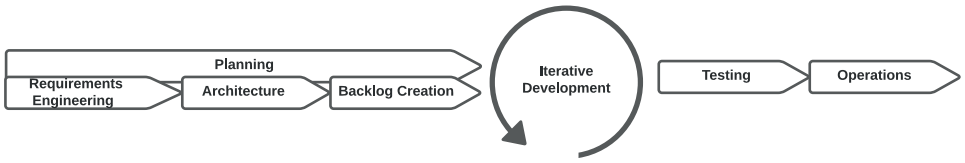

Abbildung 2.4: Basis für den generischen hybriden Entwicklungsprozesses

Im Folgenden werden die Aktivitäten des generischen hybriden Entwicklungsprozesses vorgestellt.

Aktivitäten vor der iterativen Entwicklung

Bevor die iterative Entwicklungsphase beginnt, kann es jeweils eine initiale Requirements Engineering-, Architektur-, Planungs- und Backlog-Creation Phase geben. Ob jede dieser Phasen benötigt wird, und in welchem Umfang sie durchgeführt werden, hängt vom Projekt ab.

Initiales Requirements Engineering Diese Phase wird genutzt, um die Anforderungen der Kunden und Stakeholder zu bestimmen und analysieren. Es können bereits erste Dokumente wie Epics und GUI-Mockups erstellt werden. Zuständig sind in der Regel der Product Owner sowie Business Experten.

Initiale Architektur-Phase In dieser Phase wird die grundlegende Architektur der Software erstellt. Grundsätzlich sind für diese Phase die Software-Architekten zuständig, die Software Developer können sich aber auch beteiligen. Product Owner und Business Experten aus dem vorangegangenem Requirements Engineering sollten auch zur Verfügung stehen, um gegebenenfalls Fragen beantworten zu können. Es werden unter anderem Klassendiagramme sowie weitere Dokumente erstellt. Auch das Design der GUI kann hier bereits grob erstellt werden.

Initiale Planungs-Phase Während dieser Phase werden, basierend auf dem Umfang der Anforderungen, Budget und Zeitrahmen des Projektes festgelegt. Es handelt sich aber dabei erst mal um eine grobe Planung, sodass Veränderungen der Anforderungen während der Entwicklung noch mit einbezogen werden können. Wie in Abbildung 2.4 zu sehen, überlappt diese Phase mit der Requirements Engineering, Architektur und Backlog-Creation Phase, weil die Planung in vielen Fällen in diesen Phasen integriert ist. Verantwortlich für diese Phase ist der Projektleiter.

Initiale Backlog-Creation Phase Wenn im Requirements Engineering zu Beginn hauptsächlich Dokumente angefertigt wurden, die eher in traditionellen Entwicklungsprozessen benötigt werden, kann die Phase dazu genutzt werden, diese Dokumente in einen Product Backlog, User Stories und Epics umzuwandeln. Diese Dokumente können dann in der folgenden, durch agile Methoden geprägte, Entwicklungsphase verwendet werden.

Aktivitäten während der iterativer Entwicklung

Im Mittelpunkt dieses Models steht die iterative Entwicklung. Diese ist in Abbildung 2.5 dargestellt, und kann im Bereich, der in Abbildung 2.4 mit „Iterative Development “ gekennzeichnet ist, eingesetzt werden. Phasen die in der Abbildung grau hinterlegt sind, werden iterativ durchgeführt. Konkret bedeutet dies, dass alle Phasen, die innerhalb der iterativen Phasen liegen, auch iterativ durchgeführt werden. Für den Sprint bedeutet es, dass pro Iteration der iterativen Entwicklung, mehrere Sprints durchgeführt werden.

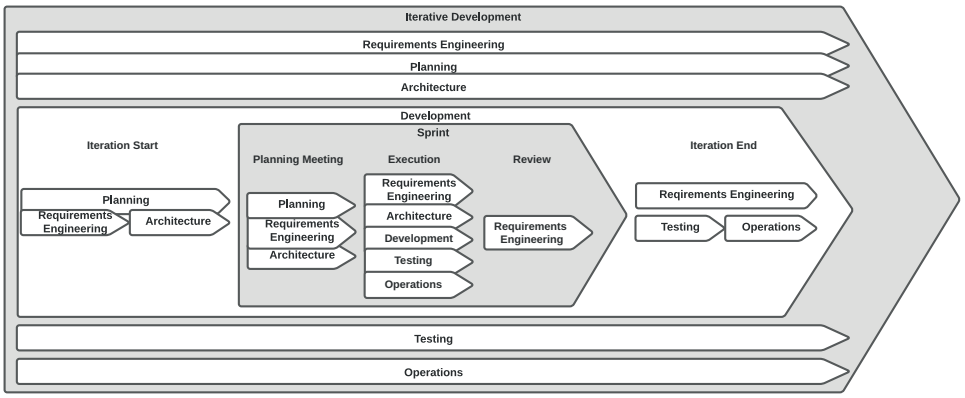

Abbildung 2.5: Iterative Entwicklungsphase

Paralelles Requirements Engineering Da die Anforderungen während des initialen Requirements Engineerings eher oberflächlich erhoben wurden,

müssen weitere Anforderungen und die Details in dieser Phase geklärt werden. Product Owner und Business-Experten arbeiten dabei zusammen. Sie prüfen konstant ob sich Anforderungen ändern oder neue Anforderungen hinzu kommen und bestimmen dabei den Umfang der Entwicklung für die nächste Iteration. In Abstimmung mit den Entwicklern ergänzen sie den Product Backlog.

Paralelle Architektur-Phase In dieser Phase wird die konkrete Architektur für die kommende Iteration entwickelt. Mit neuen Anforderungen wächst auch die Architektur der Software. Gutes Management wird benötigt, um den Umfang der Architektur zu kontrollieren.

Paralelle Planungs-Phase Wenn sich Anforderungen ändern oder neue Anforderungen hinzu kommen, hat das Einfluss auf den Gesamtplan. Daher dient dieser Prozess dazu, den Gesamtplan zu überprüfen und die kommenden Iterationen detaillierter zu planen. Nicht nur Änderungen bei den Anforderungen können zu einer Änderung des Plans führen, auch Probleme während der Entwicklung können ein Grund sein. Für diese Phase ist in der Regel der Projektleiter zuständig. Da dieser aber oft mit den Teams aus dem Requirements Engineering und der Architektur zusammen arbeitet, kann auch diese Planungsphase in den anderen Phasen integriert werden.

Entwicklungsphase Diese Phase ist als iterative Entwicklungsphase organisiert. Sie hat einen Iterationsstart und ein Iterationsende. Die eigentlichen Iterationen finden in Form von mehreren Sprints zwischen Start und Ende statt.
Der Iterationsstart findet oft in Form eines Meetings statt, welches ein oder mehrere Tage dauern kann. Hier haben die Entwickler die Möglichkeit direkt mit Stakeholdern und Nutzer über die Anforderungen zu sprechen. Das kann als weitere Requirements Engineering Phase aufgefasst werden. Der Unterschied zum parallelen Requirements Engineering ist, dass hier die Entwickler und nicht der Product Owner verantwortlich sind. Dieser direkte Austausch hilft den Entwicklern bei der Selbstorganisation und kann somit die Entwicklung beschleunigen[7].
Neben den Anforderungen kann auch die Architektur mit den Entwicklern besprochen werden.
Darüber hinaus ist die Planung der Sprints selbst wichtig. Der Unterschied zur parallelen Planungsphase ist, dass sich die Planung mehr auf die Sprints bezieht. Der direkte Einfluss und das Feedback der Entwickler in der Planung hilft dem Projektleiter den Plan besser anzupassen.
Nach dem Iterationsstart beginnt die Entwicklung in Form von Sprints. Obwohl in hybriden Entwicklungsansätzen oftmals mehrere Entwicklerteams in einem Projekt zusammenarbeiten, funktioniert dieses Modell auch für

nur ein Entwicklerteam. Die Struktur des Sprints orientiert sich am agilen SCRUM Framework. Jeder Sprint beginnt mit einem Planning Meeting, indem die Anforderungen für den aktuellen Sprint besprochen werden. Außerdem kann über die Architektur gesprochen werden. Im Vergleich zur Architektur Phase im Iterationsstart, die den Fokus auf die Koordination zwischen den Teams legt, dient diese Phase zur Besprechung der Architektur innerhalb der Teams.
Nach dem Meeting beginnt die Sprint Execution Phase. Alle Anforderungen für diesen Sprint werden in Software umgesetzt. Gibt es noch keine konkrete Architektur, wird diese auch in dieser Phase entwickelt. Um sicher zu gehen, dass die entwickelte Software auch funktioniert, können in dieser Phase bereits (Modul)-Tests durchgeführt werden. Wenn neue Funktionalität direkt für die Veröffentlichung bereit ist, kann dies bereits in dieser Phase passieren. Außerdem können während der Entwicklung Anforderungen angepasst werden, die für den nächsten Sprint relevant sind.
Am Ende des Sprints besprechen die Entwickler mit dem Product Owner, ob die gesetzten Ziele für diesen Sprint erfüllt wurden.
Wie bereits beschrieben, besteht eine Iteration aus mehreren Sprints und verschiedene Teams können an der selben Lösung arbeiten. Daher müssen am Iterationsende die verschieden Teile aus der Entwicklung zusammengeführt werden. Es müssen Integrationstest durchgeführt werden, und die neue Version kann ausgeliefert werden. Da am Ende jeder Iteration ein größeres Softwarepaket entwickelt wurde, kommen Kunden und Stakeholder zusammen, um das Produkt zu besprechen. Das Meeting wird vom Product Owner und Business-Experten unterstützt um Feedback zu sammeln. Das kann als abschließende Requirements Engineering Phase angesehen werden.

Parallele Testphase Wie im letzten Abschnitt beschrieben, muss die Integration der Software getestet werden. Der Unterschied zur Testphase am Iterationsende ist es, dass diese Phase von einem separaten Team durchgeführt wird und für die Qualitätssicherung verantwortlich ist.

Parallele Operation-Phase Ähnlich wie beim parallelen Testprozess, wird während dieses Prozesses die Software regelmäßig von einem separaten Team ausgeliefert. Aber auch hier muss das Operations-Team eng mit den Entwicklern zusammenarbeiten.

Aktivitäten nach der iterativen Entwicklung

Nachdem die iterative Entwicklung abgeschlossen wurde, können noch eine abschließende Test und Operation Phase notwendig sein.

Abschließende Testphase Die Phase ist notwendig, wenn es nicht ausreicht, die einzelnen Softwareinkremente während der iterativen Entwicklungsphase zu testen. Manchmal muss das Produkt als ganzes getestet werden.

Abschließende Operation-Phase Für diese Phase gilt das selbe wie für die vorherige Testphase. Wenn das Produkt nur als Ganzes verwendet werden kann, ist diese Phase notwendig.

2.2 Analytische Hierarchieprozess (AHP)

Der analytische Hierarchieprozess dient zum Lösen von multikriteriellen Entscheidungsproblemen. Eine Entscheidung ist definiert als die Auswahl einer Handlungsalternative aus einer Menge verfügbarer Alternativen. Multikriterielle Entscheidungsprobleme beschreibt also eine Klasse von Entscheidungsproblemen, bei denen die Güte einer Handlungsalternative von mehreren Kriterien abhängig ist. Ein Kriterium kann sich sowohl positiv als auch negativ auf die Güte einer Alternative auswirken. Ziel ist es, die Kriterien untereinander abzuwägen und in Folge dessen die beste Handlungsalternative auszuwählen. Der AHP ist ein solcher Abwägungsprozess. Er wurde von dem Mathematiker Thomas L.Saaty in den 1970er Jahren an der Wharton School of Business entwickelt, und kommt seitdem in vielen Bereichen zum Einsatz [15]. Der AHP soll im Rahmen dieser Arbeit verwendet werden um, anhand der vom Nutzer angegebenen Einflussfaktoren, die bestmögliche Anpassungsalternative für den generischen hybriden Entwicklungsprozess auszuwählen. Den AHP besteht aus mehreren Phasen, die alle durchlaufen werden müssen, um eine erfolgreiche Entscheidungsfindung durchzuführen.

2.2.1 Phase 1: Definition des Entscheidungsproblems

Zunächst muss das Entscheidungsproblem, welches die Basis eines jeden AHP ist, exakt definiert werden. Aus dem Problem müssen dann alle relevanten Einflussfaktoren bzw. Kriterien abgeleitet werden. Wenn es sich um ein sehr komplexes Entscheidungsproblem handelt, können die Kriterien weiter in Subkriterien gegliedert werden. Zuletzt müssen alle in Frage kommenden Lösungsalternativen erfasst werden. Bereits vorhandener Studien, Literaturrecherchen sowie Expertenbefragungen können bei der Ausarbeitung von Alternativen und Kriterien nützlich sein. Ist dieser Schritt abgeschlossen, erfolgt die Einordnung der Einflussfaktoren und Alternativen

in eine Hierarchie. Das Entscheidungsproblem befindet sich immer auf der obersten Hierarchieebene. Die Lösungsalternativen hingegen, befinden sich alle auf der untersten Ebene. Dazwischen befindet sich eine variable Anzahl an Ebenen für die Einflussfaktoren. Die Anzahl ist von der Komplexität des Entscheidungsproblems, also der Menge an Kriterien und Subkriterien abhängig. In Abbildung 2.6 ist eine solche Hierarchie mit genau einer Ebene für die Einflussfaktoren zu sehen. Sie stellt ein, für diese Arbeit relevantes, Entscheidungsproblem dar und wird in einem späteren Abschnitt genauer erläutert.

2.2.2 Phase 2: Datenerhebung

Jedes Element einer Ebene wird paarweise mit den anderen Elementen der selben Ebene im Bezug auf ein Element der übergeordneten Ebene verglichen. Folglich wird in diesem Prozess die relative Wichtigkeit eines Elementes ermittelt. Hierbei ist die 9-Punkte-Skala nach Saaty [14] ein häufig verwendetes Werkzeug zur Bewertung der Kriterien und Alternativen. Mit dieser Skala weißt der Entscheider jedem Vergleich einen Wert zwischen 1 und 9 zu. Umso höher der Wert, desto stärker ausgeprägt ist die Differenz der Wichtigkeiten, wobei 1 einen Wichtigkeitsgleichgewicht und 9 einen starken Wichtigkeitsungleichgewicht entspricht. Eine vollständige Aufschlüsselung der Skalenwerte können der Tabelle 2.1 entnommen werden. Im konkreten Beispiel aus Abbildung 2.6 stellen die Verbindungslinien zwischen den Elementen die Abhängigkeiten für die Vergleiche dar. Dies bedeutet, die vier Kriterien müssen paarweise in Bezug auf das Ziel miteinander verglichen werden. Außerdem müssen die Alternativen paarweise in Bezug auf jedes Kriterium miteinander verglichen werden. Wird beispielsweise ein Kriterium C_i im Vergleich zum Kriterium C_j mit dem Wert 7 bewertet, wird davon ausgegangen, dass der umgekehrte Vergleich (C_j zu C_i) mit dem reziproken Wert, also $\frac{1}{7}$, bewertet wird.

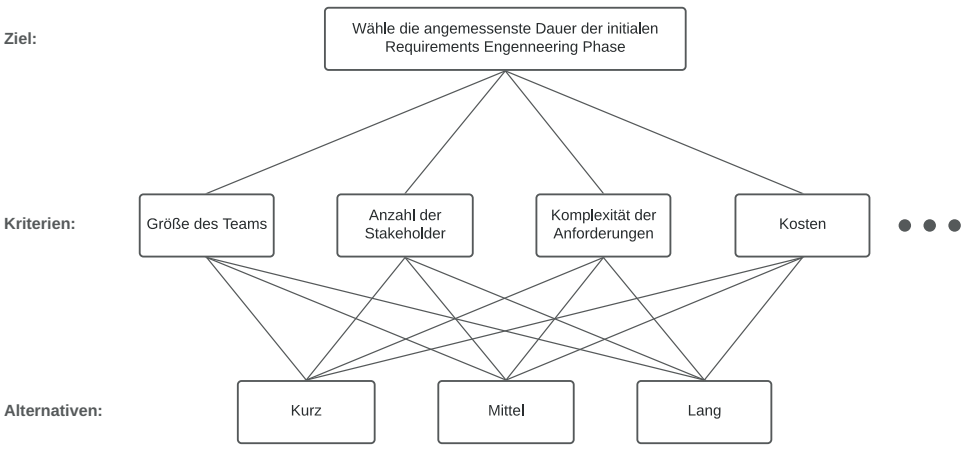

Abbildung 2.6: Auschnitt aus der AHP Hierarchie der Entscheidung über die Dauer der initialen Requirements Engineering Phase.

Tabelle 2.1: Die Fundamentalskala für Paarvergleiche

Wert der Wichtigkeit	**Definition**	**Erklärung**
1	Gleiche Beduetung	Zwei Aktivitäten sind gleichbedeutend
3	Etwas größere Beduetung	Erfahrung und Einschätzung sprechen für eine etwas größere Bedeutung der einen Aktivität im Vergleich zur anderen
5	Erheblich größere Bedueutng	Erfahrung und Einschätzung sprechen für eine erheblich größere Bedeutung der einen Aktivität im Vergleich zur anderen
7	Sehr viel größere Beduetung	Die sehr viel größere Beduetung der einen Aktivität im Vergleich zur anderen hat sich in der Vergangenheit klar gezeigt
9	Absolut dominierend	Der größtmögliche Bedeutungsunterschied welcher zwischen zwei Aktivitäten möglich ist
2,4,6,8	Zwischenwerte	Feinabstufung

2.2.3 Phase 3: Datenauswertung

Die in der letzten Phase getroffen Präferenzurteile aus den paarweisen Vergleichen werden in dieser Phase in sogenannte Vergleichsmatrizen eingetragen. Wie bereits im letzten Abschnitt erläutert, muss für jeden Eintrag a_{ij} in der Matrix gelten:

$$a_{ij} = \frac{1}{a_{ji}} \tag{2.1}$$

Es gibt eine Vergleichsmatrix, welche die Kriterien im Bezug auf das Ziel gegenüberstellt. Durch die Bedingung 2.1 ist jeder Eintag auf der Diagonalen der Matrix gleich Eins. Die Diagonaleinträge entsprechen dem Vergleich eines Kriteriums mit sich selbst. Eine Matrix mit beispielhaften Werten ist als Tabelle 2.2 dargestellt.

Tabelle 2.2: Vergleich der Kriterien im Bezug auf das übergeordnete Ziel

Ziel	Kriterium C_1	Kriterium C_2	Kriterium C_3
Kriterium C_1	1	4	5
Kriterium C_2	$\frac{1}{4}$	1	$\frac{1}{3}$
Kriterium C_3	$\frac{1}{5}$	3	1

Da die Alternativen bezüglich jedes Kriteriums miteinander verglichen werden müssen, gibt es für $n = 4$ Kriterien (wie in Abbildung 2.6) auch insgesamt 4 Vergleichsmatrizen. Für jedes Kriterium gibt es also eine Matrix der Art **??**.

Wenn alle Vergleichsmatrizen erstellt wurden, können die relativen Gewichte der Kriterien und Alternativen im Bezug auf das jeweilige Element der darüber liegenden Hierarchieebene berechnet werden. Diese Gewichte werden mit Hilfe von Eigenvektoren berechnet. Es wird stets der Eigenvektor des maximalen Eigenwertes λ_{max} verwendet [3, 14].Damit die Summe der Gewichte den Wert 1 annimmt, muss der Eigenvektor normiert werden. Es gibt verschiedene Möglichkeiten die Eigenwerte und Eigenvektoren zu ermitteln. Es können exakte Verfahren als auch numerische Näherungsverfahren verwendet werden. Die Berechnung wird in dieser Arbeit nicht weiter erläutert. Der normierte Eigenvektor entspricht dem gewünschten Gewichts-Vektor.
Am Ende kann noch eine Konsistenzprüfung durchgeführt werden. Eine Entscheidung gilt als konsistent, wenn beispielsweise das Kriterium C_1 zweimal wichtiger als C_2, C_2 dagegen dreimal wichtiger als C_3 und C_1 folglich sechsmal wichtiger als C_3 ist [1]. Mathematisch nennt man diese Eigenschaft *Transitivität*. Eine Vergleichsmatrix ist konsistent, wenn $a_{ij} \times a_{jk} = a_{ik}$ für beliebige i,j und k gilt. Jedoch ist es unwahrscheinlich, das Menschen immer perfekt konsistente Entscheidungen treffen. Daher erlaubt der AHP eine geringe Abweichung von der Konsistenz. Der Grad der Inkonsistenz gibt Aufschluss über die Validität des Ergebnisses. Mit dem maximalen Eigenwert λ_{max} und der Anzahl der Spalten bzw. Zeilen n, berechnet sich der Konsistenzindex CI wie folgt [3]:

$$CI = \frac{\lambda_{max} - n}{n - 1} \tag{2.2}$$

Um herauszufinden, ob die paarweisen Vergleiche erneut durchgeführt werden sollten, kann man den sogenannten Konsistenzwert (CR = Consistency Ratio) bestimmen:

$$CR = \frac{CI}{R} \tag{2.3}$$

R steht für den sogenannten *Random Index*, ein von der Dimension der Matrix abhängiger, durchschnittliche Konsistenzindex zufälliger, reziproker Matrizen. Die ersten zehn Werte sind in Tabelle 2.3 abhängig von der Dimension der Matrix angegeben. Nach Saaty ist für $CR \geq 0.1$ eine Re-Evaluation der Paarvergleichsurteile durchzuführen [3].

Tabelle 2.3: Durchschnittlicher Konsistenzindex R bei gegebener Matrixgröße

Größe der Matrix	1	2	3	4	5	6	7	8	9	10
Random Index R	0,00	0,00	0,52	0,89	1,11	1,25	1,35	1,40	1,45	1,49

2.2.4 Phase 4: Dateninterpretation

In dieser Phase werden alle Werte aus den berechneten Vektoren in einer Entscheidungsmatrix zusammengetragen, um die beste Alternative zu bestimmen. Der Vektor, der aus dem Vergleichen der Kriterien hervorgegangen ist, enthält die Gewichtungen der Kriterien. In den anderen Vektoren steht jeweils, wie gut sich eine Alternative im Bezug auf ein Kriterium eignet. In Abbildung 2.4 ist eine Entscheidungsmatrix mit beispielhaften Werten gefüllt. Alternative A_1 eignet sich nicht gut im Bezug auf Kriterium C_1 , aber recht gut im Bezug auf Kriterium C_3. Kriterium C_3 ist mit einer Gewichtung von 0.6 auch viel wichtiger als Kriterium C_1, mit einem Gewicht von nur 0.2. Um die beste Handlungsalternative zu bestimmen, werden die Werte der Kriterien mit der jeweiligen Gewichtung multipliziert, und dann aufsummiert. Die Handlungsalternative mit dem höchsten Score ist dann die beste. Im Beispiel aus Abbildung 2.4 wäre dies Alternative A_2 mit einem Score von 0.4.

Tabelle 2.4: Entscheidungsmatrix mit beispielhaften Werten

Kriterien	Gewichtung	Alternative A_1	Alternative A_2	Alternative A_3
Kriterium C_1	0.2	0.1	0.25	0.65
Kriterium C_2	0.2	0.7	0.25	0.05
Kriterium C_3	0.6	0.3	0.5	0.2
Score	-	0.34	0.4	0.26

Kapitel 3

Konzept

In diesem Kapitel wird das grundlegende Konzept der zu entwickelnden Software erläutert. Zunächst werden in Abschnitt 3.1 Ideen für das Konzept formuliert. Diese werden im Abschnitt 3.2 in Anforderungen umgewandelt, die bei der Konzeptionierung und Implementierung des Tools einzuhalten und umzusetzen sind. Allgemein lässt sich das Konzept der Software in drei Hauptbestandteile zerlegen legen. Der erste Teil befasst sich mit den Entscheidungsproblemen und wie diese von der Software gelösten werden sollen in Abschnitt 3.3. Im zweiten Teil, in Abschnitt 3.4, wird das Konzept für den hybriden Entwicklungsansatz vorgestellt. Der dritte Teil befasst sich schlussendlich in Abschnitt 3.5 mit der Benutzeroberfläche und den Bedienmöglichkeiten der Software.

3.1 Idee

Das Ziel des zu entwickelnden Tool ist es, dem Nutzer einen auf ihn angepassten hybriden Entwicklungsprozesses zu erstellen, welcher dann im Tool angezeigt und weiter bearbeitet werden können soll. Dazu soll der Nutzer Aussagen über Kontextfaktoren treffen, welche das Unternehmen und Projekt betreffen. Das Tool gibt dem Nutzer dabei für jeden Kontextfaktor einige Aussagen vor, von denen der Nutzer die zutreffendste Aussage auswählen soll. Mit Hilfe dieser Aussagen trifft das Tool dann eine Reihe von voneinander abhängigen, multikriteriellen Entscheidungen, die über die Struktur des hybriden Entwicklungsprozesses bestimmen. Als Ausgangspunkt für den hybriden Entwicklungsprozesses dient der generische Entwicklungsprozess von Prenner et al. [11], welcher breites in Kapitel 2, Abschnitt 2.1.2 vorgestellt wurde. Die Entscheidungen bestimmen darüber, ob bestimmte Phasen im Prozess entfernt, verlängert oder verkürzt werden, welche Aufgaben in bestimmten Phasen zu erledigen sind, und welche Personen oder Teams an bestimmten Phasen beteiligt sind. Da einige Kontextfaktoren noch nicht vor Beginn eines Projektes klar sein können,

sollen diese Kontextfaktoren auch noch nach der initialen Erstellung des Prozesses angegeben werden können. Die zu treffenden Entscheidungen wurden nicht im Rahmen dieser Arbeit aufgestellt, sondern stammen ebenfalls von Prenner et al. [11]. Das Tool soll dem Nutzer aber nicht alle Anpassungen strikt vorgeben. So soll der Nutzer beispielsweise jeder Phase individuelle Aufgaben zuteilen können, sowie die Dauer einer Phase spezifizieren können.

3.2 Anforderungen an das Tool

Aus dieser Idee wurden zu Beginn dieser Arbeit folgende Anforderungen an die zu entwickelnde Software abgeleitet:

- **A1: Die Software soll den Nutzer Kontextfaktoren bewerten lassen, um einen hybriden Entwicklungsansatz zu erstellen.**
- **A2: Auf bestimmte Kontextfaktoren soll auch noch nach der Erstellung Bezug genommen werden können.**
- **A3: Die Software soll die Möglichkeit bieten, den Entwicklungsprozess zu individualisieren.**
- **A4: Die Software soll die Möglichkeit bieten, Entwicklungsprozesse abzuspeichern und laden zu können.**
- **A5: Die Software soll komplett clientseitig, und somit unabhängig von einem Server, lauffähig sein.** Die Software soll als eine Art Prototyp angesehen werden. Es geht darum, den technischen Grundstein für ein später im Software Engineering nutzbares Tool zu legen. Dies soll zunächst unabhängig von einer Serverumgebung geschehen.
- **A6: Die Software soll möglichst modular entwickelt werden.** Es ist vorgesehen, dass der Funktionsumfang der Software zu einem späteren Zeitung erweitert wird. Außerdem kann es Änderungen am Entscheidungsfindungsprozess sowie dem generischen Modell geben. Daher soll besonderen Wert auf Modularität gelegt werden.

3.3 Entscheidungsprobleme

Die Vermischung der hybriden Entwicklungsansätze WAA, WIA und PM aus Kapitel 2, Abschnitt 2.1.2 im generischen hybriden Entwicklungsansatz sorgt dafür, dass die selben Phasen mehrfach zu verschiedenen Zeiten durchgeführt werden. Damit Unternehmen den generischen Entwicklungsansatz nutzen können, muss er den Anforderungen und dem Kontext der Unternehmen angepasst werden [11]. Um herauszufinden, welche Entscheidungsschritte durchzuführen sind und welche Handlungsalternativen in jedem Schritt gewählt werden können, hat Prenner et al. [11] eine Analyse der verschiedenen Phasen durchgeführt.
Insgesamt hat die Analyse 13 Entscheidungsprobleme ergeben, deren Handlungsalternativen die Struktur des generischen hybriden Entwicklungsprozess beeinflussen. Da es aber in dieser Arbeit nicht darum geht, welche Entscheidungsprobleme konkret behandelt werden müssen, sondern vielmehr darum, wie das Tool allgemein diese Entscheidungsprobleme löst, wird an dieser Stelle nicht konkret auf jedes einzelne Entscheidungsproblem eingegangen. Jedoch werden in den folgenden Abschnitten vereinzelt Entscheidungsprobleme herangezogen, um bestimmte Programmabläufe an einem konkreten Beispiel erklären zu können.

3.3.1 Arten von Entscheidungen

Ebenfalls ist aus der Analyse hervorgegangen, dass es zu zwei Arten von Entscheidungen kommen kann. Entweder müssen Entscheidungen getroffen werden, die von mehreren Einflussfaktoren abhängen und somit ein multikriterielles Entscheidungsproblem darstellen, oder es können Entscheidungen auftreten, die nur von einem Einflussfaktor abhängig sind.

Direkte Entscheidungen

Entscheidungen, die nur von einem einzigen Kontextfaktor abhängig sind, werden im Rahmen dieser Arbeit „direkte Entscheidungen“ genannt. Jede Lösungsalternative ist direkt an eine der Auswahlmöglichkeiten des Kontextfaktors gebunden. Sie werden immer dann genutzt, wenn einem Kontextfaktor eine so besondere Bedeutung zukommt, dass er Einfluss darauf hat wie andere Kontextfaktoren zu bewerten sind. Dies trifft zum Beispiel für die Entscheidung über die Länge der initialen Requirements Engineering Phase zu (siehe Abbildung 3.1). Maßgeblicher Faktor für die Länge dieser Phase ist der Umfang der aufgenommenen Anforderungen. Wenn generell nur wenige Anforderungen aufgenommen wurden, wird vermutlich auch keine lange RE[1] Phase zur Analyse dieser Anforderungen benötigt. Gibt es viele Anforderungen, ist wahrscheinlich eine längere RE Phase von Nöten. Jedoch

[1] Requirements Engineering

spielen bei dieser Entscheidung 10 weitere Einflussfaktoren ebenfalls eine Rolle. Eine Liste aller Einflussfaktoren befindet sich im Anhang A.

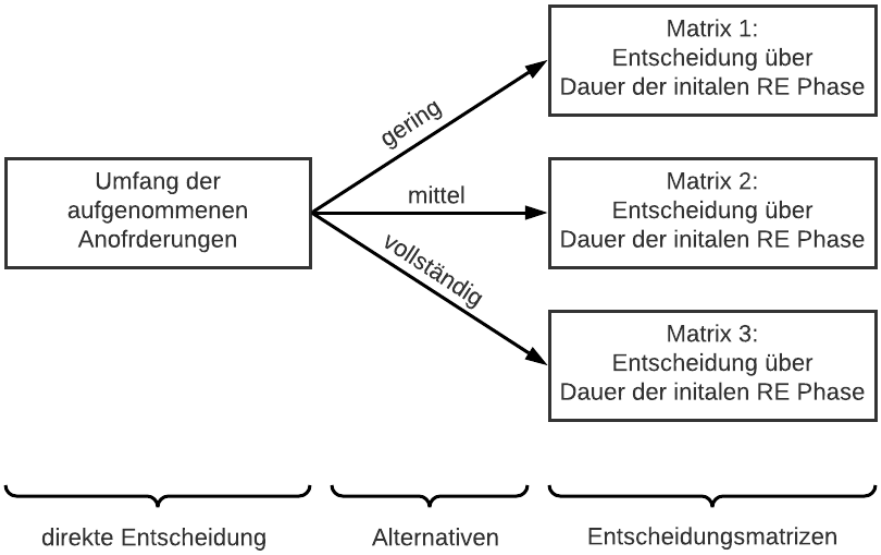

Abbildung 3.1: Entscheidung über die Dauer der initialen Requirements Engineering Phase

Entscheidungsmatrizen

Eine gute Möglichkeit, um die beste Handlungsalternative unter Einbeziehung mehrerer Einflussfaktoren zu bestimmen, bilden die Entscheidungsmatrizen. Diese werden mit Hilfe einer multikriteriellen Entscheidungsanalyse, wie dem im Kapitel 2, Absatz 2.2 vorgestellten, analytischen Hierarchieprozess erstellt. In Abbildung 3.1 ist zu sehen, dass es drei Matrizen für die selbe Entscheidung gibt. Alle drei Matrizen bieten jeweils die Handlungsalternativen: Phase verkürzen, Phase unverändert lassen und Phase verlängern. Jedoch wird nur jene Matrix zur Berechnung der besten Handlungsalternative herangezogen, welche in der vorangegangenen Entscheidung ausgewählt wurde. Die Werte in Matrix 1 sind so angepasst, das eine Entscheidung für das verkürzen der initialen Requirements Engineering Phase wahrscheinlicher ist. Für Matrix 3 gilt genau das umgekehrte.

3.3.2 Entwurf des Entscheidungsprozesses

Im Rahmen der Arbeit von Prenner et al. [11] wurden bereits Phase 1,2 und einen Teil von Phase 3 des analytischen Hierarchieprozesses aus Kapitel 2, Abschnitt 2.2 für jede der 13 Entscheidungsprobleme durchgeführt. Das bedeutet, dass für jedes Entscheidungsproblem die Handlungsalternativen im Bezug auf jeden Einflussfaktor miteinander verglichen wurden. Die sich daraus ergeben Werte dienen als Einträge in den Entscheidungsmatrizen und sind in der Software abgespeichert. Normalerweise würden in einem AHP auch noch die Einflussfaktoren im Bezug auf das Entscheidungsproblem miteinander verglichen werden, um die Gewichtungen der Kriterien zu bestimmen. Dies ist im Rahmen dieser Arbeit aber nicht zielführend, da der Nutzer ja nicht die Einflussfaktoren untereinander vergleicht, sondern jeden Kontextfaktor an und für sich bewertet. Daher müssen die Einflussfaktoren auf eine andere Weise gewichtet werden. Wie bereits in einem vorherigen Abschnitt erläutert, soll die Software dem Nutzer für jeden Einflussfaktor mehrere Aussagen zur Verfügung stellen, von denen der Nutzer die passendste auswählen soll. Damit diese Aussagen für die Gewichtung verwendet werden können, muss jeder Aussage ein numerischer Wert zugeordnet werden. Im Rahmen dieser Arbeit wurde dafür zunächst eine sehr einfache Skala gewählt. Für alle Einflussfaktoren gilt, dass die erste Aussage über den Faktor mit „wenig“ oder „schlecht“ assoziierbar ist und die letzte Aussage mit „gut“ oder „viel“. Daher werden jeweils den ersten Aussagen kleine Werte zugeordnet und den den letzten Aussagen große Werte. Die eigentliche Gewichtung des Einflussfaktors ergibt sich dann aus dem Verhältnis der vom Nutzer angegebenen Aussage und der maximalen/bestmöglichen Aussage. Beispielsweise spielt die Komplexität der Anforderungen eine Rolle bei der bereits beschriebenen Entscheidung über die Länge der initialen Requirements Engineering Phase. Der Nutzer kann zwischen wenig komplex, mittel komplex und sehr komplex auswählen. Demnach erhält „wenig komplex“ zum Beispiel den Wert 1, „mittel komplex“ den Wert 2 und „sehr komplex“ den Wert 3. Würde der Nutzer bei der Abfrage dieses Kontextfaktors die Auswahl auf „mittel komplex“ setzten, würde dieser Kontextfaktor also mit $\frac{2}{3}$ gewichtet werden, da 2 der Wert der gewählten Aussage ist, und 3 der bestmögliche Wert ist.
Um die beste Lösungsalternative einer Entscheidungsmatrix zu bestimmen, werden die Gewichte für jeden Einflussfaktor berechnet, damit für jede Alternative der Score bestimmt werden kann. Die Alternativen enthalten jeweils Informationen darüber, wie der generische hybride Entwicklungsansatz angepasst werden soll. Dies wird genauer in Kapitel 4 beschrieben. Ähnlich wie bei den direkten Entscheidungen, können auch die Alternativen in einer Entscheidungsmatrix ein nächstes Entscheidungsproblem angeben (Vergl. Abbildung 3.1). So einsteht eine Art Verkettung der Entscheidungsprobleme. Wenn die Software den hybriden Entwicklungsprozess erstellt, durchläuft

die sie so alle Entscheidungsprobleme bis zu dem Punkt, an dem eine Handlungsalternative keine nächste Entscheidungsproblem mehr angibt. An diesem Punkt wurden dann alle Entscheidungsprobleme aufgelöst und somit der Entscheidungsprozess abgeschlossen. Die Software zeigt dem Nutzer dann den hybriden Entwicklungsansatz an.

3.4 Hybrider Entwicklungsansatz

Der Entwicklungsprozesses wird im Rahmen dieser Arbeit als Baumstruktur aufgefasst. Damit lassen sich die Entwicklungsprozesse leicht aufbauen und für die Darstellung des Prozesses kann ein einfacher Traversierungsalgorithmus genutzt werden.

3.4.1 Struktur

Im Baum wird zwischen zwei Arten von Knoten unterschieden: Es gibt die sogenannten *ContentNodes*, welche die Phasen des Entwicklungsprozesses repräsentieren. Zum anderen gibt es die *StructureNodes*, welche die Anordnung der Phasen in der Darstellung bestimmen. *StructureNodes* können ihre Nachfolger auf drei weisen anordnen: Mit der Einstellung **sequentiell**, werden die Nachfolger des Knoten horizontal nebeneinander dargestellt. Diese wird genutzt, wenn die Phasen im Zeitplan hintereinander durchgeführt werden.
Mit der Einstellung **paralell**, werden die Nachfolger des Knoten vertikal dargestellt. Diese wird genutzt, wenn die Phasen im Zeitplan gleichzeitig, also von verschiedenen Teams, durchgeführt werden.
Außerdem gibt es die Einstellung **combined**, welche ein Spezialfall der parallelen Darstellung ist. Wenn parallele Phasen zusammen durchgeführt werden können, wird diese Einstellung verwendet. In der Darstellung überlappen sich dann die kombinierten Phasen. Lässt sich eine Phase in mehrere Teilphasen zerlegen, besitzt die entsprechende ContentNode genau ein Nachfolger. Dieser Nachfolger ist immer eine StructureNode, in welcher die Teilphasen angeordnet werden. In Abbildung 3.2 ist ein Ausschnitt aus der in Kapitel 2 Abschnitt 2.1.2 vorgestellten iterativen Entwicklungsphase des hybriden Prozesses als Baum dargestellt. Aus Gründen der Übersichtlichkeit wurden in dieser Abbildung die Knoten innerhalb des Sprints weggelassen.

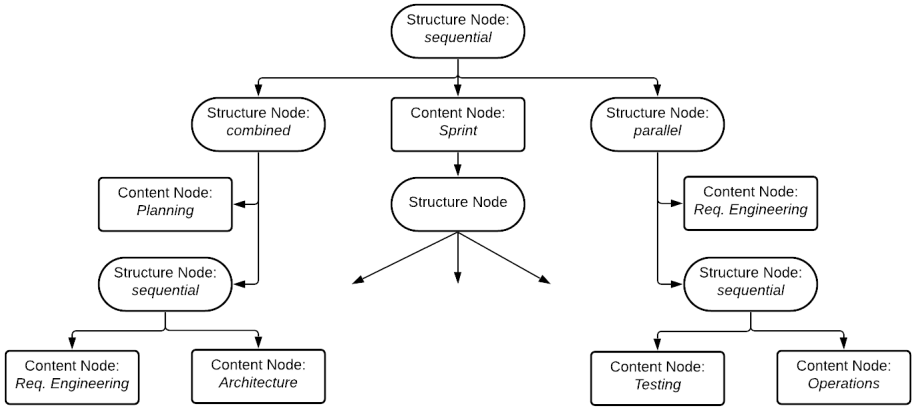

Abbildung 3.2: Baumstruktur der iterativen Development Phase.

3.4.2 Phasen

Wie bereits im letzten Abschnitt erläutert, werden die Phasen des Entwicklungsprozesses durch sogenannte *ContentNodes* repräsentiert. In ihnen werden alle, diese Phase betreffenden Eigenschaften, gespeichert. Dazu gehören ein Name sowie die Aktivität der Phase: **Requirements Engineering, Architecture (-Design), Planning, Development, Testing oder Operations**.
Der Name ist nicht immer gleichbedeutend mit der Aktivität der Phase. Beispielweise gibt vor der iterativen Entwicklung die Möglichkeit, eine Blacklog Creation Phase durchzuführen. Dieser ist aber im allgemeinen die Aktivität Requirements Engineering zuzuordnen. Die Dauer der Phase, die durchzuführenden Aufgaben während der Phase als auch eine Liste mit allgemeinen Informationen werden ebenfalls in der *ContentNode* festgehalten. Aufgaben enthalten Information darüber, ob die von der Software während des Decision Makings, oder im Editor manuell vom Nutzer hinzugefügt wurden. So können die vom Nutzer erstellten Aufgaben im Falle einer Neuberechnung des Modells automatisiert in den neuen Entwicklungsprozess übernommen werden.

3.5 Benutzeroberfächer und Interaktionen

In diesem Kapitel wird die Benutzeroberfläche mit den wesentlichen Interaktionsmöglichkeiten erläutert. Im Rahmen dieser Arbeit wurde für die Erstellung der Benutzeroberfläche das front-end open source toolkit **Bootstrap** in der Version 5.1.0 verwendet. Icons, die in der Benutzeroberfläche verwendet wurden, stammen aus der Icon-Bibliothek **Font Awesome**.

3.5.1 Abfrage der Kontextfaktoren

Startet der Nutzer das Tool, ist zunächst kein hybrider Entwicklungsprozess vorhanden. Durch das klicken des Buttons „Create New“ öffnet sich ein Dialogfenster, indem das Programm die Kontextfaktoren abfragt. Die Antwortmöglichkeiten werden dabei vom Programm vorgegeben. In Abbildung 3.3 kann man das Dialogfenster mit dem ersten abgefragten Faktor sehen.

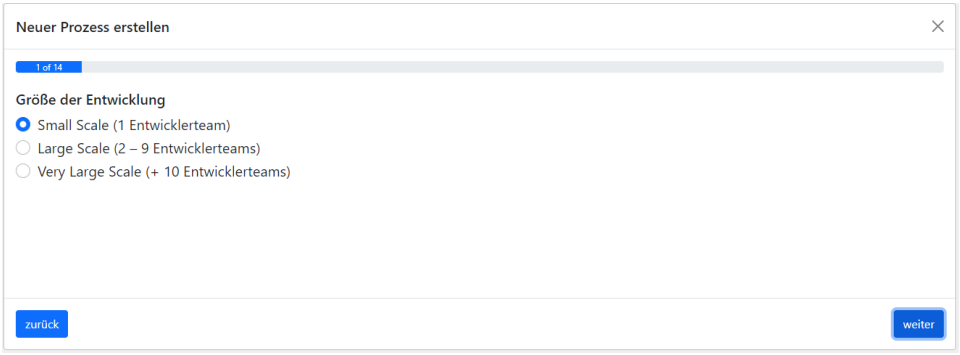

Abbildung 3.3: GUI für das Angeben der Kontextfaktoren

Über die Buttons „weiter“ und „zurück“ kann zum nächsten bzw. vorherigen Kontextfaktor navigiert werden. Nach dem letzten Kontextfaktor wird über den Button „Erstellen“ der Einscheidungsprozess in Gang gesetzt. Sobald der hybride Entwicklungsprozess angefertigt wurde, wird er im Darstellungsbereich angezeigt.

3.5.2 Darstellungsbereich

Der Darstellungsbereich nimmt den größten Teil der Benutzeroberfläche ein. In ihm wird der hybride Entwicklungsprozess angezeigt. Die Darstellung des Entwicklungsprozesses wird durch eine rekursive Funktion, die den Baum (siehe Abschnitt 3.4.1) in preorder-Reihenfolge durchläuft, realisiert. Damit man zusammengehörende Phasen besser erkennt, werden sie anhand ihrer jeweiligen Aktivität eingefärbt. Iterative Phasen werden zusätzlich durch das andeuten eines Stapels, sowie ein Symbol in der oberen rechten

Ecke hervorgehoben. Abbildung 3.4 zeigt die GUI[2] mit einem geladenen Entwicklungsprozess.

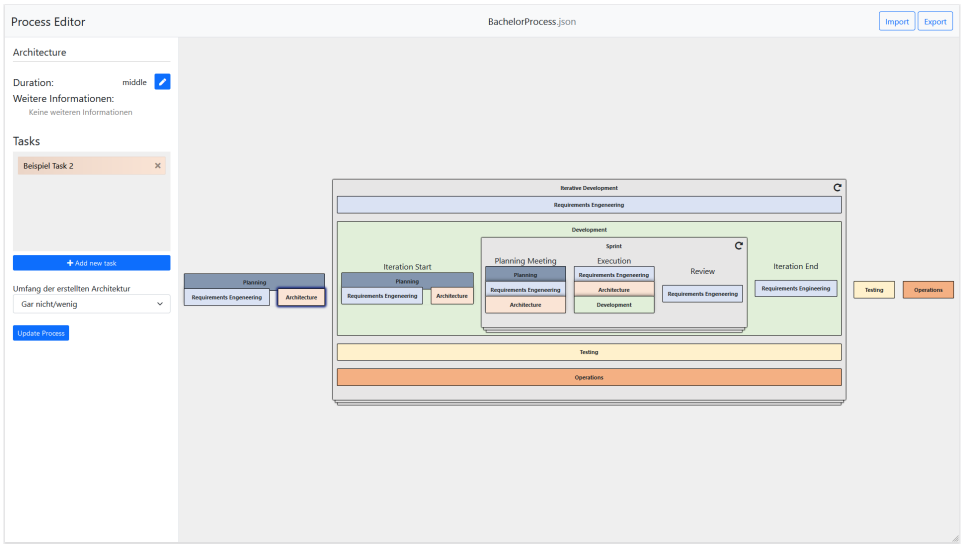

Abbildung 3.4: Zustand der GUI wenn ein hybrider Entwicklungsprozess geladen ist.

Durch das Anklicken einer Phase werden dessen Eigenschaften in der daneben liegenden Seitenleiste angezeigt.

3.5.3 Seitenleiste

In der Seitenleiste befinden sich die Eigenschaften der ausgewählten Phase und die diesbezüglichen Einstellungsmöglichkeiten. Dazu gehören Dauer, Aufgaben und allgemeine Informationen der Phase. Die Länge der Phase lässt sich durch das Klicken des Bearbeitungsbuttons anpassen. Es lässt sich der allgemeinen Dauerindikator (Kurz, Mittel, Lang) ändern, aber auch eine exakte Dauer in Form von Tagen, Wochen oder Monaten lässt sich angeben. Darunter werden weitere Phaseninformationen (falls vorhanden), sowie Aufgaben angezeigt. Aufgaben können manuell hinzugefügt und gelöscht werden. Durch eine Drag&Drop Funktion kann man diese in andere Phasen verschieben.
Wie bereits zu Beginn dieses Kapitels erläutert, gibt es Kontextfaktoren die vor dem Start des Projektes noch unklar sind und daher noch nicht zum Zeitpunkt der Erstellung des hybriden Prozesses angegeben werden können. Diese Kontextfaktoren beziehen sich in der Regel auf spezielle Phasen, und können dann zu einem gegebenen Zeitpunkt in eben jener Phase angegeben

[2] engl. Graphical User Interface

werden (siehe Abbildung 3.5). Bei der initialen Erstellung des hybriden Prozesses werden dann Default-Werte für diese Kontextfaktoren verwendet. Der Grund dafür ist, dass durch die Verkettung der Entscheidungsprobleme kein Entscheidungsproblem ausgelassen werden kann und somit schon bei der Erstellung Entscheidungen getroffen werden müssen, bei denen diese Kontextfaktoren eine Rolle spielen. Entscheidet sich der Nutzer dann, diese Kontextfaktoren zu ändern, muss er dies mit einen Klick auf den Button „Update Process“ bestätigen. Dadurch findet eine komplette Neuberechnung des Prozesses statt.

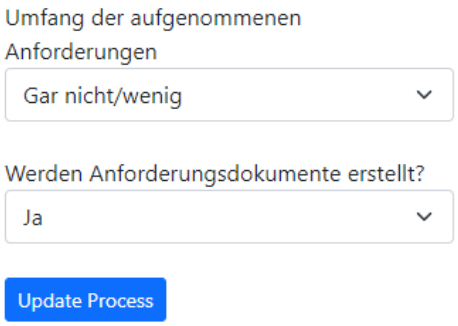

Abbildung 3.5: Abfrage der Kontextfaktoren in Seitenleiste

3.5.4 Navigationsleiste

Es ist möglich, den Namen des erstellten Entwicklungsprozesses zu ändern, in welchem der Benutzer auf den Namen in der Mitte der Navigationsleiste anklickt. Der Text wird dann durch ein Input-Feld ersetzt, indem der Nutzer einen neuen Namen eintragen kann. Durch das Drücken der Taste „Enter“ oder durch das Verlassen des Fokus des Input Feldes, wird der neue Name gespeichert. Am rechten Rand der Navigationsleiste befinden sich die Buttons für das Importieren und Exportieren von Entwicklungsprozessen. Klickt der Nutzer auf den „Exportieren“ Button, wird der vorhandene Entwicklungsprozess in einen JSON-String umgewandelt. Dieser wird dann in Form einer JSON-Datei zum automatischen Download bereitgestellt. Der Dateiname entspricht dem Namen des Entwicklungsprozesses. Durch die „Importieren“-Funktion lassen sich diese JSON-Dateien wieder einlesen.

Kapitel 4

Implementierung

Dieses Kapitel behandelt die technische Umsetzung des im letzten Kapitel 3 vorgestellten Konzeptes. In Abschnitt 4.1 wird der Leser zunächst über allgemeine Implementierungsdetails der Software informiert. Abschnitt 4.2 erläutert die konkrete Implementierung der Entscheidungsprozesse. Darauf folgend wird in Abschnitt 4.3 die interne Repräsentation des hybriden Entwicklungsprozesses beschrieben. In Abschnitt 4.4 wird abschließend erläutert, welche Schritte vor einer Auslieferung, der in dieser Arbeit entwickelten Software, nötig sind.

4.1 Allgemeine Implementierungsdetails

Um Anforderung **A5** aus Kapitel 3 zu erfüllen, wurde die Anwendung komplett in JavaScript geschrieben. JavaScript wird clientseitig im Browser des Nutzers ausgeführt, und ist somit unabhängig von einem Server. Eine Serverlaufzeitumgebung wie Node.js oder PHP kann, falls benötigt, zu einem späteren Zeitpunkt frei gewählt werden.
Außerdem sind alle Entscheidungsprozesse innerhalb dreier JSON Konfigurationsdateien gespeichert. Dies hat den klaren Vorteil, dass Änderungen am Entscheidungsprozess nicht im Programmcode vorgenommen werden müssen, sondern ganz einfach in den Konfigurationsdateien übernommen werden können.
In *decision-matrices.json* befinden sich alle Entscheidungsmatrizen und in *direct-decisions.json* befinden sich alle direkten Entscheidungen.
In *inputs.json* befindet sich eine Liste mit allen Kontextfaktoren. So lassen sich leicht weitere Kontextfaktoren hinzufügen, oder beispielsweise bestehende Faktoren ändern. Durch diese Art der Implementierung ist auch Anforderung **A6** erfüllt.

4.2 Entscheidungsprozess

4.2.1 DecisionResolver

Die Klasse **DecisionResolver.js** ist für das Lösen der Entscheidungsprobleme in der Software verantwortlich. Der Entscheidungsprozess wird gestartet, indem der DecisionResolver jeweils das erste Entscheidungsproblem der verketteten Entscheidungsprobleme an die Funktion **resolve(decision)** übergibt. In der Funktion wird überpüft, um welche Art von Entscheidung es sich handelt. Wenn es sich um eine direkte Entscheidung handelt, wird diese an die Funktion **resolveDirectDecision(decision)** weitergegeben. Dort wird die beste Handlungsalternative bestimmt, indem sie überprüft, welchen Wert der Nutzer für den betreffenden Kontextfaktor ausgewählt hat. Enthält die Alternative eine Liste mit Operationen auf dem hybriden Entwicklungsprozess, wird diese an die Funktion **applyModifications(operations)** übergeben, welche die Liste abarbeitet und die Änderungen am hybriden Entwicklungsprozess vornimmt. Enthält die Alternative einen Verweis auf eine nächste Entscheidung, wird wieder die Funktion **resolve(decision)** mit eben dieser Entscheidung ausgeführt.
Wird festgestellt, dass es sich um eine Entscheidungsmatrix handelt, wird das Entscheidungsproblem an die Funktion **resolveMatrix(decision)** übergeben. Dort werden zunächst mit Hilfe der Nutzerangaben die Gewichte der betreffenden Kontextfaktoren bestimmt. Mit diesen Gewichten wird dann die beste Alternative des Entscheidungsproblems bestimmt. Auch hier wird die Liste mit den Operationen an die **applyModifications(operations)** Funktion übergeben und das nächste Entscheidungsproblem, falls vorhanden, an die **resolve(decision)** Funktion.

4.2.2 Entscheidungsmatrizen

In Abbildung 4.1 ist zu sehen, wie die Entscheidungsmatrizen gespeichert werden. *influences* ist ein Array von ID's. Jede ID korrespondiert mit einem Kontextfaktor. Wenn der DecisionResolver das Ergebnis der Entscheidungsmatrix berechnet, verwendet er die ID's um an die von dem Nutzer angegebenen Werte zu gelangen. Im Array *alternatives* sind die Handlungsalternativen gespeichert. Im Feld *nextDecision* wird der Name der nächsten zu treffenden Entscheidung gespeichert. Dabei kann es sich ebenfalls wieder um eine Matrix handeln, oder um eine direkte Entscheidung. Das Array *values* enthält die Bewertungen der im *influences*-Array definierten Kontextfaktoren. Es ist wie folgt zu lesen:
Die erste Alternative wird im Bezug auf die Teamgröße (ID: *team-size*) mit 0.12 bewertet, die zweite Alternative mit 0.27 und die dritte mit 0.61 und so weiter. Im Array *operations* werden Operationen gespeichert, die im Falle der Entscheidung für jene Alternative, ausgeführt werden. Jede Operation hat dabei eine Zielphase (*phase*), auf der die Operation ausgeführt werden

soll. Wie Abbildung 4.1 zu sehen, kann unter anderem mit *duration* die Dauer einer Phase angepasst werden. Jede Phase des generischen hybriden Entwicklungsprozesses ist standardmäßig normale Länge. Neben den eher groben Zeitangaben kurz, mittel und lang können auch explizite Zeitspannen angegeben werden. Die Zeitangabe *duration: none* wird als nicht benötigen der Phase interpretiert. Die Phase wird dann aus dem Entwicklungsprozess entfernt.
Außerdem können Phasen Aufgaben hinzugefügt werden. Dafür braucht die Operation ein Array mit dem Namen *tasks*. In diesem können beliebig viele Tasks in Form von Strings angegeben werden.
Gleichermaßen können generelle Phaseninformationen über ein Array mit dem Namen *infos* hinzugefügt werden.

```
"mre01-upFrontRE" : {
    "influences":  ["team-size", "team-distribution",
                    "stakeholder-amount", "stakeholder-availability",
                    "requirements-changes", "result-importance",
                    "requirements-complexity", "long-term-planning",
                    "bug-impact", "cost"],
    "alternatives": [
        {"id": "reducePhase",
         "nextDecision": "decision02",
         "values": [0.12, 0.12, 0.12, 0.12, 0.49, 0.49, 0.12, 0.12, 0.12, 0.55],
         "operations": [
             {
             "phase": "UpFrontRE",
             "duration": "short"
             }
         ]
        },
        {"id": "untailoredPhase",
        "nextDecision": "decision02",
         "values": [0.27, 0.27, 0.27, 0.27, 0.31, 0.31, 0.27, 0.27, 0.27, 0.19],
         "operations": []
        },
        {"id": "extendPhase",
        "nextDecision": "decision02",
        "values": [0.61, 0.61, 0.61, 0.61, 0.19, 0.19, 0.61, 0.61, 0.61, 0.26],
        "operations": [
            {
            "phase": "UpFrontRE",
            "duration": "long"
            }
        ]
        }
    ]
},
```

Abbildung 4.1: JSON Repräsentation einer Entscheidungsmatrix

4.2.3 Direkte Entscheidungen

Der Kontextfaktor wird über das Feld *input* spezifiziert. Wie in Abbildung 4.2 zu sehen, besitzt jede Alternative ein Feld *value*. In ihm gespeichert, bei welchem Wert des Kontextfaktors die Alternative gewählt wird. In den meisten Fällen bestimmen direkte Entscheidungen nur darüber, welche Entscheidungsmatrix als nächstes gelöst werden soll (*nextDecision*). Jedoch kann eine Alternative, genau wie bei den Lösungsalternativen der Entscheidungsmatrizen, auch Operationen auf dem generischen hybriden Entwicklungsprozess beinhalten.

```
"decision02": {
    "input": "requirements-amount",
    "alternatives": [{
                      "value" : "1",
                      "nextDecision" : "mre04-team"},
                     {
                      "value" : "2",
                      "nextDecision" : "mre05-team"},
                     {
                      "value" : "3",
                      "nextDecision" : "mre06-team"}]
},
```

Abbildung 4.2: JSON Repräsentation einer direkten Entscheidung

4.3 Hybrider Entwicklungsansatz

Abbildung 4.3 zeigt die Struktur des Entwicklungsprozesses als Klassendiagramm. Es ist in JavaScript eigentlich nicht notwendig, für die objektorientierte Programmierung Klassen zu definieren. Sie sind lediglich eine, erst im Jahr 2015 eingeführte, Syntaxerweiterung des, auf Prototypen basierende, Vererbungsmodell von JavaScript. Es wurde sich dennoch bewusst für die Verwendung der Klassensyntax entschieden, da diese vor allen bei umfangreichen Programmcode sehr zur Verständlichkeit und Lesbarkeit beiträgt.
Die Klassen StructureNode und ContentNode erben beide von der abstrakten Klasse **Node**, da sie sich das Namensattribut und das Nachfolger-Array teilen. Die Klasse Process enthält den Wurzelknoten, der im letzten Kapitel beschriebenen Baumstruktur. Beim Start des Tools wird eine komplette Baumstruktur, welche dem Aufbau des generischen hybriden Entwicklungsprozesses aus Kapitel 2, Abschnitt 2.1.2 entspricht, erstellt. Nachdem der Nutzer alle Kontextfaktoren im Erstellungsprozess angegeben hat, wird der DecisionResover aufgerufen, der die Anpassungen an dieser Baumstruktur vornimmt.

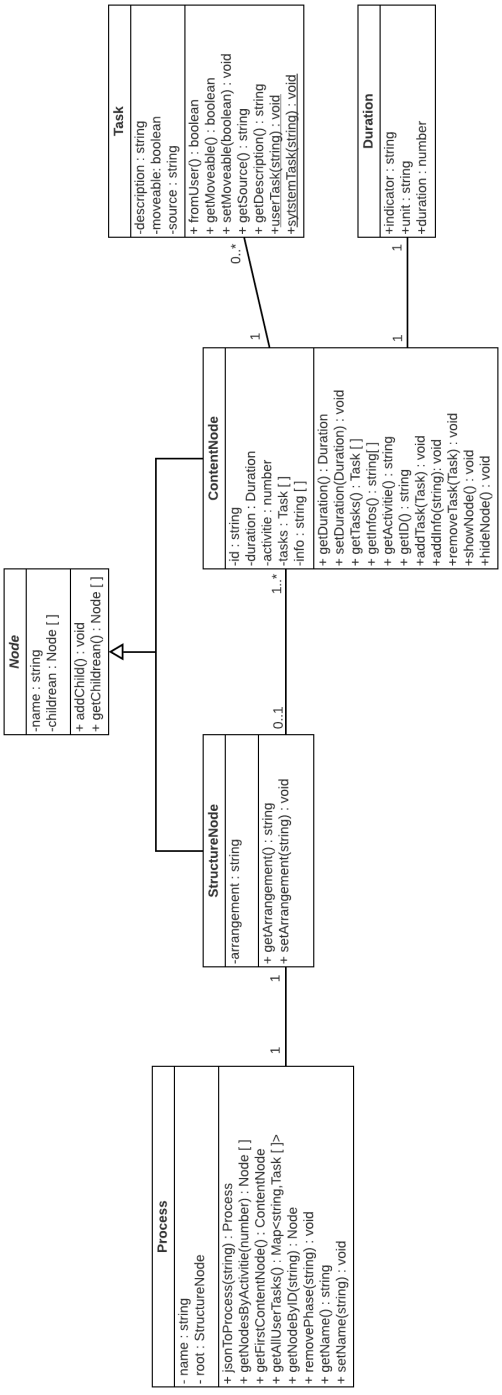

Abbildung 4.3: UML Klassendiagramm des hybriden Entwicklungsprozesses

4.4 Auslieferung

Üblicherweise werden JavaScript Tools nach ihrer Fertigstellung nicht in Form dieser Module ausgeliefert. Es wird ein sogenannter Module-Bundler verwendet, um den Code auf nur eine Datei zu reduzieren. In dieser Arbeit wurde das Tool *webpack* gewählt. Die Funktionsweise ist in Abbildung 4.4 dargestellt. Es analysiert die Abhängigkeiten innerhalb der Module (Klassen) und erstellt einen Abhängigkeitsgraphen. Dieser wird genutzt um alle Dateien zu einer *bundle.js* zusammenzuführen.
Zusätzlich wird der Code minimiert, indem überflüssige Zeichen im Code wie Leerzeichen, Tabulatoren, Linebreaks aber auch Kommentare entfernt werden. Die Reduktion des Gesamtinhalts, ohne Beeinträchtigung der Funktionalität, führt zu einer geringeren Verarbeitungszeit seitens des Browsers und schlussendlich zu einer verringerten Ladezeit von Daten.

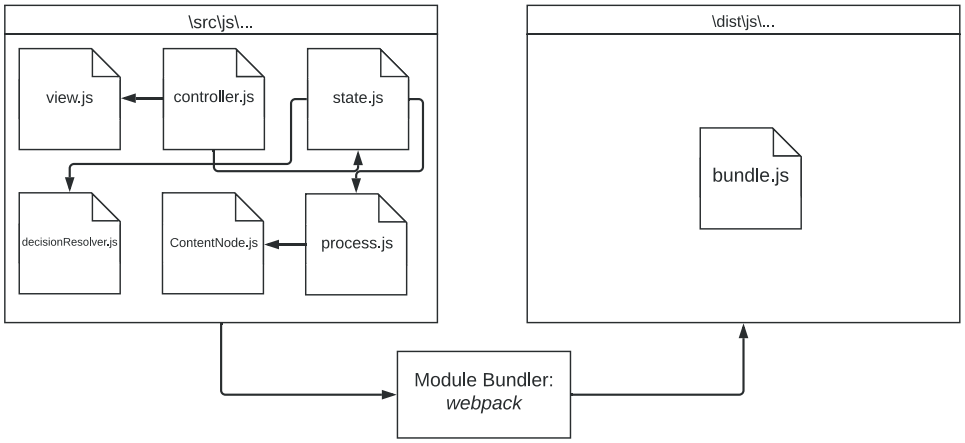

Abbildung 4.4: Schematische Darstellung der Funktionsweise eines Module Bundlers

Kapitel 5

Verwandte Arbeiten

Diese Arbeit befasste sich mit der Entwicklung und Implementierung eines Tools zur Erstellung hybrider Entwicklungsprozesse. Der Anlass für diese Bachelorarbeit ist gewesen, dass in der Praxis bisher ein Tool gefehlt hat, welches Unternehmen bei der Adaption von hybriden Entwicklungsprozessen unterstützt. Daher gibt es bis jetzt auch keine vergleichbaren Arbeiten, die sich ebenfalls mit der Entwicklung eines solchen Tools befasst haben. Jedoch ist Forschung auf diesem Gebiet noch nicht abgeschlossen, und es gibt zahlreiche Veröffentlichung die sich damit befassen, welche hybriden Entwicklungsansätze sich in Unternehmen implementiert haben, und wie sie strukturiert sind.
Allen voran ist dabei die Forschungsarbeit des Fachgebiet für Software Engineering an der Leibniz Universität Hannover zu erwähnen, da die dort entstandenen Ergebnisse als Grundlage für diese Arbeit dienten. In einer systematischen Literaturrecherche wurden von Prenner et al. [12, 10] insgesamt 24 wissenschaftliche Veröffentlichungen zu bereits existierenden hybriden Entwicklungsansätzen analysiert, um ein besseres Verständnis für die Organisation dieser Prozesse zu erlangen. Prenner et al.[11] hat auf Basis dieses erlangten Wissens einen generischen hybriden Entwicklungsprozess entworfen, der durch gezielte Anpassungen, basierend auf individuellen Kontextfaktoren, für jedes Unternehmen anwendbar sein soll.
Boehm et al. [2] haben sich ebenfalls mit der Frage beschäftigt wie hybride Entwicklungsansätze erstellt werden können. Sie verfolgen einen Risikobasierten Ansatz. In diesem findet eine Risikobewertung für das Verwenden von traditionellen und agilen Entwicklungsmethoden statt. Sollte sich daraus ergeben, dass eine rein traditionelle Entwicklungsmethode oder eine rein agile hybride Entwicklungsmethode für ein Projekt ein großes Risiko darstellt, werden gezielt diese Methoden miteinander gemischt. Jedoch wird sich nicht mit der Frage befasst, wie die daraus entstandenen hybriden Entwicklungsmethoden organisiert sein sollen.

Kapitel 6

Zusammenfassung und Ausblick

In diesem Kapitel werden die wichtigsten Inhalte dieser Arbeit zusammengefasst, sodass der Leser einen abschließenden Überblick bekommt. Zunächst wird dafür in Abschnitt 6.1 die Problemstellung wiederholt, und wie diese im Rahmen dieser Arbeit gelöst wurde. In Abschnitt 6.2 werden Weiterführungsmöglichkeiten dieser Arbeit erörtert.

6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Tool zur Erstellung und Individualisierung hybrider Entwicklungsprozesse erstellt. Sie wurde als Webanwendung entwickelt, damit einfach und unabhängig von einem bestimmten Betriebssystem verwendet werden kann. Das Tool basiert dabei vollständig auf der Forschung zur Organisation von hybriden Entwicklungsprozessen am Fachgebiet für Software Engineering an der Leibniz Universität Hannover[10, 11, 12]. Für das Konzept wurden zunächst eine Idee erstellt, welche in einem weiteren Schritt in eine Liste von Anforderungen umgewandelt wurde, die bei der Erstellung des Konzeptes und der Implementierung zu beachten war. Ein Teil des Konzepts befasste sich mit der Repräsentation von Entwicklungsprozessen in Form von Bäumen. Ein weiterer Teil des Konzeptes befasste sich damit, wie das Tool die zahlreichen Entscheidungsprobleme aus Prenner et al. [11] umsetzt und löst. Am Ende des Konzeptkapitels wurde dann noch die Benutzeroberfläche mit ihren Interaktionsmöglichkeiten vorgestellt. Im darauf folgenden Kapitel wurden die Implementierungsdetails des entwickelten Tools erläutert. Es fehlt jedoch eine Evaluation der Software, die Auskunft darüber gibt, ob die Software nutzbare Ergebnisse erzeugt.

6.2 Ausblick

Die Forschung auf dem Gebiet der hybriden Entwicklungsansätze ist noch nicht abgeschlossen. Es können jederzeit neue Erkenntnisse gewonnen werden, die zukünftig in die Software eingepflegt werden können. Durch die modulare Programmierung und das Speichern der Entscheidungsprozesse in Konfigurationsdateien, machen Änderungen am bestehenden Modell aber sehr leicht möglich. Durch eine Erweiterung der Entscheidungsprozesse könnte die Software zum Beispiel konkrete Zeitspannen für alle Phasen angeben, anstatt der groben Zeitangaben verkürzt, normal und verlängert. Außerdem könnte das Tool zukünftig mehr Aufgaben innerhalb der Phasen vorschlagen. Denn in der aktuellen Version teilen die Entscheidungsprozesse nur einigen wenigen Phasen spezifische Aufgaben vor.
Eine Evaluation der Software und deren Funktionsweise hat im Rahmen dieser Arbeit nicht stattgefunden. Dies hat vor allem damit zu tun, dass sich während der Entwicklung der Software immer wieder Änderungen am Entscheidungsprozess ergeben haben, die während der parallel laufenden Forschung im Rahmen der Arbeit von Prenner et al. [11] entstanden sind. Somit konnte nicht getestet werden, ob die Software schlussendlich die gewünschten Resultate erzeugt. Diese Evaluation des Tools kann in einer nachfolgenden Arbeit thematisiert werden.

Anhang A

Liste der Kontextfaktoren

Tabelle A.1: Liste aller Kontextfaktoren

Nr.	Einflussfaktor	Mögliche Angaben
E1	Größe der Entwicklung	small scale, large scale oder very large scale
E2	Verteilung des Teams	co-located, gleiche Region, gleiches Land oder global
E3	Anzahl der Stakeholder	1-5, 5-10, 10-15 oder mehr als 15
E4	Verfügbarkeit der Stakeholder	Am Anfang des Projektes, teilweise während Projekt oder immer während Projekt
E5	Wichtigkeit von Langzeitplänen	wenig, etwas oder sehr wichtig
E6	Wichtigkeit von Selbstorganisation der Teams	wenig, etwas oder sehr wichtig
E7	Verteilung von Ressourcen	unsicher, einigermaßen sicher oder sehr sicher
E8	Auswirkung von Fehlern in Software	Komfort des Benutzers, Verlust von wenig Geld, Verlust von viel Geld, Verlust von wenigen Menschen, Verlust von vielen Menschen
E9	Bekanntheit der Anforderungen	gar nicht, wenig, mittel oder vollständig bekannt
E10	Komplexität der Anforderungen	wenig, mittel oder sehr komplex
E11	Wahrscheinlichkeit von Änderungen in den Anforderungen	nicht, mittel oder sehr wahrscheinlich
E12	Komplexität der Entwicklung	wenig, mittel oder sehr komplex
E13	Wichtigkeit früher Ergebnisse	gar nicht, wenig, mittel oder sehr wichtig
E14	Umfang der Anforderungen	wenig, mittel oder vollständig
E15	Erstellung von Anforderungsdokumenten	Ja oder Nein
E16	Umfang der erstellten Architektur	wenig, mittel oder vollständig
E17	Gibt es feste Release Zyklen	Ja oder Nein
E18	Kosten	-

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] V. Belton. A comparison of the analytic hierarchy process and a simple multi-attribute value function. *European Journal of Operational Research*, 26(1):7–21, 1986. Second EURO Summer Institute.

[2] B. Boehm and R. Turner. Using risk to balance agile and plan-driven methods. *IEEE Computer*, 36:57–66, 06 2003.

[3] I. I. Karayalcin. The analytic hierarchy process: Planning, priority setting, resource allocation: Thomas l. saaty mcgraw-hill, new york, 1980, xiii + 287 pages. *European Journal of Operational Research*, 9(1):97–98, 1982.

[4] N. Keshta and Y. Morgan. Comparison between traditional plan-based and agile software processes according to team size project domain (a systematic literature review). In *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 567–575, 2017.

[5] J. Klunder, R. Hebig, P. Tell, M. Kuhrmann, J. Nakatumba-Nabende, R. Heldal, S. Krusche, M. Fazal-Baqaie, M. Felderer, M. F. Genero Bocco, and et al. Catching up with method and process practice: An industry-informed baseline for researchers. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, May 2019.

[6] M. Kuhrmann, P. Diebold, J. Münch, P. Tell, V. Garousi, M. Felderer, K. Trektere, F. McCaffery, O. Linssen, E. Hanser, and C. R. Prause. Hybrid software and system development in practice: Waterfall, scrum, and beyond. In R. Bendraou, F. M. Maggi, D. Raffo, and H. LiGuo, editors, *ICSSP 2017 - Proceedings of the 2017 International Conference on Software and System Process*, volume Part F128767, pages 30–39. Association for Computing Machinery, jul 2017. 2017 International Conference on Software and System Process, ICSSP 2017 ; Conference date: 05-07-2017 Through 07-07-2017.

[7] I. Laux and J. Kranz. Coexisting plan-driven and agile methods: How tensions emerge and are resolved completed research paper. 12 2019.

[8] M. Marinho, J. Noll, I. Richardson, and S. Beecham. Plan-driven approaches are alive and kicking in agile global software development. *CoRR*, abs/1906.08895, 2019.

[9] R. F. Paige, R. Charalambous, X. Ge, and P. J. Brooke. Towards agile engineering of high-integrity systems. In *Proceedings of the 27th International Conference on Computer Safety, Reliability, and Security*, SAFECOMP '08, page 30–43, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] N. Prenner. Towards improving the organization of hybrid development approaches. In *Proceedings of the International Conference on Software and System Processes*, ICSSP '20, page 185–188, New York, NY, USA, 2020. Association for Computing Machinery.

[11] N. Prenner and K. Schneider. A customizable model for the creation of hybrid software development approaches. unpublished, Unpublished.

[12] N. Prenner, C. Unger-Windeler, and K. Schneider. How are hybrid development approaches organized? - a systematic literature review. In *Proceedings of the International Conference on Software and System Processes (ICSSP'20)*, pages 145–154. Association for Computing Machinery, New York, NY, 2020.

[13] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering (ICSE '87)*, pages 328–338. IEEE Computer Society Press, Los Alamitos, CA, 1987.

[14] T. Saaty. Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, 1:83–98, 2002.

[15] T. L. Saaty. How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, 48(1):9–26, 1990. Desicion making by the analytic hierarchy process: Theory and applications.

[16] K. Schwaber. *Agile Project Management With Scrum*. Microsoft Press, USA, 2004.

[17] P. Tell, J. Klünder, S. Küpper, D. Raffo, S. G. MacDonell, J. Münch, D. Pfahl, O. Linssen, and M. Kuhrmann. What are hybrid development methods made of? an evidence-based characterization. *CoRR*, abs/2101.08016, 2021.