ulm university universität
uulm

Fakultät für
Ingenieurwissenschaften
und Informatik
Institut für Datenbanken
und Informationssysteme

Analyse zur Anwendbarkeit von Prozessanpassungsmustern in der Software Engineering Domäne

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Tobias Klein
tobias.klein@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Gregor Grambow

Inhalt

1. Einleitung

Softwareentwicklung ist ein komplexes Feld. Hierbei ist zu differenzieren zwischen Software und Programmen. Programm bedeutet hierbei, dass es sich um eher simplen, überschaubaren Code handelt. Der Produktions- und Wartungsaufwand solcher Programme ist gering da es sich auf wenige Funktionen beschränkt. Software hingegen besteht aus vielen komplexen Strukturen, viel Programmcode, vielen einzelnen Komponenten welche eine hohe Nebenläufigkeit haben können. Dementsprechend ist der Produktions- und Wartungsaufwand bei Software um ein vielfaches höher und komplexer. Ein Projekt welches solche komplexe Software erarbeiten soll ist folglich ebenfalls sehr komplex. Diese Komplexität bringt eine hohe Fehleranfälligkeit mit sich, welche man schon von Beginn des Projektes, also schon bei der Planung, berücksichtigen und minimieren sollte. Dies ist von hoher Wichtigkeit, denn ein Großteil der Kosten eines Softwareprojekts entstehen durch die Fehlersuche sowie deren Behebung. Daher gibt es verschiedenste Methoden, Modelle oder Herangehensweisen welche sicherstellen sollen, dass möglichst wenig Fehler gemacht werden, diese so früh wie möglich erkannt werden (je später entdeckt, desto teurer die Behebung) und darüber hinaus das Endprodukt alle Voraussetzungen erfüllt um den Kunden zufrieden zu stellen. Darüber hinaus wird eine möglichst hohe Effizienz und Effektivität an einen solchen Entwicklungsprozess gestellt. Prozess bedeutet hier die Abfolge verschiedener Arbeitsschritte zur Realisierung des Ziels. Dabei werden ebenfalls Dinge wie die Mitarbeiterzuordnung (wer macht was) und andere Dinge berücksichtigt (siehe auch Kapitel 2.1).

Um solche Prozesse in das Arbeitsumfeld zu integrieren existieren sogenannte Business Process Management Systeme (BPMS). Diese ermöglichen beispielsweise das Modellieren von Prozessabläufen (sog. Templates, also welche Arbeitsschritte folgen in welcher Reihenfolge aufeinander), die Zuordnung von Benutzerrollen für bestimmte Prozessschritte, die Ausführung solcher Templates in sogenannten Prozessinstanzen und vieles mehr (Nicht

alle BPMS unterstützen alle dieser Funktionen). Eine solche softwaregestützte Prozessunterstützung kann, vor allem bei großen Projekten, eine große Erleichterung der Arbeit sein und dabei helfen das jeder Mitarbeiter zu jedem Zeitpunkt den Überblick behalten kann und auch jederzeit weiß was er als nächstes zu tun hat.

Ein Prozess der auf einem solchen Prozessmodell basiert ist in der Regel allerdings sehr statisch, das bedeutet es gibt vordefinierte Arbeitsschritte und Abläufe welche nicht dafür ausgelegt sind einfach abgeändert zu werden, was es schwierig für den Anwender macht sich auf Unregelmäßigkeiten des Ablaufs, Probleme und spontane Änderungen einzustellen. Dies kann aber sehr wichtig sein, da am Anfang in der Planungsphase eines Projektes, welches über mehrere Monate bearbeitet wird, niemand absehen kann welche Arbeitsschritte von Nöten sein werden oder welche unvorhergesehenen Probleme auftauchen werden. Tritt beispielsweise ein Problem während eines Prozessschrittes auf ist es sinnvoll sofort darauf zu reagieren und beispielsweise Qualitätssicherungsmaßnahmen einzufügen oder die Reihenfolge der folgenden Prozessschritte abzuändern.
Im optimalen Fall sollte das System dies selbst erkennen und die passenden Änderungen einpflegen, ohne dass ein Mitarbeiter sich darum kümmern muss. Dies nennt man dynamische Prozessanpassung. Hierbei handelt es sich um ein recht neues Feld der Entwicklung und es existieren nur wenige Umsetzungen einer solchen dynamischen Anpassung.

1.1 Motivation

Arbeitet man heutzutage bei der Softwareentwicklung mit Prozessen und Prozessmodellen, so wird dies meist als feste statische Vorgabe gesehen, das gewählte Modell wird wie aus dem Lehrbuch umgesetzt auch wenn es nicht wirklich zum eigentlichen Projekt passt. Oft jedoch ist der vordefinierte Prozess nicht optimal auf das Projekt angepasst, beziehungsweise man stößt schnell auf Situationen in denen ein statisches Modell nicht ausreicht um optimal weiter arbeiten zu können. In einem ständig wachsenden Markt stehen Unternehmen

heutzutage unter hohem Druck, sich schnell auf Änderungen und neue Situationen einstellen zu können. Man muss schnell und einfach auf alles reagieren können, kostengünstig und schnell produzieren und verkaufen, dabei jedoch sicherstellen dass der Kunde zufrieden ist und seine Anforderungen zu komplett umgesetzt werden. Ebenso wollen Kunden in den Produktionsprozess eingebunden werden. Zum Beispiel findet der Kunde Fehler und wünscht Bugfixes, oder es kommt zu Änderungen seiner Anforderungen welche noch umgesetzt werden sollen etc.. Darüber hinaus soll das Produkt jederzeit stabil und robust laufen.

Software wird in fast allen Domänen (Industrie, Gesundheitswesen, Forschung, etc.) benötigt, und somit sind auch die verschiedenen Anforderungen der Kunden sehr unterschiedlich und vielseitig. Um das Arbeiten mit Prozessen besser unterstützen zu können sowie die Möglichkeit zu bieten diese Prozesse dynamischer gestalten zu können muss also in diese Richtung geforscht und entwickelt werden. Dies dient nicht nur zur Erleichterung der Arbeit, sondern führt ebenfalls zu mehr Effizienz und auch besserer Produktqualität, zwei Punkte welche auf dem umkämpften Markt der Softwareentwicklung sehr wichtig sind um konkurrenzfähig zu sein.

1.2 Zielsetzung

Um Prozessmodelle optimal nutzen, an jedes Projekt und jeden Projektverlauf anpassen zu können, ist eine programmseitige Unterstützung notwendig. Eine solche dynamische Anpassung von Prozessen lässt sich in der Theorie wie auch in der Praxis umsetzen und führt zu besseren, dynamischen und auf Situationen angepassten Prozessen, welche effizient und effektiv arbeiten.

Ziel dieser Arbeit ist es aufzuzeigen, wie sich Prozesse automatisch und dynamisch an die Gegebenheiten des Projektes anpassen lassen und welche Vorteile dies bringt. Gezeigt wird dies hier am Beispiel der Softwareentwicklung und verbreiteten Prozessen dieser Domäne. Die Ergebnisse dieser Arbeit lassen sich jedoch auch auf andere Domänen übertragen, sind also nicht nur der Softwareentwicklung vorbehalten.

1.3 Aufbau der Arbeit

Diese Arbeit besteht insgesamt aus sechs Kapiteln. In Kapitel 2 geht es in mehreren Unterkapiteln um die Grundlagen. Hier soll das nötige Hintergrundwissen vermittelt werden, welches für spätere Kapitel benötigt wird. Es werden zunächst verschiedene Prozessmodelle vorgestellt. Anschließend werden die Grundlagen zu dynamischen Prozessanpassungen, wie etwa Change Patterns vorgestellt und illustriert. Ebenso vorgestellt wird hier das BPMS welches später auch Verwendung findet, die Aristaflow BPM Suite.
Kapitel 3 beschäftigt sich mit den Anforderungen welche man einerseits an die Softwareentwicklung, andererseits spezieller an BPMS und auch diese Arbeit hier stellen kann. In Kapitel 4 werden verschiedene konzeptionelle Ansätze theoretisch erläutert sowie miteinander verglichen und evaluiert. Danach wird eines dieser Konzepte, die Anpassung durch einen Adaptionsworkflow, im Detail erläutert und spezifiziert. Schließlich handelt das 5 Kapitel davon, wie man eben jenes Konzept beispielhaft in die Tat umsetzen kann. Hierzu wird auf mögliche Implementierungen eingegangen und eine spezielle dann im Detail erläutert. Zum Abschluss gibt es in Kapitel 6 eine kurze Zusammenfassung sowie einen kleinen Ausblick auf die weitere Entwicklung. Es folgen noch die schon erwähnten Verzeichnisse.

2. Grundlagen

In diesem Kapitel werden für die Arbeit wichtige Grundlagen erläutert. Zuerst werden dabei verschiedene, in der Softwareentwicklung verbreitete Prozessmodelle vorgestellt und evaluiert (Kapitel 2.1). Anschließend wird das nötige Hintergrundwissen zu dynamischer Prozessanpassung vermittelt (Kapitel 2.2).
Diese Grundlagen dienen dem besseren Verständnis der restlichen Arbeit, da im Verlauf dieser dieses Wissen vorausgesetzt wird.

2.1 Prozessmodelle

In der Softwareentwicklung gibt es viele verschiedene Prozessmodelle und jedes hat seine Vor- und Nachteile. Einige sind zum Beispiel eher für kleine, andere für größere Projekte von Nutzen. Im folgenden findet sich nun eine kleine Auswahl dieser Prozessmodelle. Dabei wird am Anfang das Wasserfallmodell vorgestellt, da dieses die Grundlage für viele andere, komplexe Modelle darstellt. Daraufhin werden einige umfangreichere und komplexere Modelle erläutert, welche heutzutage in der Praxis sehr verbreitet und beliebt sind.

2.1.1 Wasserfallmodell

Das Wasserfallmodell ist ein sogenanntes Phasenmodell [1,2,3]. Erstmals vorgeschlagen wurde es 1970 von Winston Royce [4]. Später wurde es von Barry Boehm etwas erweitert und als Wasserfallmodell bezeichnet [5]. Der Name rührt von der allgemeinen Darstellung, bei der die Phasen in Kaskaden angeordnet werden.
Es ist untergliedert in verschiedene fest definierte Phasen, welche in einer strickten Reihenfolge auszuführen sind. Abgesehen davon ist eine Rückkopplung, also ein Schritt zurück in die vorhergehende Phase, vorgesehen, allerdings jeweils nur zu der direkt vorangegangenen Phase. Diese dient zur Verifikation beziehungsweise der Validierung der vorhergehenden Ergebnisse (in Abbildung 1 angedeutet).

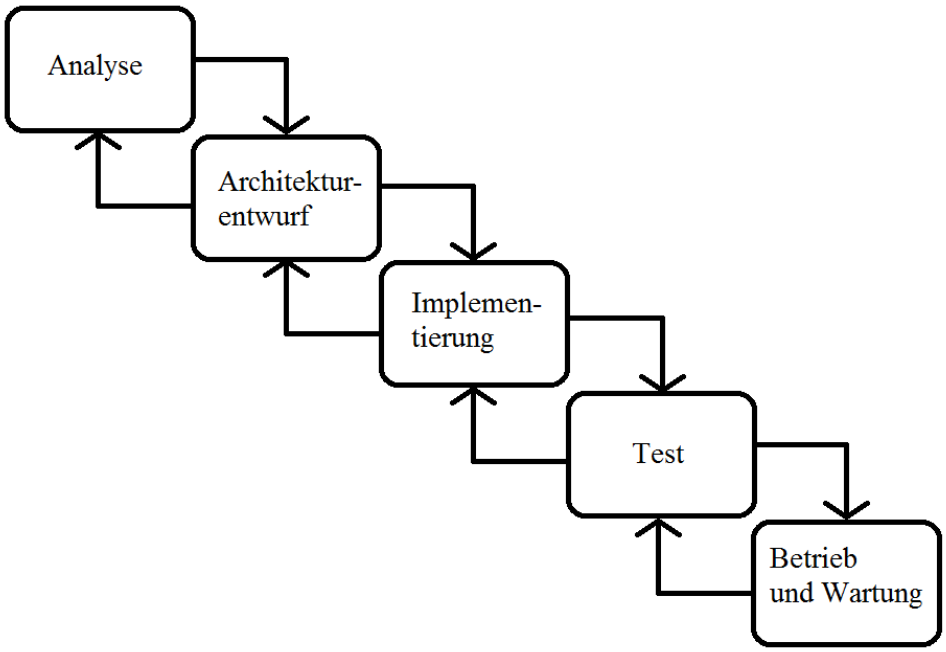

Abbildung 1: Wasserfallmodell nach Boehm

Jede dieser 5 Phasen wird mit einem sogenannten Meilenstein abgeschlossen, welcher als Input für die nachfolgende Phase dient.
Hierbei handelt es sich um ein sehr einfaches Modell, welches allerdings einige Risiken birgt. Durch den strikten Ablauf ziehen sich gemachte Fehler durch alle folgenden Phasen. Wegen der schwachen Rückkopplung lassen sich diese Fehler nur schwer wieder beheben und alle weiteren Phasen werden dadurch verzögert. Ebenso ist es bei diesem Verfahren sehr schwer, im Laufe des Projektes Änderungen (z.B. der Anforderungen) einzupflegen und umzusetzen. Dies kommt in der Praxis häufig vor, selten sind die genauen Anforderungen schon zu Projektbeginn (Analysephase) genau spezifiziert oder bekannt. Die finanziellen Risiken, vor allem bei großen, teuren Projekten sind daher enorm. Je später ein Fehler erkannt wird und je länger er sich schon durch die verschiedenen Phasen gezogen hat, desto teurer wird es, diesen zu beheben [2].
Für große Projekte ist das Wasserfallmodell also nur bedingt geeignet, die Risiken sind zu hoch. Anders sieht es bei kleinen Projekten aus, dort stehen die Vorteile dieses Modells (Einfachheit, wenig Organisationsaufwand, klarer Ablauf) mehr im Vordergrund und das Risiko ist geringer [6].

2.1.2 V-Modell (XT)

Das V-Modell [1, 2, 7, 8], erstmals vorgeschlagen 1979 von Barry Boehm, ist ein Phasenmodell. Es ist dem Wasserfallmodell sehr ähnlich und wenn man es genauer betrachtet eine Weiterentwicklung dessen. Im Unterschied zu diesem besteht das V-Modell jedoch nicht nur aus Entwicklungsphasen, es gibt auch, diesen Gegenübergestellt, Phasen zur Qualitätssicherung und zum Testen. Dadurch ergibt sich dann auch die namensgebende Form (siehe Abbildung 2).

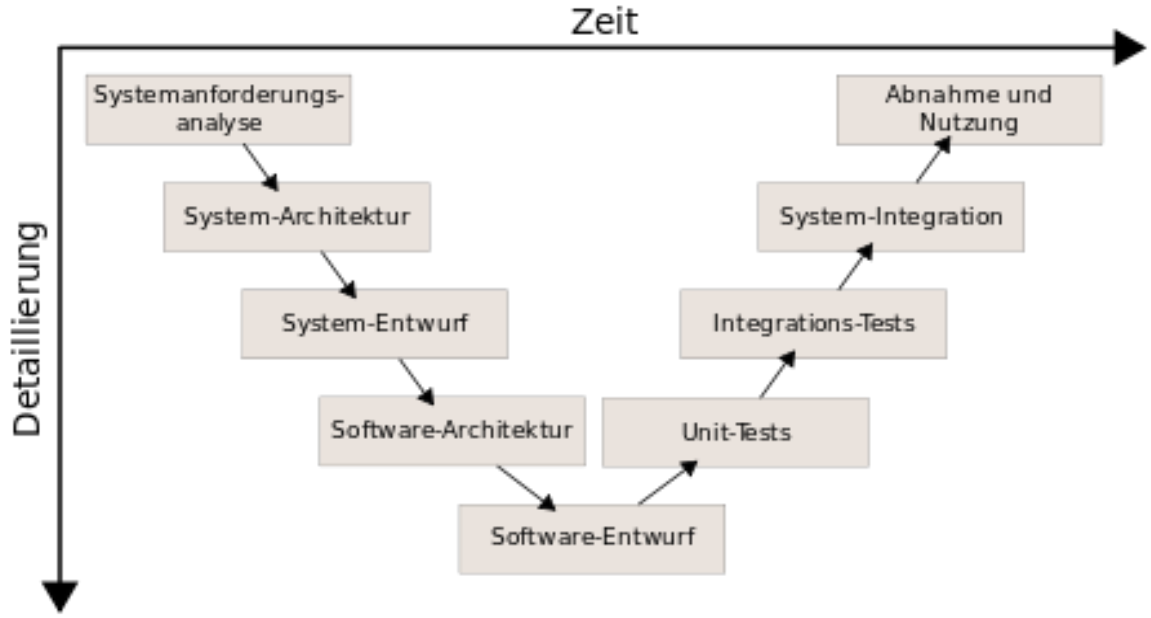

Abbildung 2: V-Modell [9]

Hierbei können nun Fehler genau auf der Abstraktionsebene gefunden werden, auf der sie gemacht wurden. Durch diesen Aufbau werden sämtliche Ebenen auf ihre eigenen Fehler getestet. Die entsprechende Kontrolle steht im Modell auf dergleichen Ebene gegenüber.

Im Laufe der Zeit wurde dieses Modell stetig weiterentwickelt und an den Fortschritt der Branche angepasst. Eine der wichtigsten Versionen ist das V-Model 97. Dieses ist der Entwicklungsstandard für IT-Systeme in Deutschland (teilweise bereits abgelöst durch V-Modell XT, siehe unten).

Anders als beim klassischen Phasenmodell üblich wird hierbei keine zeitliche Abfolge vorgeschrieben, sondern lediglich die Aktivitäten die durchgeführt werden müssen, sowie die Ergebnisse/Produkte (Dokumente etc.) dieser Aktivitäten. Ebenso werden für diese Produkte gewisse Zustände festgelegt.

Diese sind (ohne genauere Erläuterung): Geplant, in Bearbeitung, Vorgelegt, Akzeptiert.

Darüber hinaus umfasst das V-Modell eine Sammlung bestimmter Methoden und Werkzeuge, mit denen die einzelnen Aktivitäten durchzuführen sind.

Das V-Modell lässt sich weiterhin in 4 Submodelle unterteilen, welche sich auf bestimmte Teile des Projekts beziehen und sich ergänzen (siehe dazu Abbildung 3).

Diese 4 Submodelle sind im Detail:

- Systemerstellung (SD) - Hier sind Aktivitäten und Dokumente der Entwicklung zusammengefasst.
- Qualitätssicherung (QA) - Hier wird die Erfüllung der Anforderungen geprüft.
- Konfigurationsmanagement (CM) - beispielsweise die Ablage und Versionskontrolle von Dokumenten
- Projektmanagement (PM) - beispielsweise Steuerung der anfallenden Kosten und Festlegen von Terminen

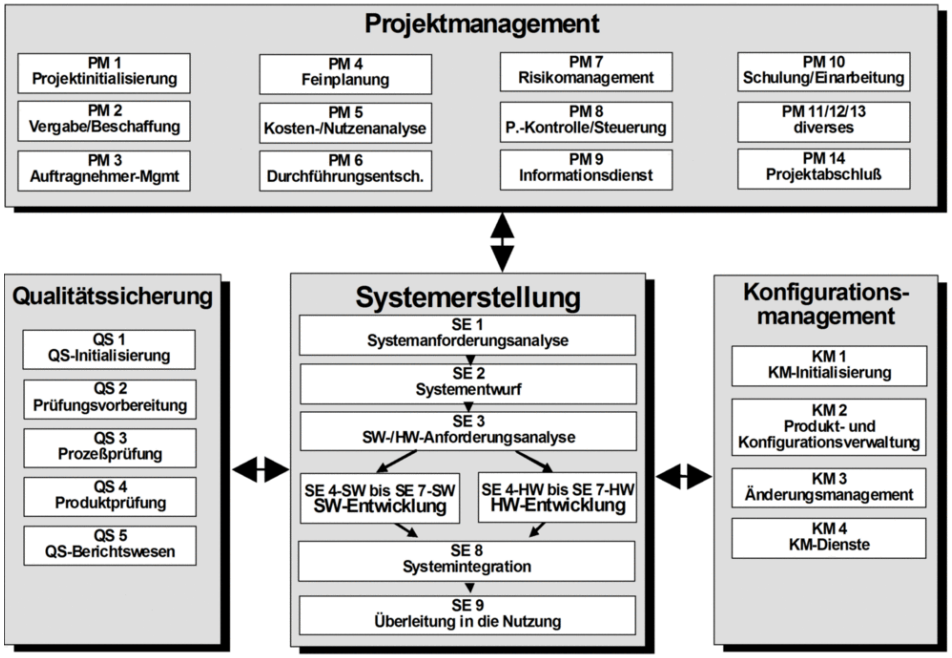

Abbildung 3: Zusammenhang der Subsysteme [10]

Alles in Allem ergibt sich ein allgemeingültiges (Standard-)Modell, welches an die einzelnen Firmen und Projekte angepasst werden kann. Das sogenannte Tailoring, also das "zurechtschneiden" (vom englischen to tailor = schneidern)

der im Projekt vorgesehenen Aktivitäten und Abläufe, dient der Reduzierung unnötiger Dokumentation und Aktivitäten. Nachdem Baukastenprinzip werden somit jene Aktivitäten und Werkzeuge aus dem Standardmodell ausgewählt, die im Projekt benötigt werden und sinnvoll erscheinen. Dies kann vor allem anderen in der Planungsphase geschehen, jedoch kann auch zur Laufzeit des Prozesses eine Anpassung stattfinden.

Das V-Modell XT [2, 3, 7] ist eine Weiterentwicklung des V-Modell 97. Version 1.0 wurde im Februar 2005 veröffentlicht und seit dem stetig weiter verbessert. XT steht hierbei für "Extreme Tailoring", dies verdeutlicht eine der Hauptänderungen im Vergleich zum Vorgänger. Tailoring steht hier weitaus mehr im Mittelpunkt und wird viel mehr angewendet. Für verschiedene Projektgrößen werden sogenannte Streichbedingungen definiert, um, gerade bei kleineren und mittelgroßen Projekten die Aktivitäten und Produkte (= Ergebnisse der einzelnen Aktivitäten) auf ein möglichst geringes Maß zu reduzieren.
Außerdem beschränken sich die Vorgaben des Modells nicht mehr nur auf den Auftragnehmer, also den der das Projekt realisiert, sondern auch auf den Auftraggeber. Dieser muss nun auch seine benötigten Aktivitäten selektieren (Baukastenprinzip) und die geforderten Produkte abliefern. Dies führt zu einer besseren Zusammenarbeit und besserem Verständnis unter den beteiligten Parteien.
Die oben genannten vier Submodelle (SD, QA, CM, PM) existieren in dieser Form nicht mehr, sondern nur noch gewisse Vorgehensbausteine. Aus diesen wird das konkrete Vorgehensmodell für ein Projekt zusammengebaut (Tailoring). Dabei wird jedoch keinerlei Aussage über die zeitliche Abfolge dieser Bausteine gemacht. Einzig die Produkte sind hier entscheidend. Insgesamt gibt es 22 dieser Bausteine, wovon bestimmte davon bei gewissen Projekten verpflichtend sind.

Das V-Modell XT ist, wie schon sein Vorgänger, ein sehr ausführliches Modell. Es werden alle Punkte die bei einem Softwareentwicklungsprozess eine Rolle spielen, abgedeckt um die Kontrolle zu behalten und hohe Produktqualität sicherzustellen. Dinge wie beispielsweise das Projekt- und

Konfigurationsmanagement sind ebenso enthalten wie die eigentliche Systemerstellung. Es ist ein Industriestandard und somit sehr weit verbreitet. Dies liegt auch daran, dass es nicht nur für große Projekte, sondern dank der Anpassungsmöglichkeiten (Tailoring) auch für kleinere Projekte anwendbar ist. Die stetige Weiterentwicklung und Anpassung an neue Technologien etc. stellt einen großen Vorteil dar.

Als negative Aspekte muss man hier erwähnen, dass ein solches Modell ohne die geeigneten Werkzeuge und ein gewisses Vorwissen der Anwender nicht umzusetzen ist. Solche Werkzeuge existieren allerdings und sind ohne Probleme zu erhalten [11]. Außerdem besteht die Gefahr für kleine und mittlere Systeme, dass das Tailoring zu wenig oder falsch eingesetzt wird und somit im Grunde "zu viele" Aktivitäten ausgeführt werden und das Projekt komplizierter wird als es eigentlich ist.

2.1.3 Unified Process (UP)

Der Unified Process [12] wurde parallel zu der bekannten Unified Modelling Language (UML) entwickelt. Er ist sozusagen ein Metamodell für Vorgehensmodelle der Softwareentwicklung. Er basiert auf den folgenden drei Grundprinzipien: (1) Im Zentrum der Planung eines Projekts soll die Architektur stehen, (2) realisiert wird es durch ein inkrementelles und iteratives Vorgehen und (3) mit die wichtigsten Komponenten sind hierbei die Anwendungsfälle des Produkts.

Umsetzungen des Unified Process sind beispielsweise OpenUP (Open Source Version entwickelt von der Eclipse Foundation) sowie der kommerzielle Rational Unified Process (RUP)

1998 wurde der Rational Unified Process (RUP) erstmals von Philippe Kruchten vorgestellt. RUP ist ein Prozessframework auf der Basis der Unified Modelling Language und realisiert 6 der wichtigsten Best Practices (Erfolgsmethoden) der modernen Softwareentwicklung [12].

- Iterative Entwicklung: Vorteil gegenüber linearen Modellen bei der Anpassung an Änderungen beispielsweise der Anforderungen.
- Qualitätsmanagement: Fehler sollen früh erkannt und behoben werden.

- Komponentenbasierte Struktur: Unabhängiges entwickeln der einzelnen Teile, spätere Wiederverwendbarkeit ist hierbei wichtig.
- Visualisierung: Diese sorgt für ein besseres Problemverständnis und erleichtert Kommunikation mit anderen Beteiligten (in diesem Fall realisiert durch UML).
- Änderungsmanagement: Es gibt eine strikte Versionskontrolle, Änderungen müssen nachvollziehbar, alte Versionen reproduzierbar sein
- Anforderungsmanagement: Die Anforderungen an das Produkt sowie die Entwicklung werden stets aktuell und im Fokus gehalten. Dadurch erreicht man eine erhöhte Produktqualität sowie Kundenzufriedenheit

Zeitlich unterteilt sich RUP in 4 Phasen:

- Konzeptionsphase
- Entwurfsphase
- Konstruktionsphase
- Übergabephase

Dies sind die sogenannten dynamischen Aspekte. Orthogonal dazu werden sie statischen Aspekte angeordnet (oft auch Disziplinen genannt):

- Geschäftsprozessmodellierung
- Anforderungsanalyse
- Analyse und Design
- Implementierung
- Test
- Auslieferung

Dynamische sowie statische Aspekte ergeben nun zusammen den gesamten RUP. Die verschiedene Aspekte und Disziplinen laufen zeitlich parallel und unterstützend zusammen. Dabei sind bestimmte Disziplinen in einzelnen oder mehreren Phasen mehr oder weniger stark vertreten und von Relevanz.
Darüber hinaus existieren weitere unterstützende Arbeitsschritte, welche von den Phasen unabhängig sind. Das Konfigurations- und Änderungsmanagement, das Projektmanagement sowie die Infrastruktur. Den endgültigen Zusammenhang zeigt die folgende Abbildung.

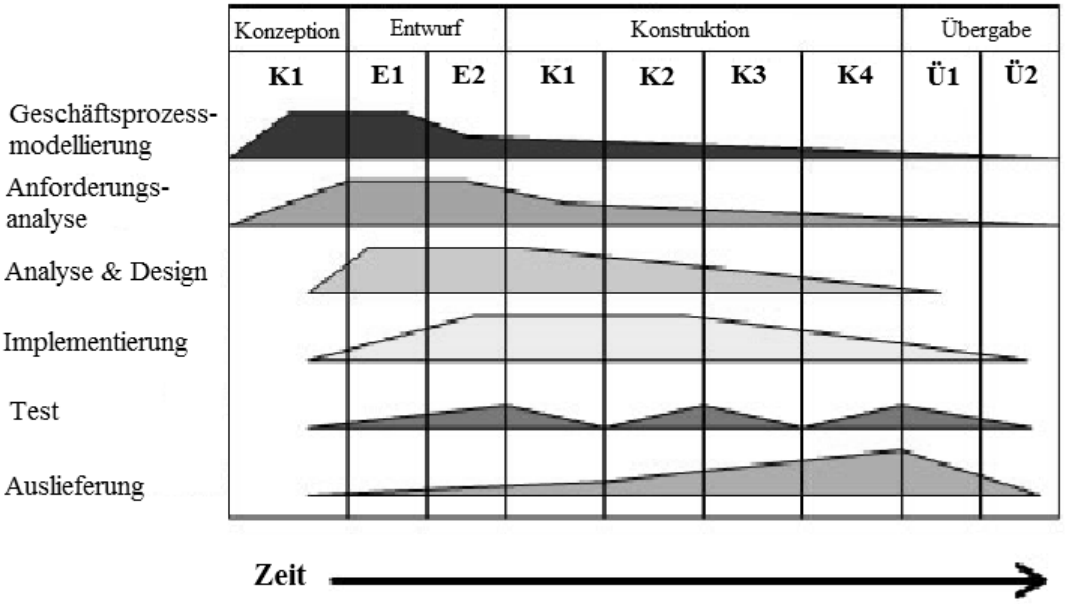

Abbildung 4: RUP Aspekte

Jede Iteration, also jeder weitere Durchlauf, dieser 4 Phasen wird mit einem vordefinierten Meilenstein abgeschlossen.

RUP ist ein aktivitätsgetriebenes Modell, das heißt bestimmte Rollen führen ihre zugeordneten Aktivitäten genauso aus wie diese definiert sind. Dies unterscheidet es z.B. vom V-Modell XT, welches die Produkte in den Vordergrund stellt. Dabei ist der genaue (zeitliche) Ablauf strikt einzuhalten.

Eine Teilmenge des RUP wird realisiert durch den sogenannten Open Unified Process (OpenUP). Dies ist ein Framework der Eclipse Foundation für ihre Entwicklungsumgebung [14]. Hierbei werden verschiedene Praktiken des RUP umgesetzt, wie z.B. inkrementelle/iterative Entwicklung, ein architekturgetriebenes Vorgehen, etc..

2.1.4 Scrum

Scrum [3, 15, 17] ist eine sogenanntes agiles Modell. Das bedeutet man versucht mit geringem bürokratischen Aufwand, wenigen Regeln und meist einem iterativen Vorgehen auszukommen. Hierbei setzt man anderen Fokus auf das Projekt und richtet sich nach dem Manifesto for Agile Software Development [16]:

"Individuals and interactions over processes and tools;
Working software over comprehensive documentation;
Customer collaboration over contract negotiation;
Responding to change over following a plan

Die Grundidee von Scrum ist es, dass komplexe Projekte nicht von Anfang an komplett durchschaut und somit alle Anforderungen richtig erkannt werden können. Um dieses Problem zu lösen gibt es ein sogenanntes Product Backlog. Hier werden sämtliche Anforderungen, welche Umgesetzt werden sollen, gesammelt. Dieses wird stetig erweitert und an neue Erkenntnisse angepasst.

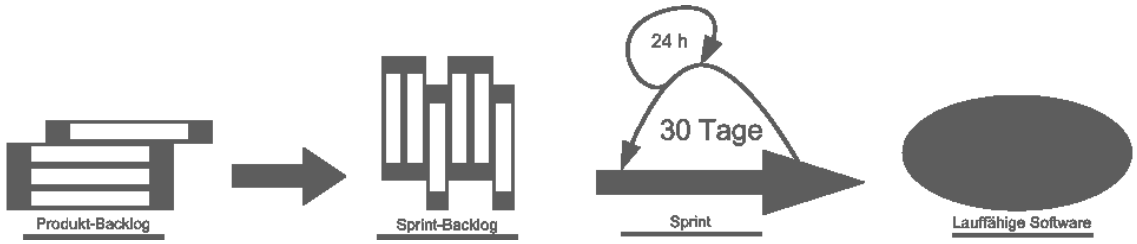

Abbildung 5: Ablauf bei Scrum

Wie in Abbildung 5 veranschaulicht, werden aus diesem gewisse Anforderungen herausgenommen und in sogenannte Sprint Backlogs eingepflegt. Diese stellen die Basis und die Aufgaben für den nun folgenden Sprint (eine Entwicklungsphase von beispielsweise 30 Tagen) dar, bei dem das Team versucht die gewählten Anforderungen umzusetzen. Den Entwicklern wird hierbei weitestgehend "freie Hand" gelassen. Täglich findet ein Treffen statt, bei dem jeder seine Fortschritte sowie den Plan für den Tag erläutert. Am Ende des Sprints soll ein lauffähiges System existieren, welches von

Auftraggeber begutachtet wird, um Änderungen, Verbesserungen etc. in der nachfolgenden Iteration mit einfließen lassen zu können.

2.2 Darstellung von Prozessen

Prozesse werden meist als Graphen dargestellt. Diese sind für den Betrachter einfach zu interpretieren und zu verstehen, selbst wenn der Prozess komplex ist. Für die Modellierung stehen verschiedene Elemente zur Verfügung, die sich rein von der Darstellung in den verschiedenen BPMS oder Modellierungswerkzeugen unterscheiden können, jedoch aber meist dieselbe Bedeutung haben. In dieser Arbeit wird mit Aristaflow (siehe Kapitel 2.3.5) gearbeitet, daher wird auch die Darstellung dieses Systems verwendet.

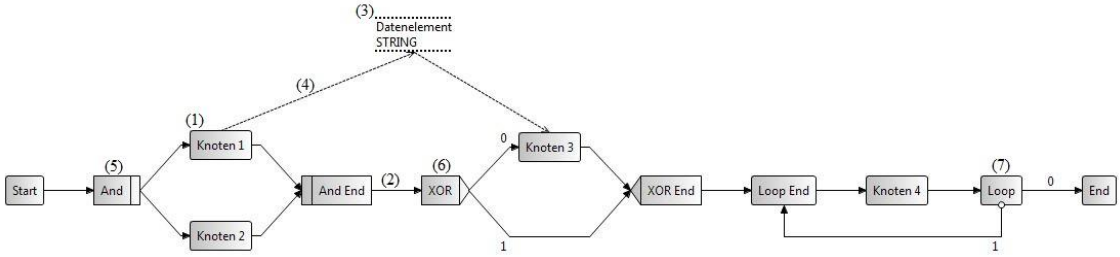

Abbildung 6: Modell eines Prozesses

Wie Abbildung 6 zeigt besteht ein Prozess aus mehreren Knoten (1). Diese stehen für Aktivitäten, welche während des Prozesses ausgeführt werden sollen. Dabei handelt es sich um die symbolische Darstellung von realen Arbeitsvorgängen wie beispielsweise die Verarbeitung von Waren oder auch die Ausführung eines Programms. Die Reihenfolge der Abarbeitung dieser Aktivitäten wird durch die sogenannten Kontrollflusskanten (2) gesteuert bzw. symbolisiert. Diese Pfeile geben also den zeitlichen Ablauf der Prozessschritte wieder. Ein weiterer wichtiger Teil sind die Datenelemente (3). Diese repräsentieren Daten (String, Integer, Boolean etc.) welche von Aktivitäten entweder geschrieben oder gelesen werden. Sie sind also entweder der Input, oder der Output eines Knotens bzw. der Aktivität. Welcher Knoten ein Datenelement schreibt und welcher eines liest wird durch die Datenflusskanten (4) dargestellt. Ob es sich um einen Schreib- oder Lesevorgang handelt erkennt

man hier an der Pfeilrichtung (in der Abbildung schreibt Knoten 1 und liest Knoten 3). Ferner existieren noch spezielle Elemente zur Steuerung des weiteren Kontrollflusses. Zuerst einmal sind dies sogenannte And-Knoten (5). Nach diesen werden alle folgenden Elemente parallel zueinander ausgeführt. Im Falle von Aristaflow wird jeder And Split wieder mit einem And Join (siehe Abbildung) geschlossen, dies dient dazu die Ausführung des Prozesses eindeutig modellieren zu können und Fehler zu vermeiden. Es ergibt sich demnach also eine gewisse Blockstruktur innerhalb eines Prozesses. Diese Blöcke sind in sich schlüssig und übersichtlich. Nicht alle Systeme sind dahingehend so streng. Ähnlich wie der And-Knoten funktioniert auch der XOR-Knoten (6), allerdings wird hier der nachfolgende Teil nicht parallel ausgeführt, sondern es findet eine Fallunterscheidung statt und nur einer der möglichen Pfade kommt zur Ausführung. Umgangssprachlich würde man sagen es wird ENTWEDER der eine Pfad ODER der andere gewählt. Schließlich gibt es noch die Loop-Knoten (7). Auch hier findet eine Fallunterscheidung statt, allerdings führt hier ein Pfad sozusagen zurück zu vorher bereits bearbeiteten Aktivitäten um eine Wiederholung realisieren zu können. XOR und Loop werden ebenfalls beide auch wieder mit dem entsprechenden End-Knoten abgeschlossen wie schon beim And erwähnt.

2.3 Prozessanpassung

Zum Thema Prozessanpassungen gibt es verschiedene theoretische Grundlagen, welche im Folgenden vermittelt werden. Hierbei wird auf verschiedene Anpassungsmöglichleiten eingegangen und es wird das System Aristaflow vorgestellt, welche für diese Arbeit Anwendung findet.

2.3.1 Definition

Eine Prozessanpassung kann einerseits von Nöten sein wenn sich der reale Prozess, also der Ablauf der Arbeitsschritte in der realen Welt (beispielsweise der Produktionsablauf eines Produkts) ändert, andererseits müssen Prozesse

angepasst werden, wenn beispielsweise ein Schritt fehlschlägt oder nicht zu einem befriedigenden Ergebnis führt. Solche Anpassungen können zum Beispiel das Einfügen von Qualitätssicherungsmaßnahmen sein, also zusätzliche Tests und Aktivitäten welche vorher nicht eingeplant waren, eine Wiederholung von bereits abgeschlossenen Aktivitäten oder aber auch das Entfernen von späteren Aktivitäten, welche eventuell überflüssig werden oder ähnliches.

Prozessanpassungen können auf zweierlei Arten stattfinden. Einerseits auf Prozessebene, andererseits auf Instanzebene. Ein Prozess kann mehrere Instanzen, also praktisch zur Ausführung gebrachte und laufende Einheiten eines Prozesses, haben, was bedeutet das eine Änderung auf dieser Ebene alle Instanzen dieses Prozesses beeinflusst, das heißt wird der zugrunde liegende Prozess verändert werden auch sämtliche Instanzen dieses Prozesses diese Änderungen übernehmen. Dies kann zu Problemen führen (siehe 2.2.3). Änderungen auf Instanzebene sind eher unkritisch, jedoch gibt es dort ebenfalls eine Aspekte welche beachtet werden müssen, um einen gültigen Prozess zu erhalten.

Wenn eine solche Prozessanpassung nicht per Hand von einem Menschen durchgeführt wird sondern automatisch, etwa von einem Programm, so spricht man von automatischer Prozessanpassung. Welche Anpassungen hierbei automatisch durchgeführt werden können kann in sogenannten Patterns definiert werden, siehe dazu Kapitel 2.2.2.

2.3.2 Adaption Patterns

Adaption Patterns [18] sind sozusagen Vorlagen dafür, welche Prozess-Anpassungen man anwenden kann und wie diese aussehen sollten. Werden bestimmte Bedingungen eingehalten so ist sichergestellt, dass nach Anwendung solcher Anpassungen die Korrektheit des Prozesses erhalten bleibt. Diese Adaptionen lassen sich sowohl auf Prozess-, wie auch auf Instanzebene anwenden.

Hier eine Auswahl von Adaption Patterns, welche zum Thema der Arbeit passen, vorgestellt und erklärt.

Es existieren weitaus mehr Patterns als die hier vorgestellten, diese Auswahl beschränkt sich auf die grundlegendsten (Die Nummerierung richtet sich nach [18]).

Insert Process Fragment (AP1)

Hierbei handelt es sich um das Einfügen von einem Knoten (atomare Aktion) oder einer komplexen Aktivität (zum Beispiel ein Subprozess) in einen Prozess oder eine Prozessinstanz. Zu unterscheiden ist hierbei, ob die Aktivität direkt zwischen zwei Aktivitäten, parallel zu einer Anderen oder nur bedingt eingefügt wird. Bei der Softwareentwicklung könnte man damit zum Beispiel Qualitätssicherungsmaßnahmen in den Prozess einfügen.

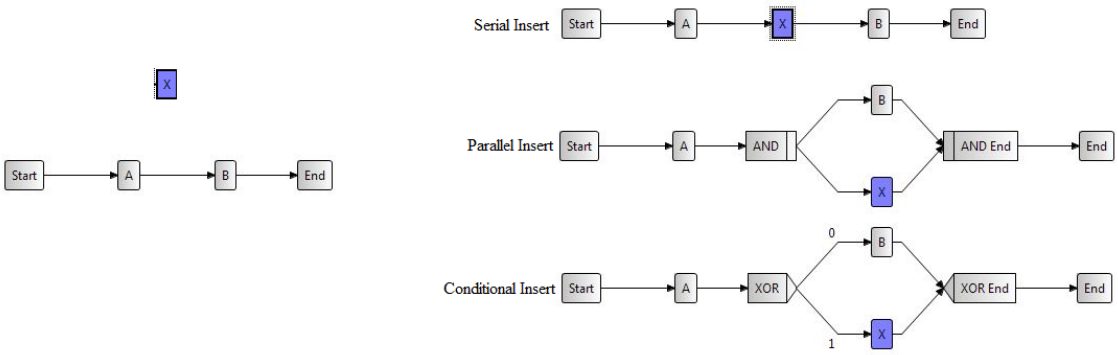

Abbildung 7: Insert Pattern

Delete Process Fragment (AP2)

Das Gegenteil Einfügung ist das Löschen einer Aktivität. Dies kann beispielsweise sinnvoll sein, wenn geplante Aktivitäten unnötig werden oder eine Aktivität nicht ausgeführt werden soll. Realisiert werden kann dies durch verschiedene Ansätze. Entweder man löscht die Aktivität tatsächlich aus dem Prozess oder der Instanz, oder man ersetzt sie durch eine "stille Aktivität", welche sozusagen nichts macht, oder man setzt eine Bedingung vor die Aktivität welche nicht erfüllt ist und die eigentliche Aktivität somit nicht zur Ausführung kommt. Ein Anwendungsfall in der Softwareentwicklung wäre zum Beispiel das Löschen von Debugging Aktivitäten, wenn diese unnötig werden weil sie schon früher als geplant ausgeführt wurden.

Abbildung 8: Delete Pattern

Move Process Fragment (AP3)

Das Bewegen einer Aktivität an eine andere Stelle des Prozesses kann durch Zusammenspiel von Insert und Delete realisiert werden. Auch hierbei kann wie bei der Einfügung zwischen direktem, parallelem und bedingtem Verschieben unterschieden werden. Nötig ist ein solches Verschieben wenn eine Aktivität zum vorbestimmten Zeitpunkt gar nicht oder nicht vollständig ausgeführt werden kann. In der Softwareentwicklung wäre dies zum Beispiel der Fall, wenn benötigter Programmcode aus einer vorherigen Aktivität nicht funktioniert oder zum vorgesehenen Zeitpunkt noch nicht fertig ist.

Abbildung 9: Move Pattern

Replace Process Fragment (AP4)

Auch das Ersetzen einer Aktivität durch eine andere kann durch Insert und Delete erfolgen. Benutzt werden kann das Ersetzen zum Beispiel wenn durch Änderungen an den Anforderungen oder des Prozesses bestimmte Aktivitäten nicht mehr ihren Zweck erfüllen und somit durch andere, passendere Aktivitäten ersetzt werden. Beispielsweise soll statt einer Programmfunktion zuerst das Interface entwickelt werden um dem Kunden schon früher Ergebnisse zeigen zu können.

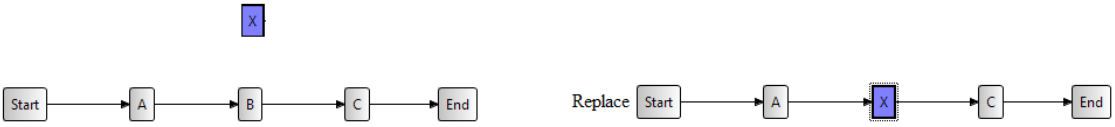

Abbildung 10: Replace Pattern

Swap Process Fragment (AP5)

Zwei existierende Aktivitäten oder Fragmente (mehrere zusammenhängende Aktivitäten) werden innerhalb des Prozesses ausgetauscht. Dabei müssen diese nicht unbedingt direkt aufeinander folgen. Umsetzen lässt sich dieses Pattern entweder durch Anwendung von AP3, oder wiederum durch das Anwenden von AP1 und AP2. Zur Anwendung kommt dieses Pattern falls die vordefinierte Reihenfolge der Aktivitäten nicht mehr zum gewünschten Ergebnis führt, oder wenn sich durch den Tausch Vorteile, wie zum Beispiel Zeitersparnis, herbeiführen lassen.

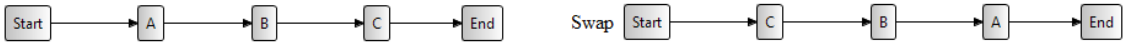

Abbildung 11: Swap Pattern

Update Condition (AP13)

Dieses Pattern erlaubt das Ändern einer Bedingung von Verzweigungen. Dazu muss sichergestellt werden, dass alle erforderlichen Attribute für die Entscheidung im Prozess (oder der Instanz) vorhanden und abrufbar sind. Eine solche Anpassung wäre beispielsweise nötig wenn sich die Menge der benötigten Ressourcen für einen Produktionsprozess ändert.

2.3.3 Automatische Prozessanpassung

Automatische Prozessanpassungen sind noch sehr neu und finden heutzutage noch kaum Anwendung in den gängigen BPMS. Gemeint ist damit, dass Prozessanpassungen nicht von Hand von einem Menschen durchgeführt werden, sondern automatisiert und selbstständig von einem Programm. Dies spielt vor allem bei dynamischen Anpassungen (also zur Laufzeit) eine interessante Rolle, da solche automatischen Anpassungen sehr viel schneller und effizienter durchgeführt werden können. Damit dies funktionieren kann benötigt das Programm, welches für die Anpassung zuständig ist, verschiedenste Informationen über die durchzuführenden Adaptionen sowie den aktuellen Status und Aufbau des Prozesses.

Der Vorteil einer solchen Automatisierung liegt einerseits darin, dass Programme sehr viel mehr Informationen in viel kürzerer Zeit verarbeiten können, andererseits lässt sich dadurch sicherstellen, dass keine Flüchtigkeitsfehler gemacht werden oder relevante Informationen nicht beachtet werden weil diese bei der Bearbeitung durch einen Menschen verloren gehen. Diese Vorteile sind vor allem bei großen, komplexen Prozessen gravierend, da bei zunehmender Prozessgröße der Mensch an seine Grenzen stößt.
Durch eine solche automatische Anpassung lässt sich der Adaptionsvorgang effizienter und kostengünstiger (weniger Arbeitszeit und -aufwand) gestalten und es werden bessere Ergebnisse erzielt.

2.3.4 Probleme

Beim Modellieren von Prozessen wie auch bei Ad hoc Änderungen (Änderungen welche spontan und zur Laufzeit durchgeführt werden) von bestehenden Prozessen können viele Fehler gemacht werden. Diese können zu fehlerhaften Datensätzen, ungewünschten Nebeneffekten und falschen Ergebnissen führen. Wird beispielsweise eine Kontrollkante vergessen, so entsteht ein völlig anderer Prozessablauf. Ebenso kann es leicht zu sogenannten Deadlocks führen, das bedeutet der Prozess steckt an einer Stelle fest und kann nicht weiter fortgesetzt werden. Auch das Gegenteil, ein sogenannter Lifelock, ist möglich. Hier befindet sich der Prozess in einer Endlosschleife und kommt niemals zum gewünschten Ende.
Ein weiteres Problem ergibt sich durch die Datenflüsse. Diese müssen stets konsistent gehalten werden und auch auf den Kontrollfluss angepasst werden (und umgekehrt). Benötigt zum Beispiel eine Aktivität bestimmte Eingabeparameter, welche aber erst in einer späteren Aktivität erstellt werden, kann diese nicht zur Ausführung kommen. Ähnliches gilt, wenn ein Knoten zur Laufzeit entfernt wird, dieser aber wichtige Daten für andere Aktivitäten geliefert hat.
Darüber hinaus kommen noch mehr Probleme hinzu, wenn man von einer automatisierten dynamischen Anpassung und Modellierung ausgeht.

Mit Hilfe von oben genannten Patterns und auch den nicht näher vorgestellten kann man festlegen welche Funkionen eine dynamische Lösung anbieten muss, jedoch stellt sich die Frage wann und wo ist welches dieser Patterns anzuwenden, und darüber hinaus wie kann man sicher stellen, dass die Anpassungen auch richtige Prozesse produzieren? Die Antwort auf diese Fragen können unterschiedlich ausfallen.

Entweder muss man eine solche Korrektheit selbst im Programm prüfen indem man die Prozesse analysiert und validiert, oder man braucht eine Modellierungs- und Ausführungsumgebung welche das selbstständig übernimmt und nur solche Änderungen durchführen lässt, welche richtig sind (siehe dazu auch [20, 21, 22]).

Einen solchen Ansatz verfolgt beispielsweise Aristaflow (siehe nächstes Kapitel). Dieses ist ein Prozessmanagement System welches die eben angesprochenen Ad hoc Änderungen zulässt und gleichzeitig die Korrektheit des Prozesses sicherstellt.

2.3.5 Aristaflow BPM Suite

Aristaflow BPM Suite [19] (BPM = Business Process Management) ist ein Tool zur Modellierung, Ausführung und Überwachung von Prozessen. Aristaflow arbeitet mit einer mächtigen Fehlererkennung und Prozessanalysen zu Modellierungs- wie auch Laufzeit. Ferner folgt es dem "Correctness by Construction" Prinzip, welches einzigartig ist unter den gängigen Konkurrenzprodukten.

Correctness by Construction

Die Korrektheit eines Prozesses wird dadurch sicher gestellt, dass zu jeder Zeit der Modellierung nur diejenigen Aktionen zur Verfügung stehen, welche wiederrum einen korrekten Prozess erzeugen [19].

Dadurch wird von Anfang an ausgeschlossen, dass strukturelle oder datenbezogene Fehler überhaupt gemacht werden.

Die Vorteile eine solchen Prinzips liegen klar auf der Hand. Die Zeit die für Tests, Debugging und Fehlersuche verwendet werden muss wird auf ein

Minimum reduziert. Außerdem wird die Prozessqualität durch einwandfreie Modellierung erhöht.
Ebenso werden nur Ad hoc Änderungen zugelassen, welche einen Korrekten Prozess erzeugen, dementsprechend muss man die nicht mehr im Programm selbst überprüfen.

Die Aristaflow BPM Suite [19] besteht aus mehreren Komponenten, diese werden im Folgenden kurz erläutert:

Process Template Editor

Mit dem Process Template Editor (siehe Abbildung 12) lassen sich Templates (also die Abbildung von Prozessen) modellieren. Dies geschieht dabei unter Beachtung der Correctness by Construction, wie eingehend beschrieben. Zusätzlich wird, zur Modellierungszeit, nebenläufig geprüft ob andere Verstöße vorliegen, welche einen fehlerhaften Prozess ergeben. So werden zum Beispiel sämtliche Datenflusselemente etc. überprüft.
Der Editor an sich ist sehr simpel aufgebaut. Am rechten Rand bekommt man jederzeit angezeigt, welche Änderungen sich bei gewählter Pre-und Postselection durchführen lassen. Damit lassen sich zwei Elemente markieren welche dann als Ausgangspunkt für die gewünschten Aktionen dienen, beispielsweise das Einfügen eines neuen Knotens zwischen den beiden markierten. Am linken Rand findet man eine Liste von Aktivitäten, welche sich einfach per Drag and Drop auf Prozessknoten legen lassen [20]. Schließlich sieht man in der Mitte jederzeit den modellierten Prozess, ergänzt durch weitere Informationen über die aktuell ausgewählten Elemente (links unten).

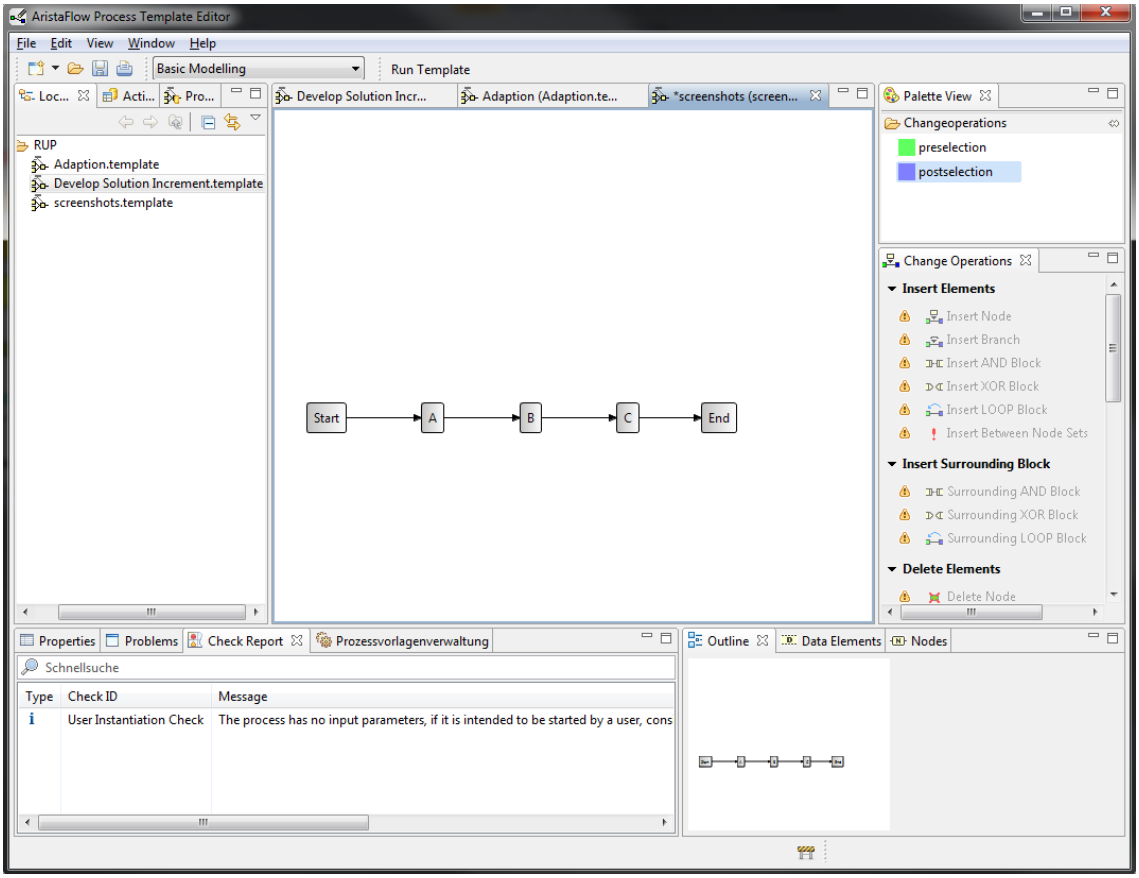

Abbildung 12: Process Template Editor

Activity Repository Editor

Mit dem Activity Repository Editor lassen sich Aktivitäten erstellen, bearbeiten und in das (Aristaflow-)System einpflegen (zum Beispiel damit sie in der Liste der möglichen Aktivitäten des Template Editors auftaucht, siehe oben). Dies geschieht einfach über typspezifische Formulare (siehe Abbildung 13, rechter Teil), welche entweder komplett ausgefüllt, oder aber auch mit fehlenden Informationen belassen werden können, welche dann später im Modellierungsvorgang ergänzt werden. Die hier erstellten Aktivitäten können nun im Template Editor ausgewählt, und auf Knoten gezogen werden. Wie genau eine solche Aktivität aussehen kann bzw. was diese tut ist dem User überlassen. Zum Beispiel gibt es Aktivitäten zum Ausführen von externen Programmen, Formulare für die Eingabe von Informationen etc..

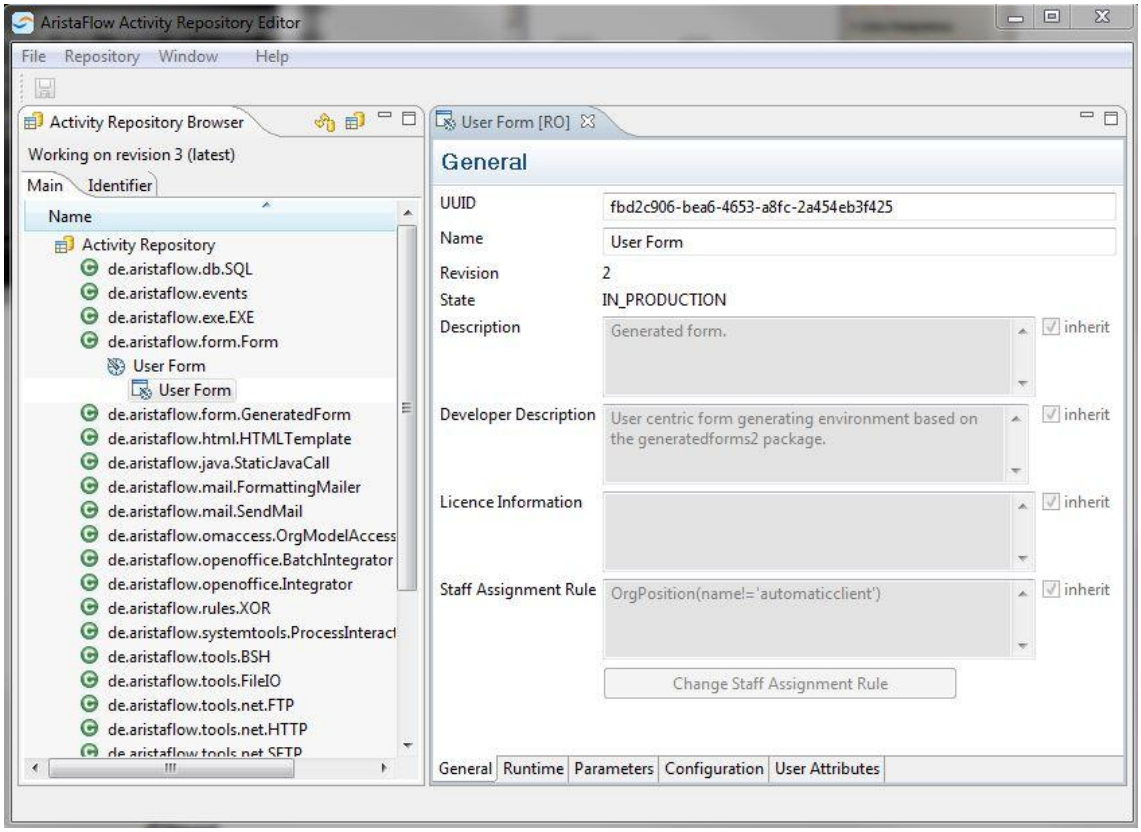

Abbildung 13: Activity Repository Editor

OrgModel Editor

Mit dem OrgModel Editor lassen sich Organisationsstrukturen erstellen und bearbeiten. Dies ist insofern wichtig, da Aristaflow mächtige Bearbeiterzuordnungsregeln besitzt und sich somit jeder Aktivität bestimmte Rollenvoraussetzungen etc. zuweisen lassen (sogar zwingend erforderlich). Das bedeutet also, dass beim modellieren mit dem Template Editor für jeden Knoten, und die damit verbundene Aktivität, festgelegt wird wer genau diese Aktivität ausführen kann oder muss. Dies können bestimmte Geschäftspositionen sein wie zum Beispiel alle Abteilungsleiter, oder aber auch bestimmte Personen wie zum Beispiel Herr Müller aus der IT-Abteilung. Der Editor ist simpel aufgebaut (siehe Abbildung). In der Mitte bekommt man die jeweiligen Entitäten zur gewählten Option angezeigt, rechts wiederrum nähere Details zur gewählten Entität. Links befindet sich eine Liste der verschiedenen Entitätstypen wie zum Beispiel Organisationseinheiten, Projektgruppen etc..

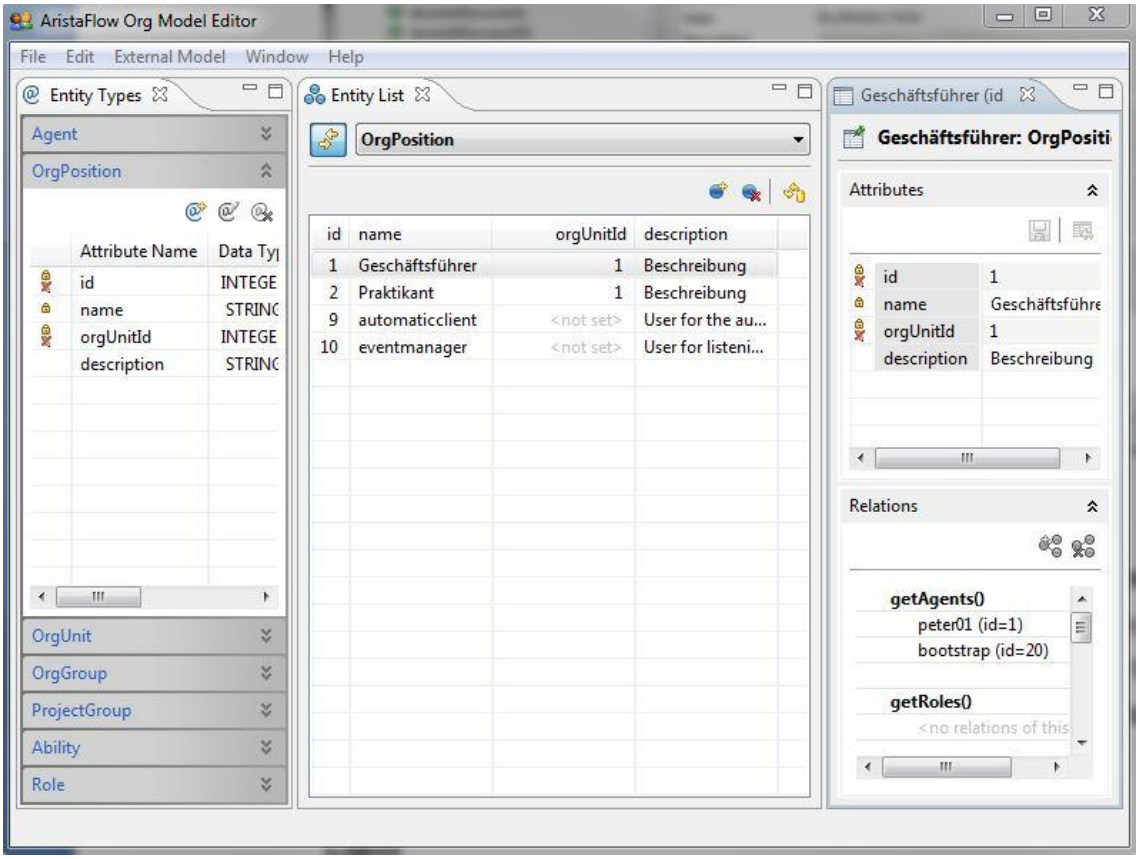

Abbildung 14: OrgModel Editor

TestClient

Mit dem TestClient lassen sich Templates ausführen und somit testen. Dies findet lokal und nicht, wie eigentlich der Fall, auf dem Aristaflow Server statt. Sämtliche Aktivitäten, welche eventuell noch nicht vollständig modelliert oder implementiert sind, werden hierbei durch Standardformulare (zum Beispiel ein Interface für die Ein- und Ausgabe) ersetzt. Somit lässt sich ein Prozess auch schon durchspielen bevor er vollständig lauffähig modelliert ist. Dadurch wird das Modellieren der Templates beschleunigt und erleichtert.

Das wichtigste Element des TestClients ist die Arbeitsliste (siehe Abbildung 15). Hier wird jeweils der nächste anstehende Arbeitsschritt angezeigt und kann auch von dort aus gestartet werden. Weiter unten bekommt man verschiedene Attribute zum jeweiligen Arbeitsschritt angezeigt, sowie die Ein- oder Ausgabeformulare sofern es sich um eine solche Aktivität handelt.

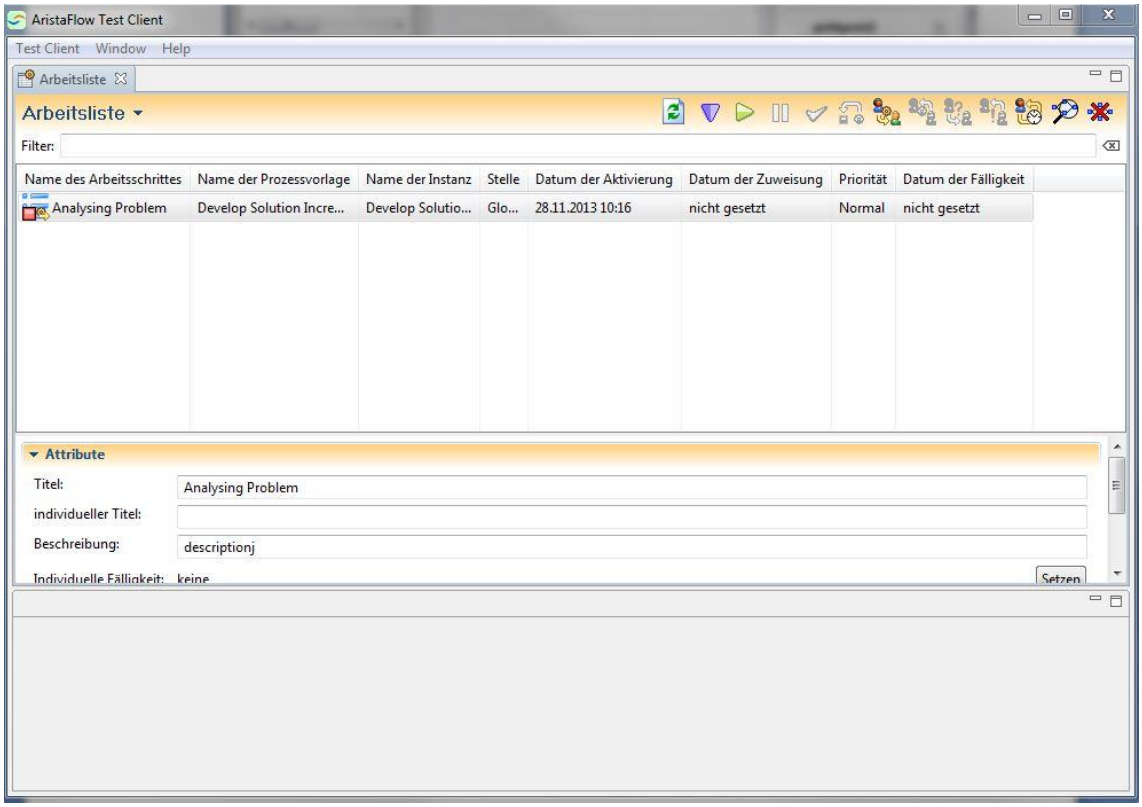

Abbildung 15: TestClient

Client

Im Client bekommt jeder Mitarbeiter seine ausstehenden Prozessschritte in einer Arbeitsliste angezeigt. Diese können auch sofort gestartet und bearbeitet werden. Darüber hinaus bietet der Client die Möglichkeit, sich Prozessinstanzen graphisch anzeigen zu lassen.

Automatic Client

Der Automatic Client ist aus Sicht des Servers nichts anderes als ein normaler Client. Der Unterschied besteht darin, dass diese Aktivitäten vom Automatic Client automatisch, selbstständig und ohne weitere Benutzerinteraktion ausgeführt werden. Dies kommt zur Anwendung, wenn keine menschliche Interaktion von Nöten ist, wie beispielsweise das Eingeben von Daten. Es wird also in einer Aktivität ein Programm gestartet, dieses läuft und erfüllt seinen Zweck und anschließend ist die Aktivität beendet. Solche automatischen Aktivitäten laufen meist über den Automatic Client.

Monitor

Der Monitor ermöglicht das Anzeigen sämtlicher laufenden Prozessinstanzen sowie deren Logs, vorausgesetzt man besitzt die nötigen Zugriffsrechte. Die Logs beinhalten sämtliche Informationen über Änderungen, Abläufe, geschriebene Daten etc. welche vom Moment des Startens der Instanz an gesammelt werden. Darüber hinaus lassen sich hier fehlgeschlagene Aktivitäten wieder zurücksetzen und neu starten. Schließlich lassen sich hier auch alle möglichen Ad hoc Änderungen an den Instanzen vornehmen. Diese Änderungen unterstehen natürlich ebenfalls dem Correctness by Construction Prinzip sowie sämtlichen anderen Validitätsprüfungen (wie schon beim Template Editor). Dadurch wird sichergestellt, dass die Änderungen keine fehlerhaften Prozesse ergeben und die weitere Ausführung nicht gefährdet ist.

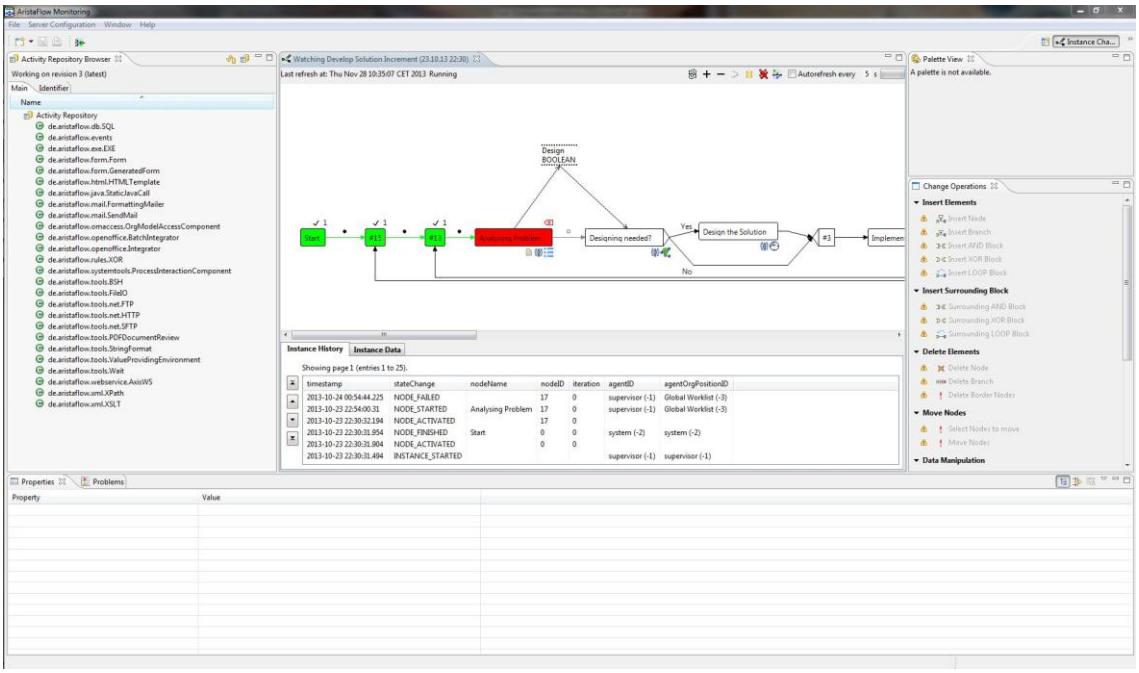

Abbildung 16: Monitor

Server

Der Aristaflow Server stellt den grundlegenden Teil der ganzen Aristaflow Architektur dar. Auf diesem laufen sämtliche Prozessinstanzen, hier sind auch Templates, Aktivitäten, Orgpositionen etc. hinterlegt. Der Server setzt intern auf eine relationale Datenbank auf, welche austauschbar ist, solange eine JDBC-Schnittstelle vorhanden ist. Bei der Architektur wurde großen Wert auf eine hohe Performanz sowie Wartbarkeit gelegt.

3. Anforderungen

Wie eingangs bereits erwähnt sind die Anforderungen in der Softwareentwicklung sehr hoch. Kurze Entwicklungszeiten und hohe gewünschte Produktqualität bringen einige Probleme mit sich. Diese Anforderungen spiegeln sich natürlich auch in allen (Unter-)Bereichen wieder, in diesem Fall speziell die Anpassung von Prozessen. Es gibt eine Vielzahl von gewünschten und benötigten Anforderungen um ein gutes und praktikables System zu erstellen. Im folgenden werden nun die wichtigsten Punkte aufgezählt und erläutert.

3.1 Anforderungen an ein Softwaresystem

Allem voran ist der wichtigste Punkt natürlich die Korrektheit des Systems. Werden nicht die richtigen Arbeitsschritte fehlerfrei ausgeführt, so erhält man auch kein korrektes Ergebnis und somit ist es sinnlos mit dem System zu arbeiten. Eng damit verknüpft sind die Sicherheit und (Daten-)Konsistenz des Systems. Zu jedem Zeitpunkt muss sichergestellt werden können, dass alle Datensätze korrekt sind und alle Arbeitsschritte ordnungsgemäß ausgeführt werden. Dies gilt nicht nur für den normalen Betrieb, sondern (und vor allem) in Ausnahmesituationen wie beispielsweise einem Stromausfall oder ähnlichem.
Ein weiterer wichtiger Punkt ist die Effizienz eines Prozesses. Je ineffizienter ein Prozess in seiner Ausführung ist, desto teurer wird das gesamte Projekt und desto geringer ist die Produktqualität am Ende. Kosten sind ein wichtiger Faktor. Einerseits geht es hierbei um die reinen Anschaffungskosten, welche natürlich so gering wie möglich sein sollten, andererseits aber natürlich auch um die weiteren Betriebskosten (Personalkosten, Lizenzgebühren etc.).
Als weiterer wichtiger Punkt bei Software ist die Userfreundlichkeit zu nennen. Dazu gehören wichtige Dinge wie eine gute Übersichtlichkeit (Interface), Kompatibilität mit anderen Programmen und eine gute und einfache Wartbarkeit sowie Erweiterbarkeit (Updates, Add-Ons etc.), aber auch Dinge wie zum Beispiel eine gewisse Offenheit und Konfigurierbarkeit. Dies ist vor allem

wichtig für Kunden die sich selbst gut auskennen und das Programm so besser an ihre eigenen Voraussetzungen anpassen können. Hinzu kommt noch dass es sich meist um Mehrbenutzer Arbeit handelt. Der Zugriff sollte für jeden einzelnen davon einfach möglich sein und es müssen die richtigen Zugriffsrechte geregelt sein. Letztlich könnte man noch einen möglichst guten Support hinzunehmen, denn das System soll ja auch von Firmen genutzt werden können welche nicht selbst die nötigen Kenntnisse über die Materie haben.

Alle diese Anforderungen (und noch mehr) sollten möglichst gut umgesetzt werden. Dabei können jedoch niemals alle gleichzeitig "maximiert" werden, da manche sich in gewisser Weise entgegen stehen. So leidet beispielsweise die Effizienz eines Programms unter einer erhöhten Sicherheit, da die nötigen Sicherheitsroutinen etc. natürlich Ressourcen benötigen welche dann an anderer Stelle fehlen. Es gilt also die richtige Balance zu finden. Wie diese genau aussieht hängt vom jeweiligen Projekt ab, so wird man bei sicherheitskritischen Projekten mehr Wert auf die Sicherheit legen als bei weniger kritischen, welche eher auf Kostenersparnis und Effizienz setzen.

3.2 Anforderungen an Prozessanpassungen

Um eine möglichst gute (Ad hoc) Prozessanpassung ermöglichen zu können müssen dahingehend einige Anforderungen erfüllt werden. An erster Stelle sei hier die Korrektheit des Prozesses genannt. Dies ist der wichtigste Punkt der beachtet und bei Änderungen eines Prozesses sichergestellt werden muss. Einerseits muss der neue Prozess weiterhin den realen Prozess korrekt wiederspiegeln, andererseits muss er in sich geschlossen korrekt sein. Das bedeutet es dürfen keine Änderungen vorgenommen werden welche die (weitere) Ausführung des Prozesses gefährden oder unmöglich machen könnten. Soll in einem Softwareentwicklungsprozess beispielsweise eine Qualitätssicherungsmaßnahme wie etwa Tests eingefügt werden, so muss darauf geachtet werden, dass die Einfügung dort stattfindet wo alle erforderlichen Daten und Ressourcen (Quellcode, Fehlermeldungen etc.) vorhanden sind. Sollte dies nicht der Fall sein so könnte dies entweder zu

falschen Ergebnissen, oder im schlimmsten Fall sogar zu einem Deadlock und somit dem Abbruch des Prozesses führen. Auch Lifelocks wären möglich sollte man beispielsweise eine Schleife in den Prozess einbauen welche aufgrund fehlerhaft gesetzter Variablen immer wieder ausgeführt wird.

Ein weiterer sehr wichtiger Punkt ist die Effizienz, also die Frage wie effizient sich die benötigte Anpassung durchführen lässt. Diese Frage ist vor allem bei großen Projekten (und somit meist auch komplexen Prozessen) relevant. Hat man also ein großes Projekt mit vielen Prozessinstanzen und vielen beteiligten Mitarbeitern, so muss sichergestellt werden, dass die Einfügung von Qualitätssicherungsmaßnamen (vom Beispiel oben) für den richtigen Bearbeiter in der richtigen Instanz geschieht. Dabei sollte der Aufwand des Findens der richtigen Instanz und des Einfügens der Aktivitäten so gering wie möglich gehalten werden. Dies dient einerseits der Fehlervermeidung, andererseits sollte versucht werden eine möglichst hohe Zeitersparnis (für die Anpassung an sich) zu erreichen.

Zwei wichtige Aspekte in der Softwareentwicklung, wie auch hier bei Prozessanpassungen, welche erstrebenswert sind, sind eine mögliche Wiederverwendbarkeit sowie eine gute Wartbarkeit. Wiederverwendbarkeit bedeutet in diesem Zusammenhang, dass vorgenommene Anpassungen zu einem späteren Zeitpunkt in der gleichen oder einer ähnlichen Situation leicht zu Wiederholen sind bzw., dass einmal vorgenommene Anpassungen auch auf andere Situationen angewendet werden können. So lässt sich viel doppelte Arbeit sparen und der ganze Anpassungsprozess beschleunigen und effizienter gestalten. Beispielsweise könnte und sollte das Einfügen von Tests bei der Softwareentwicklung so gestaltet und realisiert werden, dass dieselben Operationen auch für andere Artefakte (also zum Beispiel bestimmte Klassen oder Methoden) so ausgeführt werden können und zu einem gültigen Ergebnis führen. Somit muss man sich nicht jedes Mal damit befassen sondern kann bereits erarbeitete Adaptionen wiederverwenden. Hier kommt nun auch die gewünschte gute Wartbarkeit ins Spiel. Optimaler weise hat man nun also ein Portfolio von Adaptionen zur Hand welche man in verschiedenen Situationen verwenden kann. Diese Lösungen sollten allerdings wiederrum so gestaltet sein, dass sich diese möglichst problemlos abändern lassen. Beispielsweise könnte man beim Einfügen von Qualitätssicherungsmaßnahmen einen weiteren

Schritt hinzufügen, welcher dann aber keine Probleme verursachen kann, sodass diese Art der Adaption weiterhin für möglichst viele Anwendungsfälle passt und funktioniert.

4. Konzept

Prozessanpassungen können auf verschiedenste Art und Weise vorgenommen werden. Dabei existieren bei jedem Ansatz verschiedene Vor- und Nachteile. Hier werden nun verschiedene Ansätze erläutert, evaluiert und einander gegenübergestellt um am Ende den bestmöglichen Ansatz zu finden.

Die grundsätzliche Situation ist, dass ein bestehender Prozess zur Laufzeit, das heißt während er gerade ausgeführt wird, verändert werden soll. Die vorgenommenen Adaptionen können aus vielen Gründen von Nöten sein. Alle dieser Möglichkeiten hier zu behandeln würde den Umfang dieser Arbeit sprengen, daher wird hier ein wichtiges Beispiel heraus gegriffen.

Hierfür wird das Einfügen von Qualitätssicherungsmaßnahmen als Prozessadaption gewählt. Diese Maßnahmen sollen dabei helfen Fehler (frühzeitig) zu finden, diese zu beheben und allgemein die Produktqualität zu steigern. Eine solche Einfügung besteht hierbei aus 4 Aktivitäten, Informationen beschaffen, Code analysieren, Maßnahme anwenden und schließlich den neuen Build generieren. Was diese einzelnen Aktivitäten genau tun wird im nächsten Kapitel genauer erläutert.

Andere Gründe für eine Adaption des Prozesses wären zum Beispiel die Koordination von Aktivitäten, also das Ändern der Reihenfolge von Arbeitsschritten, oder aber auch Exception Handling, also das Reagieren auf Unstimmigkeiten oder Fehler im Prozessablauf, ebenso wie andere Situationen welche hier nicht näher erläutert werden.

4.1 Evaluation verschiedener Ansätze

Die Anforderungen an einen guten Softwareentwicklungs- und Adaptionsprozess sollen nun möglichst effizient realisiert werden. Dazu müssen einige Dinge beachtet werden.

Der grundlegende Ablauf einer Anpassung zur Qualitätssicherung könnte wie folgt aussehen: Zuerst müssen verschiedenste Informationen beschafft werden, beispielsweise die (veränderten) Anforderungen des Kunden, bereits durchgeführte Schritte, der Stand des Projektes, Mitarbeiterkapazitäten, bekannte Probleme etc..

Diese Informationen müssen nun verarbeitet werden, da sie später eventuell die Art und Weise der durchzuführenden Adaption beeinflussen können. Sollten zum Beispiel Mitarbeiter andere Aufgaben bearbeiten, müssten diese von ihren Posten abgezogen und zu den neuen Aufgaben delegiert werden, oder aber muss der Zeitpunkt einer zusätzlichen Aktivität anders gewählt werden als wenn genügend freie Kapazitäten vorhanden sind.

Nachdem diese grundlegenden Informationen vorhanden sind muss der Programmcode sowie andere Produkte von Interesse (Dokumentation, Daten, etc.) analysiert werden. Diese Analyse dient dazu die Qualität der betrachteten Produkte einzuschätzen und zu evaluieren. Werden dabei beispielsweise Fehler, Bugs etc. gefunden, so weiß man anschließend welche Qualitätsmaßnahmen von Nöten sind, also an welchen Punkten man ansetzen muss und was genau an welcher Stelle noch zu tun ist. Hierbei werden auch die Anforderungen gefunden welche noch nicht implementiert, oder vergessen wurden. Dies ist sogar sehr wichtig, da der Kunde ein Produkt erwartet welches vollständig seinen Anforderungen entspricht, und nicht, dass bestimmte gewünschte Features fehlen.

Mithilfe der nun beschafften Informationen, und allem weiteren was die Analyse erbracht hat, können die Maßnahmen bestimmt werden welche durchzuführen sind. Hierbei handelt es sich um mehrere Punkte: WELCHE Maßnahme muss durchgeführt werden, WANN und von WEM ist sie zu erledigen und WIE sieht das (optimale) Ergebnis nach der Durchführung aus.

Diese Maßnahmen müssen dann ausgeführt werden. Dabei kann es sich um automatisierte Vorgänge handeln, wie beispielsweise das weiterreichen von Informationen, oder um andere Tätigkeiten, welche von Mitarbeitern durchgeführt werden müssen (Programmieren, Bugfixes, Teammeetings und weitere Planung etc.).

Nachdem die ersten drei Schritte durchgeführt wurden kann alles wieder sauber in das Projekt integriert werden, es existiert nun sozusagen eine neue Version, und die Arbeit kann weitergeführt werden (sofern das Projekt nicht beendet wurde), bis eventuell neue Adaptionen von Nöten werden.

Diese vier Schritte stellen nun die Qualitätssicherungsmaßnahme dar, es muss nur entschieden werden an welchen Stellen des Workflows diese eingefügt werden sollen. Dazu müssen die laufenden Instanzen des Betreffenden Workflows betrachtet werden, und je nachdem in welchem Zustand sich diese befinden, können die Einfügungen und eventuelle Verschiebungen etc. durchgeführt werden.

Um die oben genannten Schritte in die Tat umzusetzen lassen sich verschiedene Ansätze vorstellen. Im folgenden werden einige davon evaluiert und an einem kleinen Beispiel Szenario durchgespielt. Jeder dieser Ansätze wurde gewählt da er naheliegend ist und auch praktisch umgesetzt werden kann. Natürlich gibt es mehr denkbare Ansätze als die hier genannten 4, aber hier soll sich auf diese beschränkt werden.

Hierbei wird mit dem Develop Solution Increment des OpenUP Prozessmodells gearbeitet. Wie der Name schon sagt wird dabei eine (Teil-)Lösung des Problems entwickelt welche dann inkrementell weiter bearbeitet wird. Es handelt sich also um eine iterative Vorgehensweise zur Softwareentwicklung. In jeder Iteration können vorher gefundene Fehler, weitere Programmteile und auch geänderte Anforderungen (zum Beispiel des Kunden) eingepflegt, behoben und entwickelt werden. So entsteht sozusagen eine ToDo-Liste von welcher man jeweils die nächsten Schritte für die nächste Iteration heraus nimmt und so sukzessive das fertige Produkt entwickelt wird.

Grafisch lässt sich dieser Prozess folgendermaßen Darstellen:

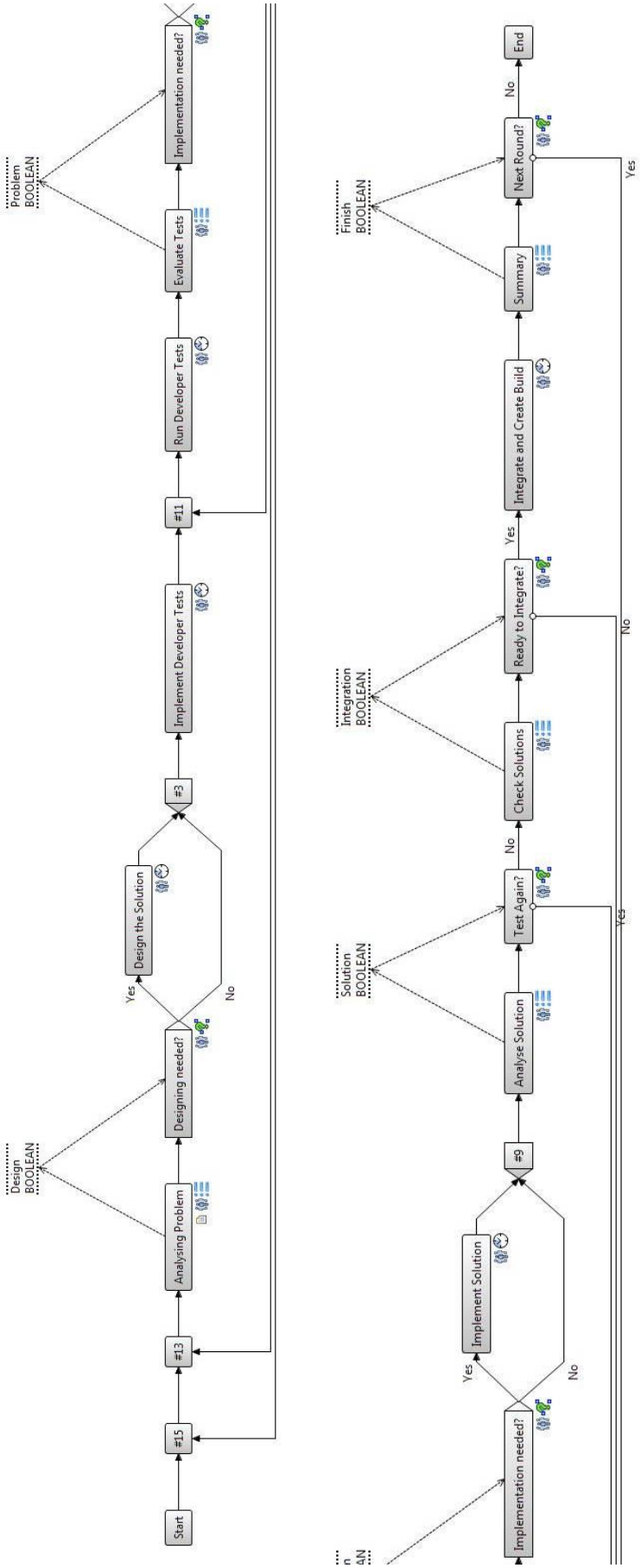

Abbildung 17: Development Solution Increment Workflow

Betrachten wir zunächst einen **naiven Ansatz**. Hierbei existiert neben dem eigentlichen Prozessworkflow ein oder mehrere weitere Prozesse für die Qualitätsmaßnahmen. Zu Beginn des Projektes gibt es also zwei Prozessworkflows, einen für die Entwicklung der Software und einen zur Qualitätssicherung (Testing, Debugging etc.). Im Normalfall würde dieser zweite Prozess nach dem eigentlichen Entwicklungsprozess angestoßen und durchgeführt werden. Eine parallele Ausführung der beiden Prozesse ist dahingehend problematisch, dass die Abstimmung der beiden recht schwierig wäre, und darüber hinaus bräuchte man mehr Mitarbeiterkapazitäten.

Immer wenn der Kunde Änderungswünsche und neue Anforderungen ins Spiel bringt, müsste ein neuer Workflow generiert werden, welcher sich allein damit befasst, da solche Qualitäts- und Änderungsmaßnahmen ja nicht zu Beginn des Projektes bekannt sind und somit nicht geplant und modelliert werden können. Daraus ergeben sich weitere Probleme der Koordination und der Planung des Ablaufs des Projektes (Mitarbeiterkapazitäten, Zeitmanagement etc.). Darüber hinaus hat eine solche "Zerstückelung" des Gesamtprojekts einen sehr hohen Koordinations- und Kommunikationsaufwand zur Folge. Die einzelnen Schritte müssen geplant und die Fortschritte jedes einziges Segments an die Kollegen kommuniziert werden. Klappt das alles nicht reibungslos, so zieht dies verschiedene Konsequenzen nach sich. Beispielsweise ist es so fast unmöglich das Gesamtprojekt in sich konsistent zu halten. Ebenso muss gewährleistet werden, dass die einzelnen Komponenten miteinander Kompatibel sind (Schnittstellen, korrekter Datenfluss, etc.). Bei Anforderungsänderungen des Kunden, welche sich auf bestehende Teile des Systems beziehen kommt ein weiteres Problem hinzu. Sollte der/die Mitarbeiter, welche diesen Teil realisiert haben, mit etwas anderem beschäftigt sein, so würden andere Mitarbeiter die Adaption vornehmen müssen. Dies hat zur Folge, dass diese sich erst einmal in den Code des anderen einarbeiten müssten um damit arbeiten zu können. Bei eventuell schlechter Dokumentation etc. kann dies viel Zeit und Arbeit in Anspruch nehmen.

Die genannten Anforderungen an die Softwareentwicklung (kostengünstig, effizient, flexibel...) werden bei diesem Ansatz nicht annähernd erfüllt. Somit ist dieser Ansatz absolut nicht praktikabel.

Separate Workflows für Qualitätssicherung und Anpassungen genügen also nicht den Anforderungen. Daher ist der nächste Schritt die Überlegung den Entwicklungsworkflow zu adaptieren um Änderungen wie Qualitätsmaßnahmen einfügen zu können. Eine solche **manuelle Adaption** könnte zum Beispiel wie folgt aussehen: Von Zeit zu Zeit führt ein dafür zuständiger Mitarbeiter eine Bestandsaufnahme durch. Sollte er zu dem Schluss kommen, dass bestimmte Qualitätsmaßnahmen nötig sind, so plant er diese, bestimmt von wem und wann sie durchzuführen sind und fügt sie letztendlich in den Workflow ein. Die Maßnahmen werden nun durchgeführt wenn sie an der Reihe sind, was im Gegensatz zum naiven Vorgehen sehr viel weniger Probleme verursacht. Beispielsweise werden so Inkonsistenzen eher vermieden, und der Ablauf des Projektes bleibt sehr viel übersichtlicher. Es ist nun deutlich besser erkennbar an welchem Punkt sich das Projekt befindet (weniger parallele Vorgänge / Workflows). Auch die Bearbeiterzuordnung lässt sich auf diese Weise sehr viel besser umsetzen.

Auf Änderungswünsche des Kunden kann hierbei genauso eingegangen werden. Der Mitarbeiter plant dazu den Zeitpunkt und den Bearbeiter und fügt die entsprechende Aktivität an einer geeigneten Stelle in den Workflow ein.

In unserem Beispiel müsste man also nach jeder Iteration das Zwischenergebnis prüfen und analysieren, um dann entsprechende Anpassungen stattfinden zu lassen. Ebenso müsste der Bearbeiter bei jedem Änderungswunsch des Kunden die entsprechenden Aktivitäten planen und in den Workflow einpflegen.

Einige Probleme werden also durch dieses Vorgehen behoben, allerdings bleiben wichtige Punkte offen. Der Mitarbeiter muss einen Überblick über sämtliche Workflows und deren Instanzen haben. Es ist ein beträchtlicher Aufwand nötig um den Status sämtlicher Aktivitäten im Blick zu haben und dementsprechend zu planen wann und wo Aktivitäten optimal eingefügt werden sollten. Ebenso stellt sich die Frage wann es nötig ist die Analysen durchzuführen und wie genau die Ergebnisse zu interpretieren und zu bewerten sind. Der Aufwand dieser Aufgaben wächst mit der Größe des Projektes und der Anzahl der involvierten Mitarbeiter, sowie der Menge der Änderungen etc.. Je größer der Aufwand wird, desto mehr Zeit wird auch benötigt um entsprechende Schritte manuell durchzuführen (Fehlende Effizienz). Es ist klar,

dass eine solche manuelle Adaption schnell zu aufwändig werden wird und somit nicht mehr die optimalen Anpassungen geplant und durchgeführt werden können. Dafür sind oft zu viele Faktoren im Spiel welche berücksichtigt werden müssen.

Die manuelle Adaption von (großen) Prozessen scheitert also an einem zu hohen Arbeitsaufwand und fehlender Übersicht über alle relevanten Informationen.
Um diese Probleme zu beheben müsste man also eine **automatische Prozessadaption** realisieren. Die könnte folgendermaßen aussehen. Es existiert ein Programm für die Analyse und Adaption von Prozessen oder deren Instanzen. Diese beschaffen sich nun die Informationen welche benötigt werden um Anpassungen vornehmen zu können (Code Analysetools etc.). Auf Grundlage dieser Informationen werden nun automatisch nötige Qualitätsmaßnahmen in den Workflow oder die Instanz eingefügt sowie alles weitere geplant (Mitarbeiterzuordnung usw.). Durch diese Automatisierung lässt sich sehr viel effizienter arbeiten und das Ergebnis wird optimaler sein als bei einer manuellen Anpassung, da ein Programm sehr viel mehr Informationen verarbeiten kann als ein Mensch und somit auch Flüchtigkeitsfehler vermieden werden.
Probleme gibt es jedoch auch hier noch. Während das Programm läuft sollten möglichst keine Aktivitäten ausgeführt werden um eine eindeutige und optimale Anpassung planen zu können. Das bedeutet jedoch auch, dass (wenn auch nur für kurze Zeit) Leerlauf in dieser Prozessinstanz entsteht. Ebenso kann es zu Problemen kommen wenn sich für eine nötige Anpassung keine geeignete Zielinstanz finden lässt, weil beispielsweise keine sich in einem passenden Zustand befindet. Ebenso ist es im Code schwierig darauf einzugehen in welchem Status sich die zu ändernde Instanz gerade befindet. Auch Änderungswünsche des Kunden (wie auch in unserem Beispiel möglich) müssten entweder weiterhin per Hand eingefügt werden, oder aber sie würden erst realisiert werden wenn das Programm wieder ausgeführt wird. Dazu müsste man dieses natürlich so implementieren, dass es möglich ist eine gewünschte Anpassung mit auf die zu realisierende Liste von Adaptionen zu setzen. Als weitere Schwachstelle könnte man hier, und dies gilt auch für die

vorhergehenden Konzepte, die schlechte Wiederverwertbarkeit anführen. Ein solches Programm müsste recht statisch auf einen bestimmten Prozess zugeschnitten werden um optimale Anpassungen zu ermöglichen.

Automatische Adaption bietet also schon mehr als die vorherigen Ansätze, jedoch sind immer noch einige Punkte vorhanden, welche man verbessern möchte.
Um nun auch diesen Problemen Herr zu werden bietet sich die Möglichkeit die Adaptionen in einen eigenen Workflow auszulagern. Dieser **Adaptionsworkflow** läuft parallel und unabhängig zum eigentlichen Produktionsworkflow und führt automatisch, wie oben beschrieben Anpassungen an diesem durch. Dieser Adaptionsworkflow kann nun speziell auf den Zielworkflow angepasst und optimiert werden. Hierbei können Änderungen jederzeit dynamisch während der Laufzeit des Prozesses an geeigneten Stellen eines passenden Workflows eingefügt werden. Der Adaptionsworkflow wählt also aus einem Portfolio von passenden Adaptionen diejenigen aus, welche zu den gesammelten Informationen, Anforderungen und dem Status des Prozesses passen und fügt diese in eine entsprechende, passende Instanz ein. So können auch Änderungswünsche des Kunden als Input in den Adaptionsworkflow gegeben werden, welcher dann automatisch alles Nötige in die Wege leitet.
Durch das Auslagern der Adaptionen können Wartungsarbeiten sowie Änderungen der Adaptionsaktivitäten leicht vorgenommen werden, ohne den eigentlichen Prozess zu unterbrechen oder zu gefährden, da solche Änderungen keinen direkten Einfluss auf diesen haben. Ebenso lässt sich ein solcher Adaptionsworkflow leicht für andere Projekte wiederverwerten. Dazu müssen lediglich ein paar wenige Änderungen und Anpassungen vorgenommen werden um den neuen Prozess optimal unterstützen zu können.
All diese genannten Punkte sorgen für eine optimale dynamische Prozessunterstützung und führen zu mehr Effizienz und besserer Produktqualität. Die gestellten Anforderungen der Softwareentwicklung werden bestmöglich beachtet und in die Tat umgesetzt.
Wie ein solcher Adaptionsprozess aussehen könnte, und wie dieser genau arbeiten soll wird im folgenden Kapitel erläutert.

4.2 Anpassung mithilfe eines Adaptionsworkflows

Wie oben bereits beschrieben handelt es sich bei einem Adaptionsprozess um einen völlig separaten, vom eigentlichen Produktionsprozess unabhängigem Prozess. Dieser könnte von Mitarbeitern gestartet werden oder aber auch regelmäßig (oder ständig) nebenbei laufen und sich um die nötigen Adaptionen kümmern.

Um dieses Konzept veranschaulichen zu können, wird wieder das Beispiel des OpenUP Workflows verwendet. Natürlich lässt sich für jeden anderen Prozess ebenso ein Adaptionsworkflow erstellen. Diese funktionieren grundlegend auf dieselbe Art und Weise, es müssten lediglich kleinere Anpassungen durchgeführt werden. Einen allgemein gültigen Adaptionsworkflow für alle möglichen Prozesse könnte man zwar theoretisch auch erstellen, dieser müsste dann allerdings sehr allgemein gehalten werden und wäre nicht so effizient wie unser speziell angepasster. Darüber hinaus dient eine gewisse Vereinfachung hier zur besseren Übersicht und Verständnis. Grafisch sieht der fertige Prozess folgendermaßen aus:

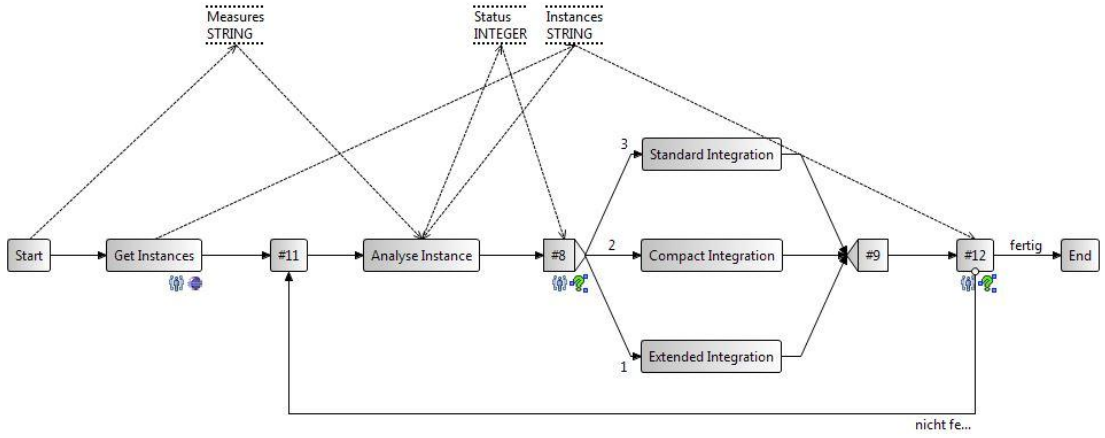

Abbildung 18: Adaptionsworkflow

Die grundlegende Idee ist es, dass der Prozess am Anfang übergeben bekommt, welche Qualitätsmaßnahmen erledigt werden müssen, und auf welche Artefakte sich diese beziehen. Artefakte sind in diesem Fall beispielsweise Klassen, Methoden oder ähnliches. Ziel ist es nun diese Qualitätsmaßnahmen in die jeweiligen Instanzen der betreffenden Workflows

einzupflegen. Dazu muss zuerst eine Liste der laufenden Instanzen generiert werden. Dies geschieht in der Aktivität "Get Instances". Nun beginnt eine Schleife, welche durch sämtlichen Instanzen läuft und die folgenden Schritte darauf ausführt.

Zunächst muss die Instanz analysiert werden, dies geschieht mithilfe der Aktivität "Analyse Instance". Relevant bei dieser Analyse sind hierbei zwei Dinge: Erstens ob die Instanz eines der Artefakte bearbeitet für welches eine Anpassung notwendig ist, also ob diese Instanz mit den relevanten und betrachteten Klassen oder Methoden usw. zusammenhängt oder nicht. Zweitens muss man den Status der laufenden Instanz in Erfahrung bringen. Vom Status der Instanz hängt das weitere Vorgehen maßgebend ab. Mit Status der Instanz ist hierbei gemeint, welche der einzelnen Aktivitäten der Instanz wurden bereits ausgeführt und erfolgreich beendet, welche Aktivitäten laufen gerade, wurden übersprungen oder stehen noch an. Die Summe dieser Zustände ergeben somit den Status der gesamten Instanz und bilden die Grundlage für die anstehende Entscheidung wie genau der Adaptionsworkflow weiter agieren soll. Je größer und komplexer ein Prozess ist, desto mehr mögliche Status gibt es. Für das bestmögliche Ergebnis mit dem höchsten Grad an Effizienz müsste man jeden möglichen Status berücksichtigen und separat behandeln. Da dies sehr komplex und unübersichtlich werden kann, wird für diese Arbeit und das verwendete Beispiel eine Vereinfachung auf drei Möglichkeiten vorgenommen. Dadurch lässt sich die Funktionsweise übersichtlicher und leichter darstellen. Nach der Analyse findet nun die Fallunterscheidung statt. Läuft bereits die letzte Aktivität der Instanz, so wird der obere Pfad gewählt und die Aktivität "Extended Integration" kommt zur Ausführung (siehe Abbildung 19).

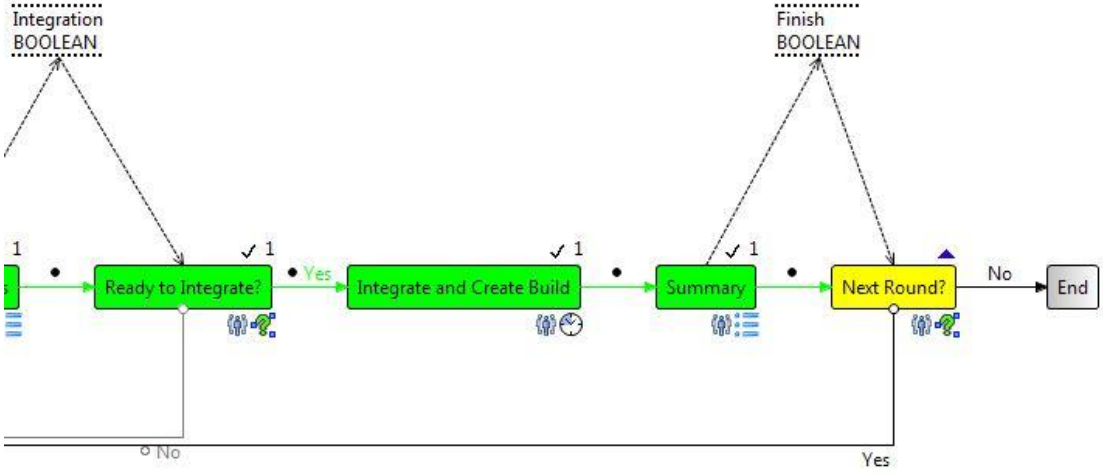

Abbildung 19: Fall: letzte Aktivität aktiv (Ausschnitt)

Dabei werden die vier Qualitätsmaßnahmen alle nacheinander direkt nach der eigentlich letzten Aktivität eingefügt (siehe Abbildung 20).

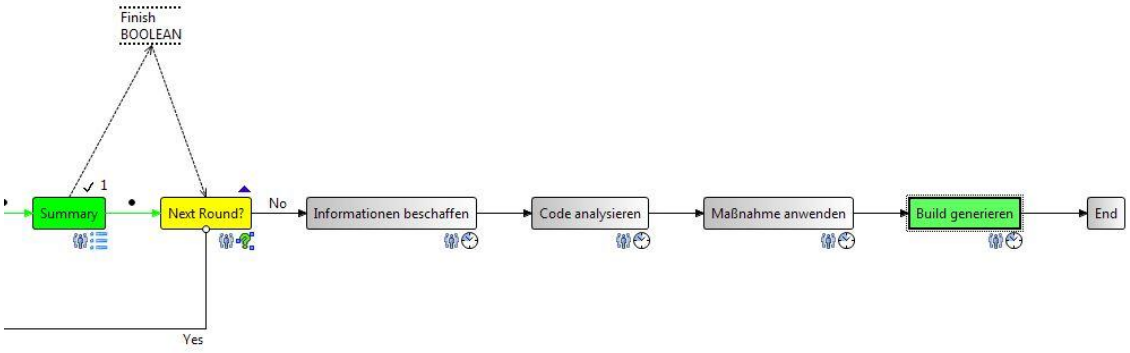

Abbildung 20: Nach Ausführung von Extended Integration (Ausschnitt)

Also der Reihe nach: Informationen beschaffen, Code analysieren, Maßnahme anwenden und schließlich die Integration und das generieren des neuen Builds. Diese Vorgehensweise macht Sinn, da die letzte Aktivität bereits läuft und somit alle vorherigen bereits abgeschlossen sind. Ein Einfügen an einer anderen Stelle wäre sinnlos, da sonst die eingefügten Aktivitäten niemals zur Ausführung kommen würden.

Läuft die letzte Aktivität der Instanz noch nicht, sondern eine der vorhergehenden so fassen wir dies vereinfachend als einen anderen Status zusammen (siehe Abbildung 21).

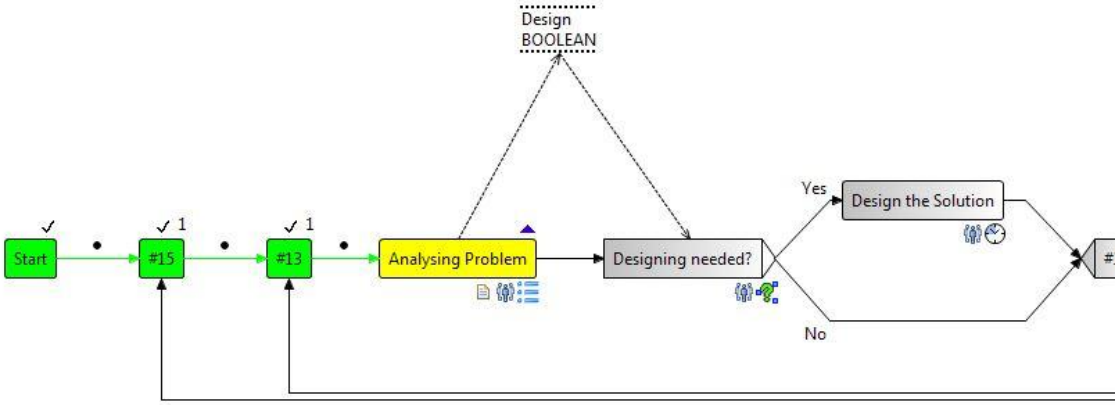

Abbildung 21: Fall: Eine andere Aktivität ist aktiv (Ausschnitt)

Nun muss noch unterschieden werden ob die Instanz ein Artefakt bearbeitet welches Anpassung benötigt oder nicht. Sollte dies nicht der Fall sein, so wird der obere Pfad zur "Standard Integration" gewählt und eben diese Aktivität kommt zur Ausführung. Dabei werden dann die vier Maßnahmen zur Qualitätssicherung am Anfang des Prozesses an passenden, sinnvollen Stellen eingefügt (siehe Abbildung 22).

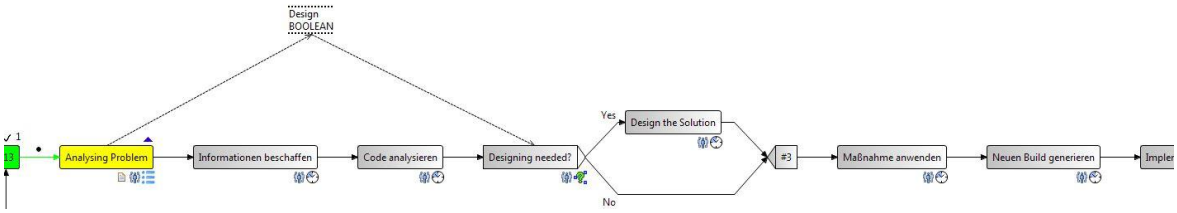

Abbildung 22: Nach Anwendung von Standard Integration (Ausschnitt)

Sinnvoll bedeutet in diesem Fall erstens, dass sie natürlich in der richtigen Reihenfolge bleiben, andererseits sollte man Stellen wählen welche möglicherweise mehr geeignet sind als andere, beispielsweise weil nach einer Aktivität mehr relevante Daten vorliegen als davor oder ähnliches. Allgemein lässt sich hier natürlich nicht sagen an welchen Stellen die Einfügungen stattfinden sollten, dies muss vom Programmierer bei der Erstellung des

jeweiligen Adaptionsworkflows für jedes Projekt entschieden werden. Hierbei zeigt sich schön einer der großen Vorteile eines separaten Adaptionsworkflows. Sollte sich am eigentlichen Prozess etwas ändern, oder sollte entschieden werden das die Einfügungen an anderen Stellen stattfinden sollen oder auch andere Aktivitäten mit hinzu kommen, so kann dies leicht und unabhängig im "abgekapselten" Workflow umgesetzt werden ohne den eigentlichen (und vielleicht laufenden) Produktionsworkflow anfassen zu müssen. In anderen Worten lässt sich ein solches System viel einfacher warten und aktualisieren als andere. Dies ist auch der Punkt den man angehen müsste um einen existierenden Adaptionsworkflow auf andere Workflows zu portieren. Die Grundlegenden Voraussetzungen sind und bleiben meistens dieselben, nur Kleinigkeiten bei der Anpassung (Reihenfolge, Stellen etc.) müssten ausgetauscht und angepasst werden. In anderen Worten zeigt sich hier das Potential zur Wiederverwertung eines solchen Prozesses.

Befindet sich die Instanz im selben Status wie eben, bearbeitet jedoch eines der relevanten Artefakte, so findet die "Compact Integration" ihre Anwendung. Hierbei werden wie schon zuvor Einfügungen weit am Anfang des Prozesses vorgenommen, wiederum unter denselben Voraussetzungen wie oben beschrieben, jedoch kann hier auf den Schritt der Codeanalyse verzichtet werden. Dieser Schritt ist hierbei überflüssig, da man ja bereits weiß dass dieses Artefakt bestimmte, ebenfalls bereits bekannte Maßnahmen, benötigt.

Nach Auswahl und Ausführung der richtigen Vorgehensweisen kann nun einerseits dieses Artefakt aus der Liste der nötigen Anpassungen entfernt (sofern eine Übereinstimmung stattgefunden hat) und mit der nächsten Instanz fortgefahren werden. Diese Schleife läuft nun so lange, bis die Liste der Instanzen leer, und somit alles erledigt ist. Der Adaptionsworkflow kann nun erfolgreich beendet werden. Im hier verwendeten Beispiel wird das Insert Pattern verwendet um die Qualitätsmaßnahmen umzusetzen. In diesem Fall macht dies auch Sinn, da diese Maßnahmen vorher nicht modelliert waren und nun zusätzlich hinzu kommen. In anderen Anwendungsfällen wäre es dementsprechend sinnvoll auch mit anderen Pattern zu arbeiten, beispielsweise könnten im Prozess selbst schon Maßnahmen zur Qualitätssicherung vorgesehen sein, welche dann wenn nötig mithilfe des Move Patterns an dafür geeignete Stellen verschoben werden. Ebenso wäre es in diesem Beispiel

möglich, dass bei fehlerfreien und guten Ergebnissen Maßnahmen aus dem Prozess entfernt werden (Delete Pattern) wenn diese unnötig werden oder bereits zu einem früheren Zeitpunkt anderweitig ausgeführt wurden.

Auf diese Art und Weise kann jeder Workflow, für den eine dynamische automatisierte Prozessanpassung gewünscht ist, ein passender Adaptionsworkflow erstellt und betrieben werden. Dabei macht es eigentlich keinen Unterschied um was für einen Produktionsprozess es sich im Detail handelt. Solche Adaptionen sind nicht nur für Softwareentwicklungsprozesse gültig, sondern könnten auch in anderen Branchen und Anwendungsfällen benutzt werden. dafür müsste man lediglich die einzufügenden Aktivitäten ausgetauscht, und an das jeweilige Projekt angepasst werden. Dabei kann es sich um alles Mögliche handeln, beispielsweise im Sektor der Krankenhäuser, genauer der Patientenbetreuung (mögliche Aktivitäten zum Beispiel zusätzliche Untersuchungen wie Röntgen etc.) oder der Produktion von industriellen Gütern (mögliche Aktivitäten zum Beispiel Materialtests oder Langzeittests etc.).
Darüber hinaus ist das Arbeiten mit Adaptionsworkflows offensichtlich unabhängig von der Projekt- und Teamgröße praktikabel. Es macht im Endeffekt keinen Unterschied ob der Adaptionsworkflow nur wenige Instanzen verarbeitet (kleine Projekte mit weniger Mitarbeitern und parallelen Arbeitsschritten an verschiedenen Artefakten) oder hunderte (große Projekte, viele Mitarbeiter die parallel an verschiedenen Artefakten arbeiten).

Der hier beschriebene beispielhafte Adaptionsprozess ist absichtlich sehr simpel gehalten und nimmt einige Vereinfachungen vor, zum Beispiel gibt es nur eine Fallunterscheidung von drei verschiedenen Status, natürlich sind weit mehrere denkbar, welche so im eigentlichen Betrieb nicht gemacht werden sollten, dies dient hier einfach zur besseren Übersichtlichkeit und einem leichteren Verständnis der Grundidee und der Vorgehens- und Arbeitsweise solcher Adaptionsprozesse. Einerseits kann der Adaptionsprozess beliebig komplex werden um alle möglichen Fälle unterscheiden zu können (auch wenn er nach Möglichkeit so simpel wie möglich gehalten werden sollte), was natürlich auch von Nöten ist, vor allem bei größeren Projekten und anspruchsvollen Anwendungsbereichen. Andererseits wird hier im Beispiel

lediglich das Einfügen von Aktivitäten vorgenommen. Wie aber in Kapitel 2.3.2. Adaption Patterns bereits beschrieben, gibt es noch viele weitere Möglichkeiten zur Adaption von Prozessen. Natürlich könnten (und sollten) auch andere Patterns in Adaptionsprozessen realisiert und zur Anwendung gebracht werden. Dadurch ergeben sich nun noch mehr Möglichkeiten den Prozess zu optimieren und somit Zeit und Geld zu sparen und die Produktqualität zu erhöhen. Beispielsweise kann es Sinn machen nach einer Adaption weitere Anpassungen durchzuführen, da vorher festgelegte Aktivitäten bereits früher als geplant zur Anwendung kommen und somit zu einem späteren Zeitpunkt überflüssig werden (hier könnte das Delete Pattern zum Einsatz kommen).
Die Adaption in diesem Beispiel führt mehrere Einfügungen auf einmal durch und ist sehr an den Zielprozess gebunden, dies ließe sich beispielsweise durch mehrere einzelne Adaptionsvorlagen allgemeiner formulieren (siehe dazu Kapitel 6)

Die Vorteile von Adaptionsprozessen und deren theoretische Umsetzung wurden erläutert, im nächsten Kapitel findet sich eine exemplarische technische Umsetzung des hier beschriebenen Beispiels.

5. Technische Umsetzung

Für eine exemplarische Umsetzung eines Adaptionsprozesses für den Develop Solution Increment Prozess wird hier Aristaflow verwendet. Dieses bietet die dafür nötigen Anpassungsmöglichkeiten und die Möglichkeit über die offene API diese zu verwenden. Verwendet wird die Version 1.0.92 - r17 (2013-08-19 15:49) mit den entsprechenden ClientLibraries (Sammlung von bereitgestellten Klassen zur Interaktion mit der API). Diese enthalten alle nötigen Klassen und Methoden, welche für eine dynamische automatische Prozessanpassung benötigt werden. Als Programmierumgebung dient Eclipse und die Programmiersprache ist Java, da derzeit von Aristaflow noch keine andere unterstützt wird.

Der entsprechende Code für die Aktivitäten des Adaptionsworkflows ist simpel gehalten und auf den OpenUP angepasst.

Die wichtigste Klasse ist hierbei InstanceAdapter. Diese dient sozusagen als Schnittstelle zwischen dem Adaptionsprozess und dem OpenUP. Der Konstruktor dieser Klasse sieht folgendermaßen aus:

```
public InstanceAdapter() {
  String sessionID = "instanceAdapter";

  service = MultiClientService.getInstance();

  try {
              csf =
service.getAuthentication().authenticate("instanceAdapter", 9,
  "password");
  csf.setClientURIs(new URI[]
{URI.create(InetAddress.getLocalHost().getHostAddress()) });
} catch (AuthenticationException e1) {
  e1.printStackTrace();
} catch (DataSourceException e1) {
  e1.printStackTrace();
}
  catch (UnknownHostException e1) {
  e1.printStackTrace();
}

service.setAuthenticatedAgents(new ClientSessionFactory[] { csf });
sessionToken = service.getSessionToken();

wcs = (AdministrationService)service;

}
```

Hierbei wird zunächst eine Authentifizierung mit der Schnittstelle des Servers durchgeführt (wichtig für die Kommunikation mit dem Aristaflow Server), ein AdministrationService (wcs) erstellt (für spätere Zugriffe wichtig) und mit setAuthenticatedAgents() werden eben jene zugewiesen. Das bedeutet, dass diese Agents die Zugriffsrechte für Änderungen haben.
Dadurch ist sozusagen die Schnittstelle geschaffen und es kann mit dem Aristaflow Server interagiert werden. Die einzelnen Schritte wurden bereits im theoretischen Teil besprochen.
Zuerst müssen die entsprechenden Instanzen des Prozesses ermittelt werden, dies geschieht mithilfe der Get Instances Aktivität. Dahinter steckt eine Methode welche diese Aufgabe, mithilfe der geschaffenen Schnittstelle, ausführt.

```
public String getInstances (){
 SessionToken session = wcs.getSessionToken();
 UUID [] templateID = new UUID[1];
 templateID [0] = UUID.fromString("5f07fa62-7441-4782-86a6-1db3a2115bc6");
 InstanceManager iMgr = wcs.getProcessManager().getInstanceManager();
 Map<java.util.UUID,java.util.Set<java.util.UUID>> instanceListTemp =
            iMgr.findInstanceIDsOf(session, templateID,
            null, null);

Set<UUID> temp = instanceListTemp.get(templateID[0]);
for (Iterator<UUID> it = temp.iterator(); it.hasNext();) {
      UUID uuid = it.next();
      if (wcs.getProcessManager().getInstanceManager().
            getInstanceStatus(session, uuid).getExecutionStatus() ==
            ProcessConstants.InstanceExecutionStatus.IE_SUSPENDED){
                  it.remove();
          }else if (wcs.getProcessManager().getInstanceManager().
            getInstanceStatus(session, uuid).getExecutionStatus() ==
            ProcessConstants.InstanceExecutionStatus.IE_FINISHED){
                  it.remove();
          }else if (wcs.getProcessManager().getInstanceManager().
     getInstanceStatus(session, uuid).getExecutionStatus() ==
            ProcessConstants.InstanceExecutionStatus.IE_ABORTED){
                  it.remove();
          }

      }

       return temp.toString();
}
```

Die Methode liefert einen String zurück, welcher alle aktiven Prozessinstanzen (bzw. deren IDs) enthält. Dafür wird zunächst mit findInstancesIDsOf() ein Set<String> der IDs generiert. Diese Methode wird vom InstanceManager bereitgestellt. Dieses Set enthält nun aber noch Instanzen welche abgebrochen, oder bereits beendet wurden. Daher werden diese innerhalb der for Schleife nun aus der Liste entfernt, da sie für das weitere Vorgehen nicht von Relevanz sind. Schließlich wird das Set noch in einen String umgewandelt und dieser dann als Ergebnis zurückgeliefert.

Die nun vorhandene Liste von zu bearbeitenden Instanzen wird nun an die nächste Aktivität weitergegeben.
Die Methode AnalyzeInstance() bekommt einen String, welcher die ID einer Instanz aus der Liste repräsentiert, sowie die Liste von zu bearbeitenden Artefakten, übergeben. Zurück liefert sie einen Integerwert, welcher den Status der Instanz repräsentiert und für die spätere Entscheidung benötigt wird, welche Anpassung vorgenommen werden muss.

Der wesentliche Teil dieser Methode wurde simpel umgesetzt. Dabei ist zu beachten, dass einige Teile des Codes hier nicht besprochen werden, wie beispielsweise das weitere Anmelden beim Server oder das beziehen eines SessionTokens (in jeder Methode vorhanden, wird für die Identifikation der aktuellen Session benötigt), etc.

```
ProcessManager ProM = wcs.getProcessManager();
InstanceManager im = ProM.getInstanceManager();
Instance instance = im.getInstance(session, instanceID);

if (instance.getNodeState(16).
      equals(ProcessConstants.
      NodeState.NS_ACTIVATED)){
       return 1;
 }else if (measures.contains(artefact)){
       return 2;
 }else {
       return 3;
 }
```

Mithilfe der übergebenen Instanz ID kann über den Instanz Manager die entsprechende Instanz erreicht werden. Diese wird nun folgendermaßen in der if Abfrage analysiert: Befindet sich der Knoten mit der ID 16 (das ist die letzte Aktivität des OpenUP) im Zustand "aktiviert", so erhält die Instanz den Status 1. Das bedeutet es wird anschließend die Aktivität ExtendedIntegration ausgeführt werden. Sollte dies nicht der Fall sein, so wird in der zweiten if Abfrage überprüft, ob die Instanz ein Artefakt bearbeitet, welches auf der Liste der zu bearbeitenden Artefakte steht. Ist dies der Fall, so erhält die Instanz den Status 2, was bei der späteren Verarbeitung zur CompactIntegration Aktivität führt. Der dritte Fall (mit dem Status Code 3), wird anschließend mit einer StandardIntegration Aktivität weiterverarbeitet. Wie im vorherigen Kapitel erwähnt, ist diese Unterscheidung in nur drei Status eine Vereinfachung für diese Arbeit. Ebenso wird hier nicht mit tatsächlichen Referenzen auf Artefakte gearbeitet, sondern mit einer einfachen Simulation dessen, um die Arbeitsweise erläutern zu können.
Der Statuscode dient der folgenden XOR-Verzweigung im Adaptionsprozess als Eingabeparameter (siehe Abbildung 18). Hier wird entschieden welcher der drei Pfade eingeschlagen und welche Adaption zur Anwendung kommt.

Der Code zur Einfügung der Qualitätssicherungsaktivitäten ist ebenfalls überschaubar.

```
SessionToken session = wcs.getSessionToken();
InstanceChanging instanceChanging = wcs.getExecutionManager().
     getInstanceChanging();
ProcessModelFactory pmf = wcs.getProcessModelFactory();
UpdateManager updateManager = wcs.getUpdateManager();

ChangeableInstance instance;

try {
    instance = instanceChanging.getInstanceForChanging
      (session, instanceID);
} catch (InvalidInstanceStateException e) {
    throw new RuntimeException(e);
}
```

Hier werden zunächst die Objekte vorbereitet, welche für eine Ad hoc Änderung von Nöten sind (ProcessModelFactory, UpdateManager, ChangeableInstance, diese sind alle aus den Aristaflow Libraries).

```
try {StartTransaction.performOperation(instance, updateManager);
     CheckReport checkReport = new CheckReport(
               ProcessElementIdentifierTools.getInstanceIdentifier("iliad",
               instance));
     Template template = instance.getTemplate();
```

Hiermit beginnt die Transaktion mit dem Server. Um an eine Liste der Knoten zu kommen wird das passende Template zur Instanz gesucht und daraus die Knoten ausgelesen.

```
Node pred = template.getNode(16);
Node succ = template.getNode(1);
```

So können nun der Vor- und Nachfolgeknoten der Einfügung festgelegt werden. In diesem Fall handelt es sich um ExtendedIntegration, also findet die Einfügung nach dem letzten Knoten (ID 16) statt. Im folgenden wird nun die Einfügung durchgeführt:

```
Node insertNode;
if (InsertNode.isPossible(instance, pred, succ, checkReport)) {
    insertNode = InsertNode.performOperation(session, instance,
    pred, succ, updateManager);
} else {throw new RuntimeException(checkReport.
                getSimpleReportSummary());
}

nodeClone = pmf.createNode(insertNode.getID(), "Informationen
  beschaffen", node.getDescription(),
  node.getStaffAssignmentRule(), null, false, node.getIconID(),
  null, null, null, null, null);
```

```
                //which node to update???

                if (UpdateNode.isPossible(instance, insertNode, checkReport,
                          NodeProperty.NAME,
                          NodeProperty.DESCRIPTION,
                          NodeProperty.STAFF_ASSIGNMENT_RULE)) {
                    UpdateNode.performOperation(instance, nodeClone,
                             updateManager,
                            NodeProperty.NAME,
                            NodeProperty.DESCRIPTION,
                            NodeProperty.STAFF_ASSIGNMENT_RULE);

            } else {throw new RuntimeException
                   (checkReport.getSimpleReportSummary());
                }
```

Durch die Abfrage InsertNode.isPossible() kann getestet werden ob die Einfügung mit den übergebenen Parametern von Aristaflow erlaubt ist. Sollte dies der Fall sein wird diese durch InsertNode.performOperation() durchgeführt. Wurde der Knoten erfolgreich eingefügt, wird er über Update Operationen mit den nötigen Eigenschaften versorgt. In diesem einfachen Fall wird nur der Name des Knotens geändert auf "Informationen beschaffen". Das Update funktioniert wie die Einfügung über UpdateNode.isPossible() und UpdateNode.performOperation().
Der nun eingefügte Knoten ist jetzt zwar vorhanden, allerdings noch leer. Das bedeutet ihm wurde noch keine Aktivität zugewiesen. Dazu muss zuerst ein EBP, ein Executable Business Prozess, also sozusagen die Aktivität selbst, definiert werden. Dies geschieht über einige Zuweisungen von Parametern und Referenzen etc. und wird hier nicht näher behandelt.
Nun wird der EBP dem Knoten angefügt.

```
                if (ebp != null) {
                  if (AssignExecutableBusinessProcess.isPossible(instance,
                     insertNode, ebp, paramsWithoutExistingDataElements,
                     paramsWithExistingDataElementsDE, checkReport)) {
                          AssignExecutableBusinessProcess.performOperation(session,
                          instance, insertNode, ebp,
                         paramsWithoutExistingDataElements,
                         paramsWithExistingDataElements, updateManager);
                    } else {
                        throw new RuntimeException(checkReport.
                                getSimpleReportSummary());
                    }
                }
```

Dies geschieht nach dem gleichen Prinzip wie oben über einen Check (isPossible()) auf welchen die eigentliche Operation folgt (performOperation()).

Somit wurde die erste Aktivität der Qualitätssicherungsmaßnahmen erstellt und in den Prozess eingefügt. Die anderen Aktivitäten werden auf dieselbe Art und Weise erstellt und integriert. Wie die einzelnen Aktivitäten im Detail aussehen und wie sie funktionieren wird bei der Erstellung des EBP festgelegt und ist nicht Teil dieser Arbeit.
Nachdem alle Knoten eingefügt wurden folgt dann noch der abschließende Teil der Methode:

```
            finally {
                if (success) {
                    ChangeReport changeReport = instanceChanging.
                          changeAndUnlockInstance(session, instance);
                    if (changeReport.newInstanceID == null) {
                          System.out.println(changeReport.getReportSummary());
                        throw new RuntimeException("Prozess konnte nicht
                          eingefuegt werden:" + changeReport.getReportSummary());
                    }

                }
                else {
                    instanceChanging.abortInstanceChanging(session, instanceID);
                }
            }

        }
```

Im Falle eines Erfolgs, also wenn keinerlei Laufzeitfehler aufgetreten sind, werden durch changeAndUnlockInstance() die Anpassungen des Prozesses nun durchgeführt und auf dem Server gespeichert, und die Instanz wird wieder freigegeben. Das heißt sie befindet sich dann wieder im Zustand Aktiv und kann weiter ausgeführt werden.
Sollten Fehler aufgetreten sein, so wird eine Fehlermeldung generiert und die Instanz wird wieder freigegeben ohne dass die Änderungen durchgeführt werden.
Für die anderen beiden Integrationen sieht die Vorgehensweise ebenso aus. Es müssen lediglich (nach der Definition der Aktivitäten in dieser Arbeit) die Vorgänger- und Nachfolgeknoten anders gewählt werden.

Damit der Code der Aktivitäten des Adaptionsprozesses mit dem Aristaflow Server und den darauf befindlichen Templates und Instanzen kommunizieren kann, müssen noch weitere Vorkehrungen getroffen werden.

Aristaflow bietet dabei einerseits die Möglichkeit über ein externes Programm oder System die Anpassungen durchführen zu lassen. Beispielsweise ist es möglich den Code in Servlets zu programmieren und diese dann aus dem Adaptionsprozess aufzurufen. Dabei laufen diese Servlets auf einem externen Server (nicht dem Aristaflow Server).

Die andere Möglichkeit ist es, den Code sozusagen in einen bestimmten sogenannten System Execution Environment einzubetten. Dieser wird von Aristaflow zur Verfügung gestellt und ermöglicht es, die Aktivitäten des Adaptionsprozesses mit dem Server kommunizieren zu lassen. Dabei wird der Code beim AutomaticClient in einem Verzeichnis als .jar Datei hinterlegt und kann von einer Prozessinstanz aufgerufen werden. Da es sich um automatisierte Vorgänge handelt werden diese vom AutomaticClient und nicht von einem normalen Client ausgeführt.

Für diese Arbeit wurde die Methode mit dem Environment gewählt. Dafür gibt es mehrere Gründe. Unter anderem benötigt dieses Vorgehen keinen eigenen Server. Dadurch erspart man sich einerseits das konfigurieren eines solchen, andererseits können so eventuelle Verbindungs- und Kommunikationsprobleme zwischen den beiden Servern (Aristaflow Server) vermieden werden. Desweiteren lässt sich ein Adaptionsworkflow mit Environment leichter an den Kunden ausliefern, da alles kompakt und in der Aristaflow Architektur zusammengehörig ist. Der Aufbau eines solchen Environment ist ebenfalls simpel, als Beispiel die run() Methode zur Aktivität getInstances:

```
public void run() {

logger.log(Level.INFO, "Coseeek Environment entered run method...");
logger.log(Level.INFO, "Adapt Environment stepping...");

final RuntimeEnvironment runenv = sessionContext.getRuntimeEnvironment();
if (runenv == null) {
       String message = "No appropriate runtime environment supplied -
                        failing.";
  }

  logger.log(Level.INFO, "Adapt Environment starting latch and
            listener...");


  InstanceAdapter adapter = null;

     try {
            adapter = InstanceAdapter.getInstance();
     } catch (ServiceNotKnownException e) {
            // TODO Auto-generated catch block
```

```
                e.printStackTrace();
            } catch (AbortServiceException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        adapter.getInstances();

        logger.log(Level.INFO, "Adapt Environment all done waiting for latch...");
        // wait for close
        try {

            sessionContext.getRuntimeEnvironment().applicationClosed();

        } catch (Exception e1) {

            }

    }
```

Hierbei wird die entsprechende Methode aufgerufen und anschließend die Anwendung wieder geschlossen.

Durch dieses einfache Vorgehen lassen sich die gewünschten Einfügungen vornehmen. Auch andere Change Patterns wie beispielsweise das Löschen oder das Verschieben einer Aktivität lassen sich auf ähnliche Art und Weise in Aristaflow umsetzen. Selbst wenn man von den Vereinfachungen, welche für diese Arbeit vorgenommen wurden, absieht, ist dies eine gute und effiziente Möglichkeiten die automatische Anpassung von Prozessen zu realisieren.

6. Zusammenfassung und Ausblick

Kaum ein anderer Industriezweig entwickelt sich und wächst so schnell wie der der Softwareentwicklung. Um bei diesem rasanten Tempo und bei großer Konkurrenz auf dem Markt bestehen zu können und erfolgreich zu sein muss man sich und seine Arbeitsmethoden ständig verbessern und möglichst effizient und effektiv arbeiten. Welche Modelle und Konzepte man dabei zur Verfügung hat wurde hier erläutert. Auf einem so dynamischen Markt reicht es nicht, sich einmal alles zurecht zu legen um dann immer auf dieselbe Art und Weise zu arbeiten, vielmehr muss man ständig auf Veränderungen eingehen, neue Technologien akquirieren und sich selbst stetig weiterentwickeln. Im hier

betrachteten Fall der Prozessmodelle wäre eine solche Weiterentwicklung die hin zu mehr Flexibilität und dynamischen Prozessanpassungen.
Leider ist der Markt derzeit noch weit davon entfernt diese Punkte zu genüge umzusetzen. Von den derzeit auf dem Markt verbreiteten und kommerziellen BPMS sind die meisten zu statisch aufgebaut und bieten kaum, wenn überhaupt, Möglichkeiten für dynamische Anpassungen und Ad hoc Änderungen zur Laufzeit. Die Notwendigkeit solcher Systeme ist offensichtlich und auch bekannt, jedoch ist es noch ein langer Weg bis die etablierten Systeme hinreichend erweitert werden oder neue, bessere Systeme auf den Markt kommen.
Eine Ausnahme stellt hier Aristaflow dar. Mit der hohen Benutzerfreundlichkeit (Drag and Drop, Plug and Play, Correctness by Construction, etc.), der nötigen Offenheit für Erweiterungen (API, Schnittstellen, etc.) und den bisher einzigartigen Möglichkeiten für flexibles Prozessmanagement (Ad hoc Änderungen, etc.) bietet Aristaflow hier genau das, was andere Systeme so stark vermissen lassen.
In dieser Arbeit wurde exemplarisch gezeigt welche verschiedenen Ansätze es für Prozessanpassungen gibt und welche Vor- und Nachteile die einzelnen Ansätze bieten. Daraus wurde ersichtlich, dass eine automatische dynamische Prozessanpassung mithilfe von Adaptionsprozessen erstrebenswert ist und wie ein solcher (vereinfacht) realisiert werden könnte. Ebenso wurde erläutert wie komplex das Gebiet der dynamischen Prozessanpassungen ist und dass es noch einiges an Forschung und Entwicklung benötigt bis diese Ansätze zu genüge in die Realität umgesetzt werden können.

Da es den Umfang dieser Arbeit übersteigen würde, wurde hier nicht behandelt wie sich solche Adaptionsprozesse modularisieren lassen könnten. So könnte beispielsweise für die verschiedenen Patterns jeweils ein Adaptionsprozess erstellt werden, welche sich dann nach dem Baukastenprinzip (also eine Auswahl der benötigten Adaptionsmodule) zu einem vollständigen Prozess zusammenbauen ließen. Dadurch ließe sich leicht für jeden Prozess der passende Adaptionsprozess erstellen. Die Wiederverwendbarkeit und Wartbarkeit der einzelnen Module wäre sehr hoch und der Prozess wäre auch

für Laien einfach zu erstellen. Dies kann in weiteren Arbeiten überprüft und implementiert werden.

Literaturverzeichnis

[1] J. Ludewig, H. Lichter: Software Engineering, 2. Auflage, dpunkt, 2010

[2] G. Pomberger, W. Pree: Software Engineering, Hanser, 2004

[3] Enzyklopädie der Wirtschaftsinformatik, http://enzyklopaedie-der-wirtschaftsinformatik.de, 19.01.2014

[4] W. Royce: Managing the Development of Large Systems, IEEE WESCON, pp. 1-9, IEEE Computer Society, 1970

[5] B. Boehm: Software Engineering Economics, IEEE Transactions on Software Engineering, 10 (1): 4-21, IEEE Computer Society, 1984

[6] K. Pfetzing, A. Rohde: Ganzheitliches Projektmanagement, Vol. 2, Verlag Goetz Schmidt, 2009

[7] IABG: Das V-Modell, http://v-modell.iabg.de, 19.01.2014

[8] Graphical Development Process Assistant, http://informatik.uni-bremen.de/gdpa, 18.01.2014

[9] Wikipedia: Das V-Modell, http://de.wikipedia.org/wiki/V-Modell, 18.01.2014

[10] IABG: Das V-Modell XT, http://v-modell.iabg.de/v-modell-xt-html, 19.01.2014

[11] IABG: V-Modell Werkzeuge, http://v-modell.iabg.de/index.php?option=com_content&task=view&id=27&Itemid=51, 19.01.2014

[12] P. Kruchten: The Rational Unified Process: An Introduction, Addison-Wesley Professional, 2004

[13] pinga™solutions, http://www.pingasolutions.com, 18.01.2014

[14] Eclipse Foundation: Eclipse Process Framework, http://www.eclipse.org/epf/, 20.01.2014

[15] Scrum Alliance®, http://scrumalliance.org/, 20.01.2014

[16] Manifesto for Agile Software Development, www.agilemanifesto.org/, 20.01.2014

[17] it-agile: Scrum, http://www.it-agile.de/wissen/methoden/scrum, 20.01.2014

[18] B. Weber, M. Reichert, S. Rinderle-Ma: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems, Data and Knowledge Engineering, 66(3): 438-466, Elsevier Science, 2008

[19] P. Dadam, M. Reichert, S. Rinderle-Ma, K. Göser, U. Kreher, M. Jurisch: Von ADEPT zur Aristaflow BPM Suite - Eine Vision wird Realität, Ulmer Informatik-Berichte, UIB-2009-02, 2009

[20] G. Grambow, R. Oberhauser, M. Reichert: Contextual Injection of Quality Measures into Software Engineering Processes, Int'l Journal on Advances in Software, 4(1&2): 76-99, IARIA, 2011

[21] G. Grambow, R. Oberhauser: Towards Automated Context-aware Software Quality Management, Fifth International Conference on Software Engineering Advances (ICSEA 2010), pp. 347-352, IEEE Computer Society Press, 2010

[22] G. Grambow, R. Oberhauser, M. Reichert: Employing Semantically Driven Adaptation for Amalgamating Software Quality Assurance with Process Management, Second Int'l Conf. on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE'10), pp. 58-67, Xpert Publishing Services, 2010

Abbildungsverzeichnis