Lifecycle-Modelle

Bachelorarbeit II

eingereicht von: **Martin Papai**
Matrikelnummer: 0810470039

im Fachhochschul-Bachelorstudiengang Wirtschaftsinformatik

der Ferdinand Porsche FernFH Gesellschaft zur Erhaltung und Durchführung von Fachhochschul-Studiengängen

zur Erlangung des akademischen Grades <einer / eines>

Bachelor of Arts in Business

Betreuung und Beurteilung: Univ. Prof. Dr. Tom Gross

Wien, Juni 2011

Ehrenwörtliche Erklärung

Ich versichere hiermit,

1. dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Inhalte, die direkt oder indirekt aus fremden Quellen entnommen sind, sind durch entsprechende Quellenangaben gekennzeichnet.

2. dass ich diese Bachelorarbeit bisher weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit zur Beurteilung vorgelegt oder veröffentlicht habe

3. dass die vorliegende Fassung der Arbeit mit der eingereichten elektronischen Version in allen Teilen übereinstimmt.

Trumau, 11. Mai 2011 _____

Unterschrift

Zusammenfassung

Kurzzusammenfassung: Lifecycle-Modelle

In dieser Arbeit werden die verschiedenen Lifecycle-Modelle für die Entwicklung und Einführung von Softwarelösungen beschrieben. Das Problem ist, dass nicht jedes Vorgehensmodell für jede Art von Software geeignet ist. Ziel war es einen Überblick über alle Modelle mit deren Eigenschaften, Vor- und Nachteilen sowie deren Einsatzgebieten zu geben. Weiters wird aufgezeigt wie in den einzelnen Phasen der Modelle auf die Anforderungen und Bedürfnisse der Benutzer und der Anwender eingegangen werden kann.

Zum Verständnis wird der Begriff „Vorgehensmodell“ detailliert erklärt. Weiters werden einige Eigenschaften wie Sichten, Anforderungen, Komponenten oder Arten von Vorgehensmodellen angeführt. Das Hauptaugenmerk liegt bei den verschiedenen Arten der Vorgehensmodelle und bei dem bekanntesten Vertreter aus der jeweiligen Art.

Der Abschluss des Hauptteiles dieser Arbeit besteht aus dem direkten Vergleich der Modelle. Einige relevante Fragen über das Einsatzgebiet, Umfang des Projektes oder Komplexität der Software werden hier beantwortet.

Zum Schluss wird aus den beschriebenen Vorgehensmodellen noch ein eigens entwickeltes Modell zur erfolgreichen Data-Warehouse-Entwicklung vorgestellt.

Schlagwörter:

Vorgehensmodell, Lifecycle-Modelle, Phasen, Eigenschaften, Anforderungen, Komplexität, Vergleich, Software, Vor- und Nachteile, Data Warehouse

Abstract: Lifecycle models

During the course of this thesis different lifecycle models concerning the development and implementation of software-based solutions shall be described and detailed; with a view to providing a concise overview over the different models, including their different characteristics areas of application as well as „Pros“ and „Cons“ of the respective solutions. Subsequently this thesis will highlight potential responses to users‘ requirements and needs occurring during the different phases of the respective lifecycle models.

For methodological clarity the term „lifecycle-model“ will be defined in detail. In addition, certain features/properties such as view, requirements, components of different approaches to solving related issues. The focus is on the different approaches as well as most common approaches and solutions thereto.

The closing chapter comprises a comprehensive comparison of different models; providing answers concerning areas of use of the software, scope of the project as well as the software’s complexity.

In conclusion, the thesis will develop a separate role model of its own, comprised of the ideal components of the models analysed.

Keywords:

lifecycle model, phases, properties, features, requirements, complexity, comparison, software, „pros“ and „cons“

Inhaltsverzeichnis

EHRENWÖRTLICHE ERKLÄRUNG II

ZUSAMMENFASSUNG III

INHALTSVERZEICHNIS V

1. EINLEITUNG 1

2. WAS IST EIN LIFECYCLE-MODELL 3

3. SICHTEN AUF LIFECYCLE-MODELLE 5

4. ANFORDERUNGEN 6
4.1 Prinzipielle Anforderungen 6
4.2 Spezielle Anforderungen 6

5. KOMPONENTEN VON LIFECYCLE-MODELLEN 7

6. TÄTIGKEITSBEREICHE 8

7. DARSTELLUNG VON LIFECYCLE-MODELLEN 10
7.1 Prozessmodell 10
7.2 Rollenmodell 11

8. FORMALISIERUNG VON LIFECYCLE-MODELLEN 12

9. NUTZEN UND WIRTSCHAFTLICHE SICHT VON LIFECYCLE-MODELLEN 13

10. ARTEN VON LIFECYCLE-MODELLEN 14
10.1 Sequentielle Vorgehensmodelle 14
10.1.1 Das Wasserfallmodell 17
10.2 Prototyping 19
10.3 Spiralmodell 22
10.4 V-Modell 97 23
10.5 V-Modell XT 26
10.6 Iterative Vorgehensmodelle 28
10.6.1 Rational Unified Process (RUP) 32
10.7 Agile Vorgehensmodelle 35
10.7.1 Extreme Programming (XP) 36
10.7.2 Scrum 38

11. PROBLEME BEI LIFECYCLE-MODELLEN 40

12. LIFECYCLE-MODELLE IM VERGLEICH 41

13. ENTWICKLUNG EINES LIFECYCLE-MODELLS ZUR DATA-WAREHOUSE-ENTWICKLUNG 43
13.1.1 Phasen 44
13.1.2 Ablauf 45

13.1.3 Data-Warehouse-Schichten *46*

14. SCHLUSS **47**

LITERATURVERZEICHNIS **48**

ABBILDUNGSVERZEICHNIS **52**

TABELLENVERZEICHNIS **53**

ANHANG **54**

A. Abkürzungsverzeichnis 54

1. Einleitung

In dieser Arbeit werden die verschiedenen Lifecycle-Modelle aus der Softwaretechnik näher beschrieben und miteinander verglichen. Besonderes Augenmerk liegt auf der Vorgangsweise bei der Entwicklung einer Software. Hervorzuheben sind hier die einzelnen Phasen die durchschritten werden müssen und wie diese im Detail aufgebaut und organisiert sind. Die Modelle zeigen wo ihre Stärken und Schwächen liegen und welche Softwarearten für die einzelnen Lifecycle-Modelle geeignet sind.

Das Ziel dieser Bachelorarbeit ist es die verschiedenen Vorgehensmodelle den Benutzern, im besonderen Softwareprogrammierern und Softwareprojektmanagern, die Vor- und Nachteile der Modelle näherzubringen und zu zeigen wie auf die Anforderungen und Bedürfnisse von Anwendern und Programmierern eingegangen werden kann. Es gibt zu den Vorgehensmodellen viele Fachbücher und Internetseiten, die sich mit diesem Thema beschäftigen, jedoch fehlt es an einer kompakten Zusammenfassung sämtlicher Lifecycle-Modelle, bei der kurz und bündig die einzelnen Modelle näher beschrieben werden, die Vor- und Nachteile aufgelistet sind und weitere Eigenschaften verglichen werden. In dieser Arbeit wurde sich genau diese Thematik als Ziel gesetzt. Ein weiteres Ziel war es diese komplexe Materie auch für Außenstehende, dh. Leute ohne IT Background, interessant und verständlich aufzubereiten. Die letzte Zielsetzung war es ein eigenes Vorgehensmodell auf Basis der zu beschreibenden Vorgehensmodelle zu entwickeln und vorzustellen.

Hauptnutzen dieser Arbeit werden Softwareprogrammierer und Softwareprojektmanager in ihren ersten Berufsjahren haben, da sie erst Erfahrung sammeln und für sich die beste Methode finden müssen. Hier liefert diese Arbeit eine sehr gute Grundlage um einen Überblick über die verschiedenen Modelle zu bekommen. Weiters sehe ich diese Arbeit als Gewinn für meine berufliche Tätigkeit als Controller, wo Softwareentwicklung, im Besonderen die Entwicklung

von Datenbanken und Data-Warehouse-Systemen, genauso eine wichtige Rolle spielt wie wirtschaftliche Analysen zu erstellen.

2. Was ist ein Lifecycle-Modell

Ein Lifecycle-Modell ist vielen Softwareentwicklern und Projektleitern auch als Vorgehensmodell ein Begriff. Ein Lifecycle-Modell wird in der Software-Entwicklung auch als Software-Lebenszyklus bezeichnet. Das Vorgehensmodell ist ein ausgefeilter Plan mit zeitlichen und sachlichen Vorgaben. Es legt die Reihenfolge einzelner Schritte mit ihren jeweiligen Ergebnissen fest und regelt detailliert, **wer wann was** in einem Projekt zu tun hat. Es dient zur Steuerung der Software-Entwicklung von der Konzeption bis zum Einsatz im Betrieb inklusive der im Betrieb anfallenden Änderungen einer Software. Das Vorgehensmodell gestaltet die Software-Entwicklung übersichtlicher und erleichtert die Beherrschung der Komplexität.[1]

Ein Vorgehensmodell unterteilt den Entwicklungsprozess in überschaubare, zeitlich und inhaltlich begrenzte Aktivitäten, deren Ergebnisse einzelne Teilprodukte sind.[2] Das Vorgehensmodell gibt die Methoden und Werkzeuge vor, die die Erarbeitung von Ergebnissen innerhalb einer Aktivität unterstützen.[3] Die Aktivitäten werden zu Phasen (z.B. Analyse, Design, Implementierung, Fehlersuche, Test) zusammengefasst und diese dann evtl. mit geringen Modifikationen einmal (z.B. Wasserfallmodell) oder mehrmals (z.B. Spiralmodell) durchlaufen. Bei mehrmaligen Durchläufen erfolgt eine iterative (d.h. wiederholte) Verfeinerung der einzelnen Software-Komponenten.[4]

Ein Lifecycle-Modell ist von vielen Faktoren abhängig. Ein wesentlicher Faktor ist die Komplexität des Projektes. Ein weiterer relevanter Faktor ist das Projektteam bzw. die am Projekt beteiligten Personen. Je statischer und eindimensionaler die

---

[1] vgl. Helmke, Höppner, & Isernhagen, 2007, S. 170

[2] vgl. Helmke, Höppner, & Isernhagen, 2007, S. 170

[3] vgl. Wallmüller, 2001, S. 129

[4] vgl. Helmke, Höppner, & Isernhagen, 2007, S. 170

Betrachtungsweise allerdings ist, desto weniger hat sie in der Regel mit der Praxis der Software-Entwicklung zu tun.[5]

[5] vgl. Helmke, Höppner, & Isernhagen, 2007, S. 170

3. Sichten auf Lifecycle-Modelle

Das Vorgehensmodell wird grundsätzlich von zwei Sichten betrachtet. Die statische Sicht beschreibt im Grunde den Aufbau eines Modells, wo hingegen die dynamische Sicht nur das Verhalten des zu modellierenden Systems beschreibt.[6]

Eine detaillierte Auflistung der Aufgaben und des Verhalten des Modells sieht wie folgt aus.

Statische Sicht:[7]

- interne Architektur, z.B. Strukturierungstiefe
- Aufteilung in Phasen/Aktivitäten oder Tätigkeitsfelder
- Form der Dokumentation. z.B. Ausgliederung wiederholter Definitionen gesonderter Beschreibungsteile

Dynamische Sicht:[8]

- strukturelle Anpassungen. z.B. Weglassen und Ergänzen von Aktivitäten
- Aktivitäten werden mit Personen besetzt
- Aktivitäten erhalten zeitliche Einordnung mit Start- und Endzeit

[6] vgl. Kolditz, 2009, S. 62

[7] vgl. Kirk, 2010, S. 22

[8] vgl. Kirk, 2010, S. 22

4. Anforderungen

Die Anforderungen an ein Vorgehensmodell sind sehr weit greifend. Diese werden in prinzipielle sowie in spezielle Anforderungen eingeteilt.

4.1 Prinzipielle Anforderungen[9]

Folgende Anforderungen muss ein Vorgehensmodell unbedingt erfüllen:

- allgemeingültig
- anwendbar
- anpassbar
- regelbasiert
- zielorientiert
- wiederverwendbar

4.2 Spezielle Anforderungen[10]

Diese unten angeführten speziellen Anforderungen sind sehr schwierig umzusetzen, aber extrem wichtig für ein erfolgreiches Vorgehensmodell:

- systematisch
- modular
- konfigurierbar
- detailliert
- Kundenintegration
- Projektmanagement-Unterstützung
- EDV-Unterstützung

---

[9] vgl. Apitz & Rasch, 2011, S. 14

[10] vgl. Apitz & Rasch, 2011, S. 14

5. Komponenten von Lifecycle-Modellen

Ein Vorgehensmodell bietet zu jeder Aktivität der Systementwicklung eine Unterstützung in Form einer Checkliste, Leitlinie, Methode, eines Templates oder einer Sammlung von Best Practices.[11]

Im Allgemeinen ist ein Vorgehensmodell des Software Engineering eine sehr umfangreiche, für viele denkbare Ablauf-Varianten geeignete Beschreibung von

- Aktivitäten (Phasenmodell) und
- Ergebnistypen (Produktmodell).[12]

Das Phasenmodell als Komponente der Vorgehensmodelle sorgt für die Einteilung des gesamten Software Engineering Prozesses in Hauptphasen (zum Beispiel: Voruntersuchung, Fachkonzept, DV-Konzept, Realisierung, ...) und dazugehörige Arbeitsschritte der Software-Entwicklung. Das Phasenmodell liefert eine standardisierte Abfolge der Aktivitäten für die Prozessmodellierung und die Anwendungsentwicklung (Software Life Cycle). Für die zeitliche Anordnung bzw. zustandsabhängige Ausführung der Aktivitäten wird der Begriff Prozessmodell verwendet. Phasenmodelle ohne Berücksichtigung der Produkte stellen nur eine unzureichende Basis für das Entwickeln von Anwendungen dar.[13]

---

[11] vgl. Informatik Forum Simon GmbH, 2011

[12] vgl. Informatik Forum Simon GmbH, 2011

[13] vgl. Informatik Forum Simon GmbH, 2011

6. Tätigkeitsbereiche[14]

Die Aktivitäten, die im Rahmen von Vorgehensmodellen definiert werden, können auf einer allgemeinen Ebene vier Tätigkeitsbereichen zugeordnet werden:

- Projektmanagement: Dieser Tätigkeitsbereich umfasst die Planung, Steuerung und Kontrolle eines Softwareentwicklungsprojektes.
- Konfigurationsmanagement: Dieser Tätigkeitsbereich umfasst die Identifikation der Konfigurationen eines Systems zu diskreten Zeitpunkten zum Zwecke der systematischen Steuerung von Konfigurationsänderungen und der Aufrechterhaltung der Vollständigkeit und Verfolgbarkeit der Konfigurationen während des gesamten SW-Lebenszyklus.
- Qualitätsmanagement: Dieser Tätigkeitsbereich umfasst die Planung, Steuerung und Kontrolle der Qualität eines Softwaresystems (Produkt) sowie des Prozesses seiner Erstellung.
- Systementwicklung: Dieser Tätigkeitsbereich umfasst die unmittelbare Erstellung des Softwaresystems. Dagegen nehmen die anderen drei Tätigkeitsbereiche nur eine mittelbare Rolle bei der Systementwicklung ein.

[14] vgl. Fettke, Intorsureanu, & Loos, 2011

Die nachstehende Abbildung 1 zeigt den Zusammenhang der Tätigkeitsbereiche, Komponenten, des Vorgehensmodells und dessen Methoden und Werkzeuge.

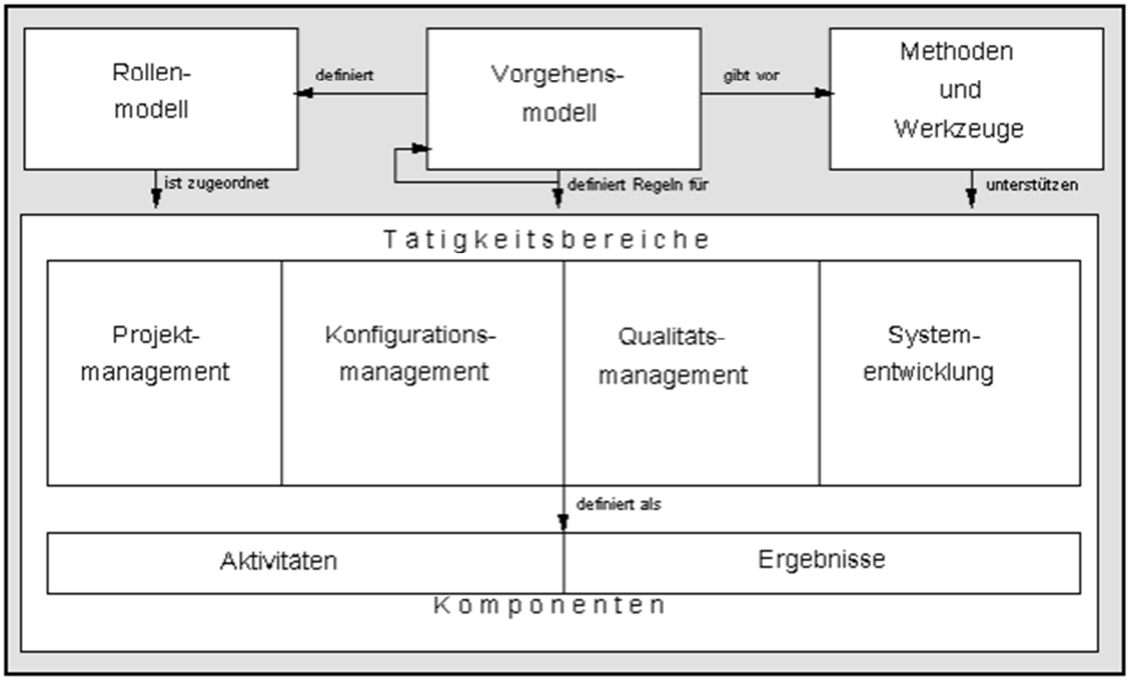

Abbildung 1: Tätigkeitsbereiche[15]

[15] Quelle: Biskup & Fischer, 2005, S. 3

7. Darstellung von Lifecycle-Modellen

Aus der genannten Definition von Vorgehensmodellen als eine Abfolge von Aktivitäten und Ergebnissen folgt die Visualisierung. Man unterscheidet im Wesentlichen die Darstellung des Prozessmodells, also die Darstellung der eigentlichen Aktivitäts-Ergebnis-Sequenz, sowie deren Eigenschaften und die Darstellung des Rollenmodells.[16]

Da Rollen und Aufgaben nicht gänzlich unabhängig von den Aktivitäten betrachtet werden können, gibt es auch Mischformen.[17]

7.1 Prozessmodell

Das Prozessmodell kann mit mehreren Modellierungssprachen beschrieben werden. Die Abbildung 2 zeigt einen kurzen Überblick von drei Beschreibungsmöglichkeiten.

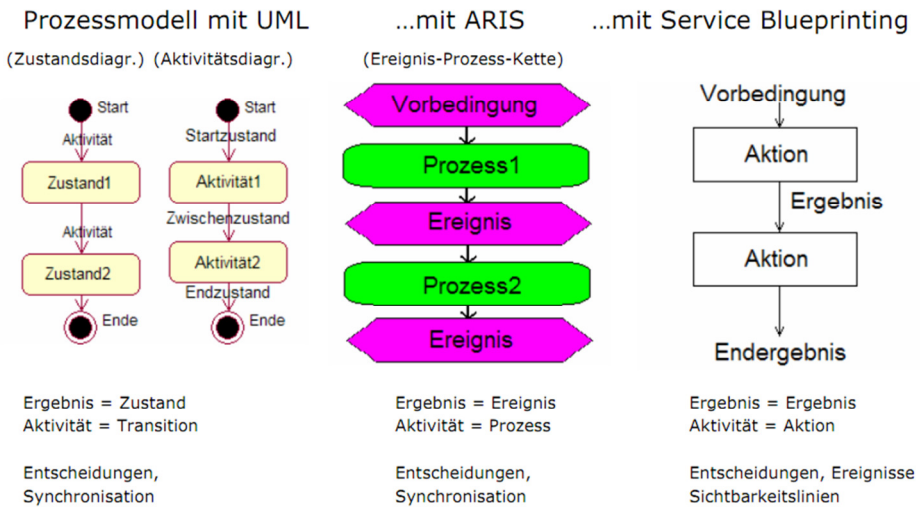

Abbildung 2: Visualisierung Prozessmodell[18]

[16] vgl. Apitz & Rasch, 2011, S. 25

[17] vgl. Apitz & Rasch, 2011, S. 25

7.2 Rollenmodell

Wie beim Prozessmodell gibt es auch beim Rollenmodell mehrere Arten zur Beschreibung der Rollen. Die Abbildung 3 zeigt einen kleinen Einblick in Möglichkeiten der Rollenbeschreibung.

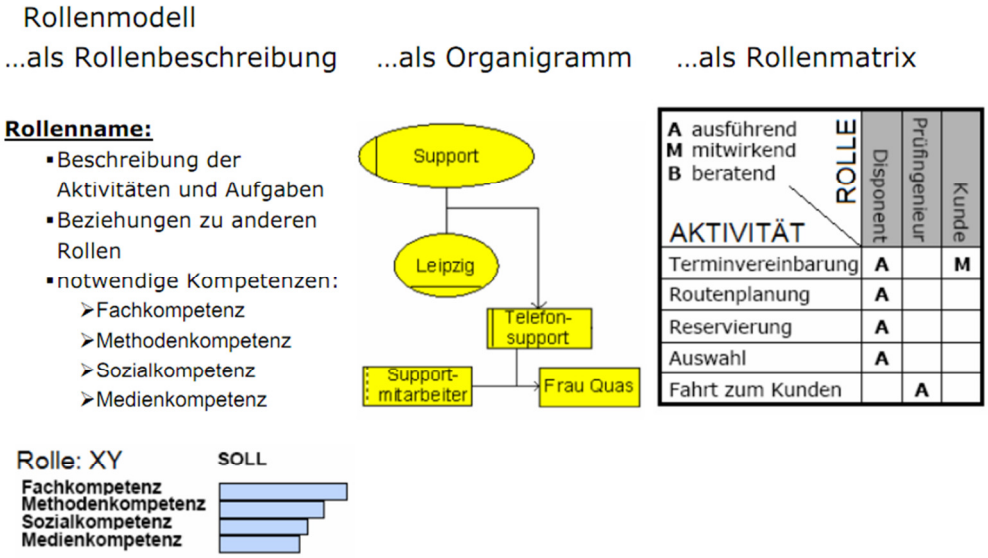

Abbildung 3: Visualisierung Rollenmodell[19]

---

[18] Quelle: Apitz & Rasch, 2011, S. 26

[19] Quelle: Apitz & Rasch, 2011, S. 27

8. Formalisierung von Lifecycle-Modellen

Bei der Formalisierung von Lifecycle-Modellen ist das Ziel eine einfache Entwicklung von Vorgehensmodellen. Bisher wurde das Modell in natürlicher Sprache erstellt, jedoch wird es zukünftig eine Unterstützung durch spezielle Modellierungssprachen geben.[20]

Des Weiteren können die Sprachen wiederrum unterteilt werden in:[21]

- Modellierungssprachen:
  - Vorwiegend aus der Softwareprozessmodellierung abgeleitet
  - Ziel: Vorgehensmodell für Werkzeuge zugänglich machen
  - Sprache als Leitfaden → systematische Erschließung (Qualität)
  - keine allgemein beste Modellierungssprache (projektabhängig!)
- Prozessentwurfssprachen:
  - abstrakte Beschreibung von Prozessen → Ziel: besseres Verständnis existierender Prozesse
  - leicht pflegbare Darstellung von Prozessen für jedermann lesbar Verwendung von natürlicher Sprache und Tabellen
- Prozessimplementierungssprachen:
  - Feingranulare Beschreibung von Prozessen
  - für Werkzeuge/Maschinen interpretierbar → nur für geschultes Personal lesbar

[20] vgl. Apitz & Rasch, 2011, S. 28

[21] vgl. Apitz & Rasch, 2011, S. 28

9. Nutzen und wirtschaftliche Sicht von Lifecycle-Modellen

Nutzen:[22]

- Hilfsmittel für die Projektsteuerung
- Mittel zur Risikominderung
- Direkte Kostenreduzierung (ressourcen-schonend)
- Indirekte Kostenreduzierung (Vermeiden von Personalaufwand und Doppelentwicklungen)

Wirtschaftliche Sicht:[23]

- Wirtschaftlichkeit leider nicht genau nachweisbar Verhältnis zwischen Aufwand und Nutzen nicht eindeutig
- kein Bedarf bei kleinen Projekten (übermäßiger Bürokratismus) → nur in großen Projekten mit hohen Qualitätsanforderungen

[22] vgl. Apitz & Rasch, 2011, S. 41

[23] vgl. Apitz & Rasch, 2011, S. 41

10. Arten von Lifecycle-Modellen

Es gibt eine Vielzahl von Arten von Vorgehensmodellen. Je nach Autor und Anwendungsbereich werden diese individuell gegliedert und eingeordnet.

Es existieren Vorgehensmodelle die in Gruppen mit gemeinsamen Eigenschaften unterteilt werden. Jede Gruppe unterscheidet Vorgehensmodelle hinsichtlich verschiedener Kriterien, wie zum Beispiel hinsichtlich der Bezeichnung einzelner Entwicklungsschritte oder der Zuordnung der Aktivitäten und der Ergebnisse zu bestimmten Entwicklungsschritten. [24]

In dieser Arbeit wurden die Modelle in Sequentielle Vorgehensmodelle, Prototyping, Spiralmodell, V-Modell, iterative und agile Vorgehensmodelle gegliedert.

10.1 Sequentielle Vorgehensmodelle

Sequenzielle Vorgehensmodelle stellen als die klassischen Vorgehensmodelle die älteste Kategorie von Vorgehensweisen in der Softwareentwicklung dar. Wie der Begriff „sequenziell" bereits andeutet, zeichnen sich diese Vorgehensmodelle dadurch aus, dass ihnen eine relativ strenge Ablauffolge von verschiedenen Phasen immanent ist. Ihnen liegt damit eine Linearität zugrunde. So gesehen beruht die sequenzielle Vorgehensweise auf der Annahme, dass die einzelnen Phasen klar voneinander abgegrenzt werden können.[25]

Jede Phase schließt dabei mit einem oder mehreren Ergebnissen ab, die geprüft und abgenommen werden können. Der Abschluss dieser Phasen entspricht dann zugleich den zuvor festgelegten Projektmeilensteinen. Die darauffolgende Phase

---

[24] vgl. Koord & Krauter, 2009

[25] vgl. Hoffmann, 2010, S. 41f

wird erst dann begonnen, wenn die vorherige Phase vollständig abgeschlossen ist.[26]

Die folgende Abbildung 4 zeigt ein Beispiel für ein sequentielles Vorgehensmodell, dass aus 5 Phasen besteht. Die Anzahl der Phasen werden je nach Autor und Anwendungsfall unterschieden.[27] Folglich können auch die einzelnen Phasen unterschiedliche Bezeichnungen aufweisen.

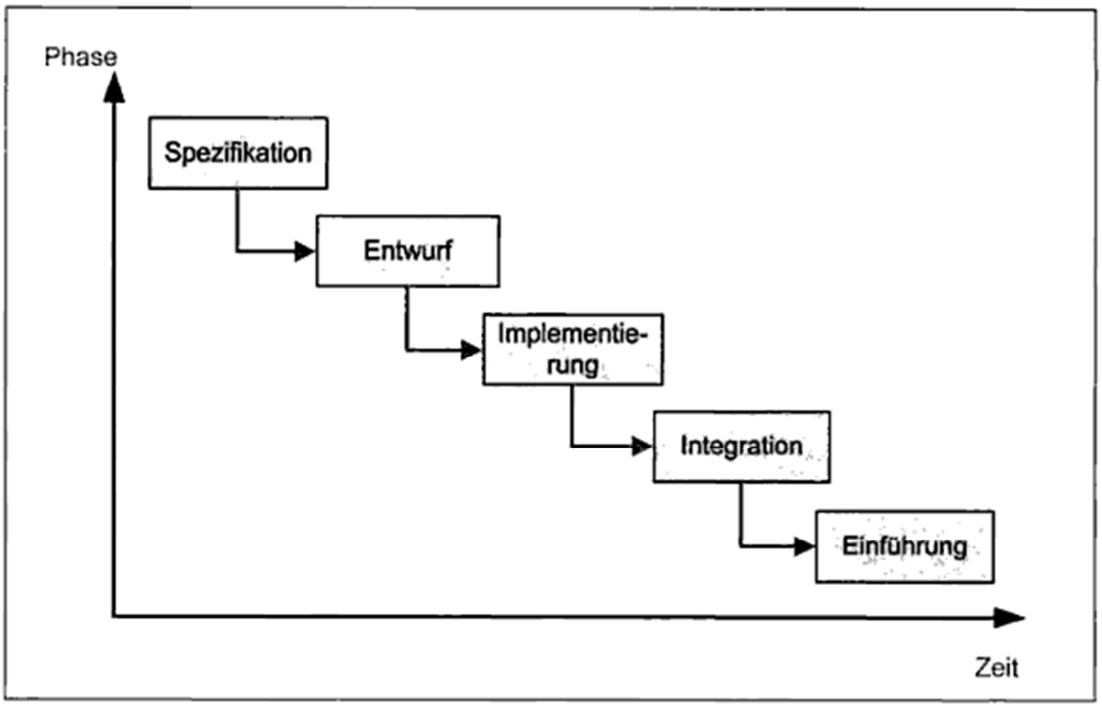

Abbildung 4: Sequentielles Vorgehensmodell[28]

In der Phase „Spezifikation" wird das Problem analysiert, die Ist-Analyse durchgeführt und anschließend das Projektthema definiert. Nach dieser Phase steht fest, was als Endprodukt entstehen wird. In der Entwurfsphase wird das

[26] vgl. Hoffmann, 2010, S. 42

[27] vgl. Ruf & Fittkau, 2008, S. 29

[28] Quelle: Ruf & Fittkau, 2008, S. 30

Soll-Konzept entwickelt und bestimmt, wie das Projekt durchlaufen werden soll. Danach werden die Pläne aus der Entwurfsphase umgesetzt und anschließend wird das fertige Produkt, vor der Auslieferung, getestet. Nach den erfolgreichen Tests erfolgen die Einführung und Inbetriebnahme des Produktes, das Feintuning und die Wartung, falls diese vorgesehen wurden. Von jeder Phase werden Entwicklungsergebnisse in Form von Anforderungs- bzw. Entwurfsdokumenten geliefert und in ihrer nachfolgenden Phase weiterverarbeitet.[29]

Die Kennzeichen der sequentiellen Vorgehensmodelle:[30]

- Am Ende einer Phase steht ein Phasenprodukt, das geprüft werden kann.
- Die Nachfolgephase wird gestartet, wenn die Vorgängerphase abgeschlossen ist.
- Die erstellten IT-Systeme können nach Abschluss der Entwicklung gegen die in der Spezifikation aufgestellten Anforderungen geprüft werden.

Ein sequentielles Vorgehensmodell kann nicht bei jedem Projekt bzw. bei jeder Softwareentwicklung angewandt werden. Wenn folgende Voraussetzungen gegeben sind, dann ist eine Anwendung eines Modells, das sequentiell aufgebaut ist, möglich:[31]

- Es soll ein IT-System mit relativ fixen Anforderungen entwickelt werden.
- Während der Entwicklungszeit ist nicht damit zu rechnen, dass sich die Anforderungen ändern.
- Es handelt sich um ein kleineres Entwicklungsprojekt, an dem ca. ein bis zwei Entwickler ca. 3 Monate beschäftigt sind.

---

[29] vgl. Koord & Krauter, 2009

[30] vgl. Ruf & Fittkau, 2008, S. 30

[31] vgl. Ruf & Fittkau, 2008, S. 30

Die Vorteile dieser Modelle sind:[32]

- klar strukturierte Rollen und Aktivitäten, daher entsteht ein minimaler Managementaufwand
- sehr übersichtlich und leicht verständlich
- Zentrales Prinzip ist die Trennung von WAS und WIE.

Weiters gibt es auch Nachteile dieser Modelle:[33]

- Vernachlässigung der Risikogefahr, da das was am Anfang definiert wurde, bis zum Projektabschluss durchgezogen wird.
- Die Benutzerbeteiligung ist nur am Anfang des Projekts und beim Projektabschluss vorhanden.
- Gefahr von falscher Prioritätensetzung, aufgrund der Fokussierung des Dokumentes anstelle des Produktes.
- Änderungen während des Entwicklungsprozesses sind nicht möglich
- Das Gesamtprojekt wird verzögert, wenn eine der Phasen nicht zeitgemäß abgeschlossen wird.

Typische Vertreter dieser Gruppe sind

- das Phasen-,
- das Wasserfall- oder
- das Schleifenmodell.[34]

10.1.1 Das Wasserfallmodell

Das wohl berühmteste Vorgehensmodell ist das klassische Wasserfallmodell. Es geht in seinen Ursprüngen auf ROYCE5 zurück und teilt den

[32] vgl. Koord & Krauter, 2009

[33] vgl. Koord & Krauter, 2009

[34] vgl. Koord & Krauter, 2009

Softwareentwicklungsprozess in die Phasen Vorstudie, Analyse, Design, Implementierung, Test und Betrieb ein (Abbildung 5). Wie bereits erwähnt können die Bezeichnungen der einzelnen Phasen abweichen. Zusätzliche Verfeinerungen beim Ablauf sind verbreitet. Wesentlicher Bestandteil dieses Modells ist die Vorstellung, dass eine Phase erst dann begonnen wird, wenn die jeweils vorangehende Phase komplett abgeschlossen ist. Den Abschluss einer Phase bildet eine Meilensteinsitzung, in der das Phasenergebnis begutachtet wird, um dann zu entscheiden, ob einzelne Aktivitäten zu wiederholen sind, oder ob das überprüfte Ergebnis in die Folgephase einfließen kann.[35]

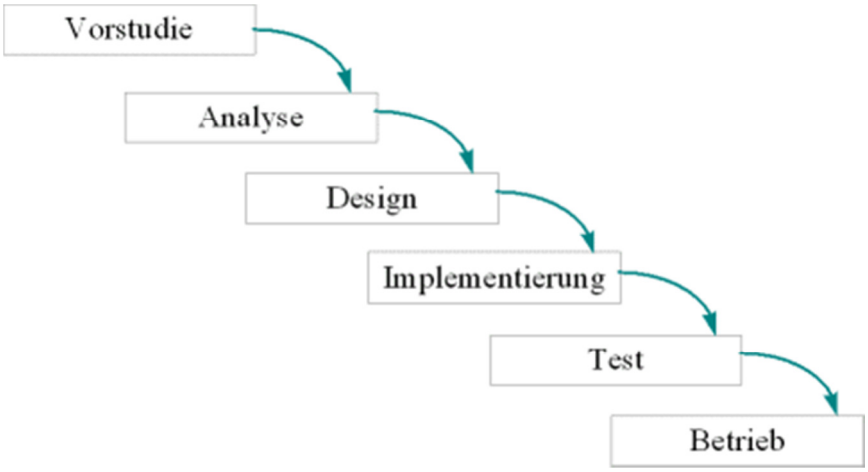

Abbildung 5: Das Wasserfallmodell[36]

Die Aufteilung des Entwicklungsprozesses in Phasen hat den Vorteil, dass der Projektmanager den Fortschritt des Projektes besser überprüfen kann. Zudem besteht die Möglichkeit, Mitarbeiter im Sinne einer Arbeitsteilung auf einzelne Tätigkeiten zu spezialisieren.[37]

---

[35] vgl. Weiss, 1998

[36] Quelle: Weiss, 1998

[37] vgl. Weiss, 1998

Als Nachteil wird häufig angeführt, dass das Modell insofern zu statisch sei, als Wiederholungsschritte nur ungenügend berücksichtigt würden. Eine nachträgliche Anpassung aufgrund wechselnder Anforderungen ziehe stets das Durchlaufen einer kompletten Phase nach sich. Überdies werde die Wartung als zeitlich größte Determinante nicht ausreichend beachtet, da es sich hierbei im Wesentlichen um ein Modell der Ersterstellung einer Software handele.[38]

Aus diesem Grund wurde das Wasserfallmodell überarbeitet und mit Rückkopplungsschritten versehen. Das erlaubt ein schrittweises „Aufwärtslaufen" der Kaskade, sofern in der aktuellen Phase etwas schieflaufen sollte, um den Fehler auf der nächsthöheren Stufe beheben zu können.[39]

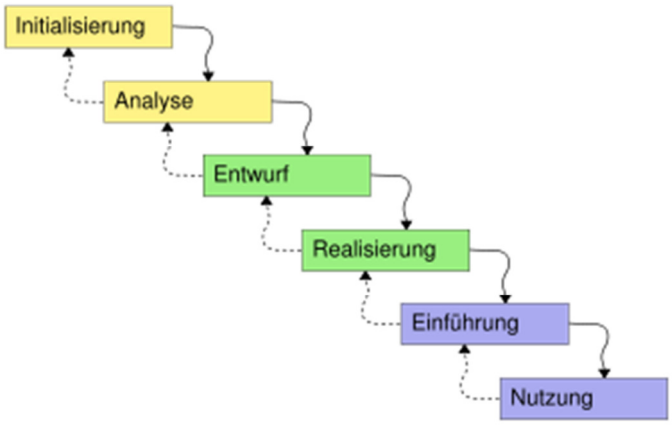

Abbildung 6: Wasserfallmodell mit Rückkopplung[40]

10.2 Prototyping

Die für diese Anforderungen besser geeigneten Systementwicklungsmodelle basieren in der Regel auf der Grundidee des Prototyping. Unter Prototyping versteht man in der Softwareentwicklung im allgemeinen die vorgezogene

[38] vgl. Weiss, 1998

[39] vgl. Academic, 2010

[40] Quelle: Academic, 2010

Entwicklung einer Teilfunktion, um deren Verhalten vor der endgültigen Realisierung unter ausgewählten Gesichtspunkten erproben zu können. Der Vorteil bei der Verwendung von Prototypen besteht darin, dass die systemtechnische Umsetzung der Nutzeranforderungen mit relativ geringem Aufwand am konkreten Beispiel veranschaulicht werden kann. Durch den geringen Erstellungs- und Anpassungsaufwand der Prototypen werden schnelle Iterationsschleifen ermöglicht und der Dialog zwischen Anwender und Systementwickler optimal unterstützt. Der Nachteil ist, dass die Prototypen in der Regel eine deutlich geringere Funktionalität aufweisen als das später realisierte Produktivsystem und so nicht alle Interaktionen gleichermaßen getestet werden können.[41]

Es kann zwischen **explorativem**, **experimentellem** und **evolutionärem** Prototyping unterschieden werden. Beim *explorativen* Prototyping wird in Zusammenarbeit mit den Benutzern versucht, aus zunächst vagen Systemvorstellungen eine möglichst vollständige Systemspezifikation für das Gesamtsystem zu erarbeiten. Anschließend erfolgt dann die gesamthafte Realisierung des Produktivsystems. Demgegenüber werden beim *experimentellen* Prototyping komplette Prototypen erstellt, um die softwaretechnische Umsetzbarkeit und Qualitätsaspekte zu überprüfen. Das *evolutionäre* Prototyping ist ein Prozess. bei dem der erste Prototyp schrittweise weiterentwickelt und verbessert wird, bis ein funktionsfähiges, den Anforderungen entsprechendes System vorliegt. Das evolutionäre Prototyping ist wie das Konzept der inkrementellen Entwicklung ein iterativer Prozess. Der Unterschied besteht darin, dass bei der inkrementellen Entwicklung zunächst alle Systemanforderungen spezifiziert sein müssen, bevor die schrittweise Umsetzung beginnt. Demgegenüber beginnt bei der evolutionären

---

[41] vgl. Wehlitz, 2001, S. 33

Vorgehensweise die Realisierung von Teilfunktionalitäten schon, bevor die Gesamtheit aller Anforderungen festgelegt ist.[42]

Eines der bekanntesten evolutionären Vorgehensmodelle ist das sogenannte Spiralmodell Bei dem der Kreislauf aus Anforderungsermittlung. Systemumsetzung, Systemtest und Systemeinführung für einzelne Teilfunktionalitäten mehrmals durchlaufen wird, bis das Gesamtsystem vollständig realisiert ist. Dieses Modell wird im nächsten Kapitel näher erläutert.[43]

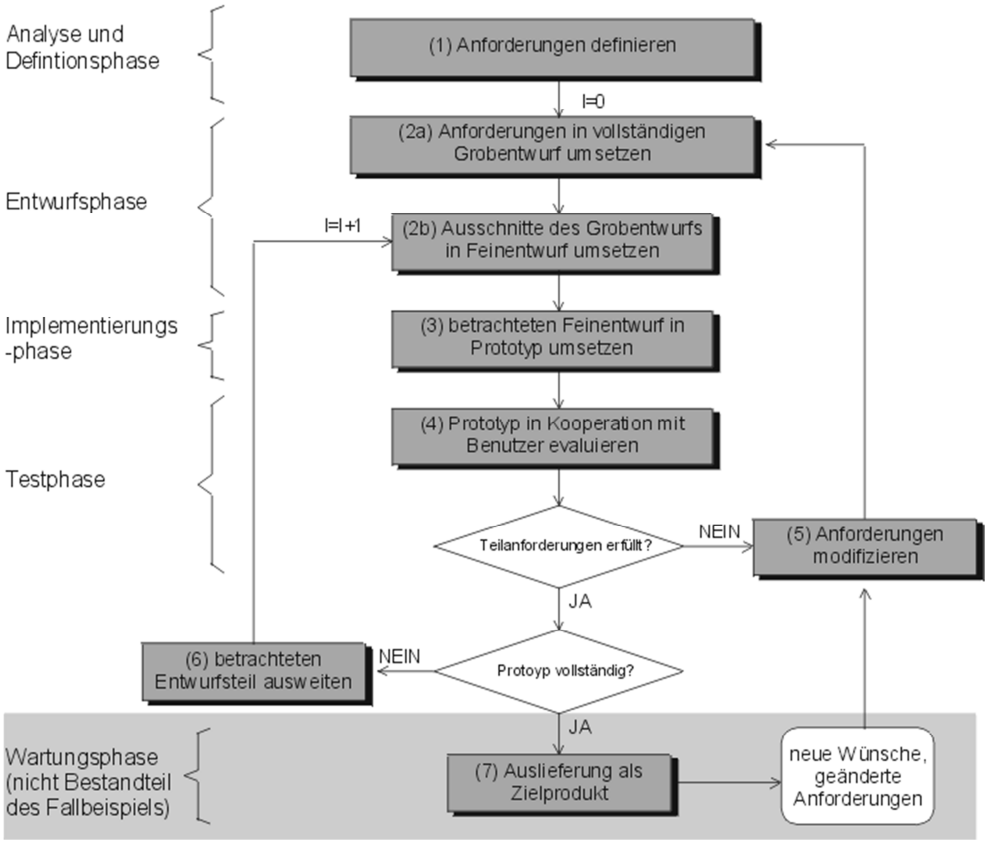

Abbildung 7: Inkrementelles Prototyping im Fallbeispiel[44]

---

[42] vgl. Wehlitz, 2001, S. 33

[43] vgl. Wehlitz, 2001, S. 34

[44] Quelle: University of Oldenburg

10.3 Spiralmodell

Beim Spiralmodell wird der gesamte Entwicklungsprozess auf einer aus vier Quadranten bestehenden Spirale abgebildet. Im ersten Quadranten werden Ziele und deren mögliche Alternativen identifiziert, die dann im zweiten Quadranten bewertet werden. Die Bewertung beinhaltet die Aufdeckung von Risikoquellen und die Definition von Gegenmaßnahmen. Der dritte Quadrant umfasst die Implementierung und die Abnahme des jeweiligen Entwicklungsschritts. Der vierte Quadrant befasst sich mit der Planung der nächsten Stufe des Software Systems. Aufgrund einer Risikoabschätzung wird in jedem Zyklus der weitere Entwicklungsverlauf abgesichert. Am Ende jedes Zyklus werden die Entwicklungsergebnisse von den betroffenen Benutzern überprüft. Erst nach Abnahme der Ergebnisse wird mit dem nächsten Zyklus begonnen.[45]

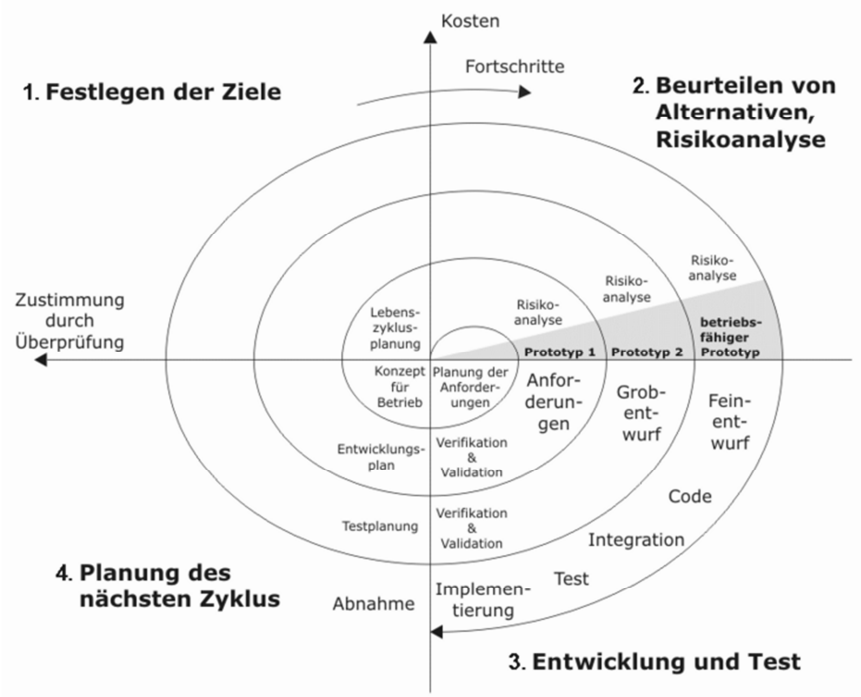

Abbildung 8: Das Spiralmodell

[45] vgl. Koord & Krauter, 2009

Die Vorteile des Spiralmodells sind:[46]

- permanentes Feedback → auf Moving Targets kann sofort reagiert werden
- pro Zyklus kann ein neuer Prozess oder Team gewählt werden
- Fehler werden relativ schnell erkannt
- bessere Eingriffsmöglichkeiten als bei einem Wasserfallmodell

Die Nachteile beim Spiralmodell sind:[47]

- besseres Management wird benötigt
- für kleine Projekte sehr aufwändig

10.4 V-Modell 97

Das V-Modell ist eine Erweiterung des Wasserfallmodells. Es gliedert den Software-Entwicklungsprozess in eine Sequenz von Phasen und integriert explizit den Qualitätssicherungsprozess, und zwar so, dass eine klare Zuordnung der Phasen des Qualitätssicherungsprozesses zu den Phasen des Entwicklungsprozesses in Form eines V vorgenommen wird. Eine Darstellung des Vorgehensmodells zur Illustration der (Haupt-)Aktivitäten, ihrer Reihenfolge, ihrer Wechselwirkungen und der erwarteten Ergebnisse zeigt die folgende Abbildung 9. Aktivitäten werden durch ein Rechteck, Ergebnisse durch ein abgerundetes Rechteck dargestellt und Pfeile geben an, welche Aktivitäten zu welchen Ergebnissen bzw. Produkten führen bzw. welche Produkte vor welcher Aktivität vorhanden sein müssen.[48]

---

[46] vgl. Keller, 2002

[47] vgl. Keller, 2002

[48] vgl. Pomberger & Pree, 2004, S. 18f

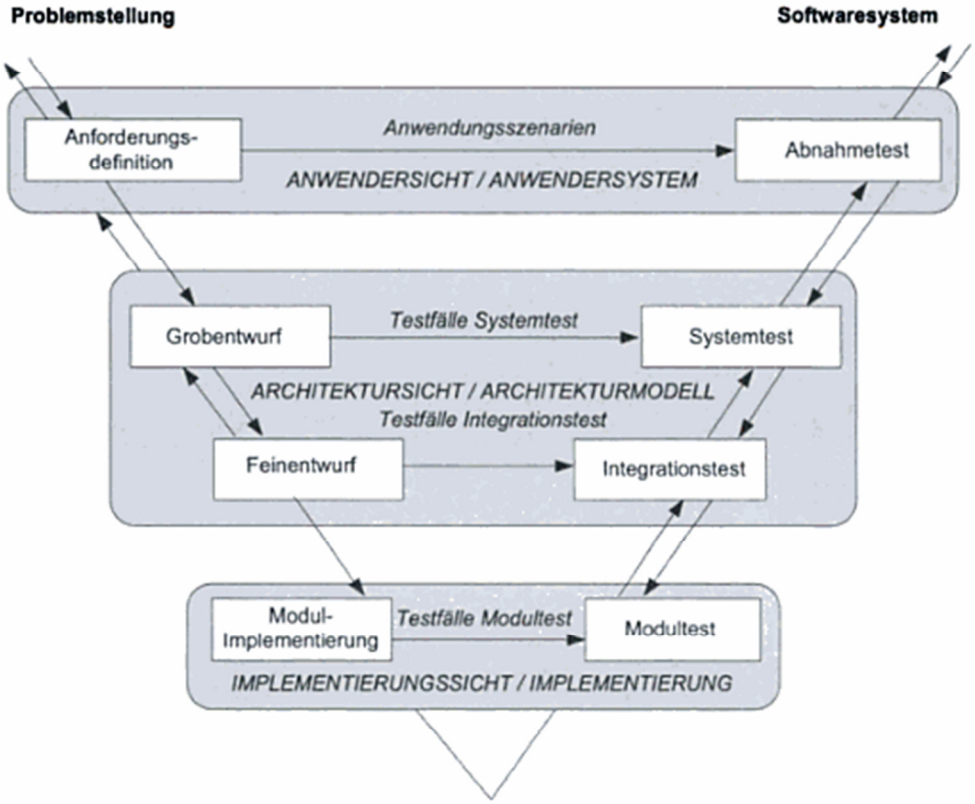

Abbildung 9: Das V-Modell[49]

Struktur des V-Modells[50]

Das Modell besteht aus 3 Ebenen:

Ebene 1: Vorgehensweise (Was ist zu tun?)

Es wird festgelegt, welche Tätigkeiten im Verlauf der Systementwicklung durchzuführen sind, welche Ergebnisse dabei zu produzieren sind und welche Inhalte diese Ergebnisse haben müssen.

---

[49] Quelle: Pomberger & Pree, 2004, S. 19

[50] vgl. Institut für Wirtschaftsinformatik Software Engineering, 2011

Ebene 2: Methodenzuordnung (Wie ist etwas zu tun?)

Es wird festgelegt, mit welchen Methoden die in Ebene 1 definierten Tätigkeiten durchgeführt werden sollen und welche Darstellungsmittel in den Ergebnissen zu verwenden sind.

Ebene 3: Werkzeuganforderungen (Womit ist etwas zu tun?)

Auf dieser Ebene wird fixiert, welche Softwaretools zur Entwicklung der Software eingesetzt werden sollen.

Auf allen Ebenen werden die Regelungen nach den Tätigkeitsbereichen, den sogenannten Submodellen, gegliedert.

Folgende Submodelle existieren:

- Projektmanagement (PM)
- Systemerstellung (SE)
- Qualitätssicherung (QS)
- Konfigurationsmanagement (KM)

Abbildung 10: Die drei Ebenen der Standardisierung[51]

[51] Quelle: Institut für Wirtschaftsinformatik Software Engineering, 2011

Die Vorteile des V-Modells sind:[52]

- Das V-Modell enthält nur die allgemeinen Regeln.
- Das V-Modell regelt die Aspekte des Softwareentwicklungsprozesses eindeutig. Diese Festlegungen können sowohl für den Auftragnehmer als auch für den Auftraggeber als Anleitung benutzt werden.
- Das V-Modell ist über unterschiedliche Detaillierungsebenen hinsichtlich der beschrieben Aktivitäten und Produkte konsistent.
- Das V-Modell ist für verschiedene Anwendungen durch „Tailoring“ nutzbar.
- Das V-Modell ist organisationsneutral und setzt keine speziellen Strukturen beim Anwender voraus.

Die Nachteile sind:[53]

- Das V-Modell beinhaltet viel Bürokratie, insbesondere die hohen Anforderungen an die Dokumentation während der Entwicklung werden bei kleinen Projekten als belastend empfunden.
- Bei der Einführung des V-Modells im Unternehmen müssen verschiedene Faktoren berücksichtigt werden. Es muss „Tailoring“ stattfinden und Mitarbeiter müssen eingeschult werden.
- Das V-Modell ist sehr schwergewichtig und daher weniger flexibel wie ein leichtgewichtiges Prozessmodell (z.B. Extreme Programming).

10.5 V-Modell XT

Das V-Modell XT ist ein empfohlenes Standardvorgehensmodell zur Planung und Durchführung von Projekten im öffentlichen Dienst.[54] Das V-Modell XT ist eine Weiterentwicklung aus dem V-Modell 97.

---

[52] vgl. Institut für Wirtschaftsinformatik Software Engineering, 2011

[53] vgl. Institut für Wirtschaftsinformatik Software Engineering, 2011

Die Grundphilosophie des V-Modells XT ist die Ziel- und Ergebnisorientierung. Die im Modell definierten Produkte stehen im Mittelpunkt und sind die zentralen Zwischen- und Endergebnisse des Projektes. Im Modell wird klar festgelegt, "Wer", "Wann", "Was" im Projekt zu tun hat. Diese detaillierte Planung dient als Basis für die Bearbeitung und Fertigstellung des Produktes, die von den umfassenden Projektdurchführungsstrategien und Entscheidungspunkten unterstützt wird und somit die grundlegende Struktur bildet. Dies ermöglicht die Vermeidung von unnötigen Tätigkeiten und die Überprüfung der Qualität des Produktes. Das Modell ist umfassend dokumentiert und frei zugänglich. Es besitzt eigene computergestützte Werkzeuge, die ebenfalls frei zugänglich sind und unter Apache-Lizenz verbreitet werden. Das Modell besteht aus aufeinander aufbauenden Vorgehensbausteinen, die zu modularen Einheiten gebildet werden. Eine konkrete Aufgabenstellung wird von einem Vorgehensbaustein komplett abgedeckt, indem Produkte, Rollen und Aktivitäten festgelegt werden. Insgesamt gibt es im V-Modell XT 21 Vorgehensbausteine.[55]

Folgende vier Bausteine bilden den Kern:[56]

- Projektmanagement
- Qualitätssicherung
- Konfigurationsmanagement
- Problem- und Änderungsmanagement

Die Kern-Bausteine können je nach Projekttyp durch optionale Vorgehensbausteine erweitert werden. So wird durch die Wahl eines der vier Projekttypen die Organisation und Durchführung von Anfang an festgelegt.[57]

---

[54] vgl. Koord & Krauter, 2009

[55] vgl. Koord & Krauter, 2009

[56] vgl. Koord & Krauter, 2009

[57] vgl. Koord & Krauter, 2009

Es stehen folgende Projekttypen zur Auswahl:[58]

- Systementwicklungsprojekt (AG)
- Systementwicklungsprojekt (AN)
- Systementwicklungsprojekt (AG/AN)
- Einführung und Pflege eines organisatorischen Vorgehensmodells

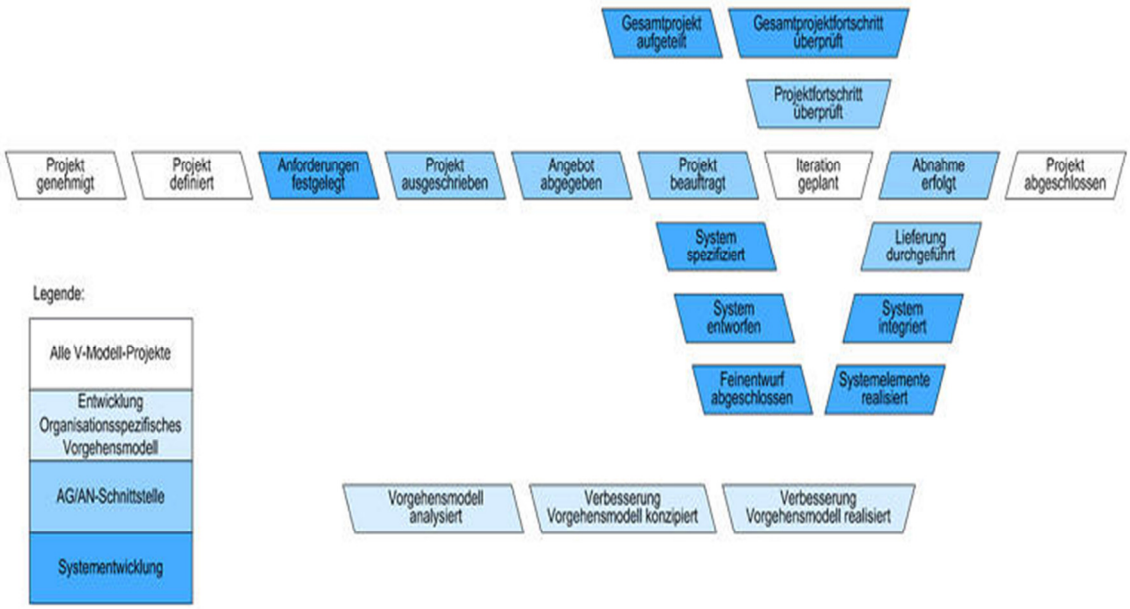

Abbildung 11: Gesamtstruktur des V-Modell XT[59]

10.6 Iterative Vorgehensmodelle

Iterative Vorgehensmodelle sind durch einen mehrfachen Durchlauf von Prozessphasen gekennzeichnet. Die Erstellung des Endprodukts erfolgt also nicht in einem Durchlauf des Vorgehensmodells, sondern in mehreren. Mit jedem Durchlauf wird die Software umfangreicher und nähert sich dem gewünschten

[58] vgl. Koord & Krauter, 2009

[59] Quelle: Informatik Forum Simon GmbH, 2009

Endprodukt an. Iterative Vorgehensmodelle lassen sich unterscheiden in evolutionäre und inkrementelle Modelle.[60]

Ein prominenter Vertreter aus dieser Reihe ist das Modell von IBM „Rational Unified Process (RUP)".

Evolutionäre Modelle

Ausgangspunkt bei evolutionären Modellen sind Anforderungen bzw. Aufträge eines Kunden, für den ein Softwareprodukt erstellt werden soll. Dabei wird zwischen Muss- und Kann-Anforderungen unterschieden. Die Muss-Anforderungen definieren die minimalen Funktionsanforderungen an die Software, die Kann-Anforderungen die darüber hinausgehenden wünschenswerten Funktionen. In einem evolutionären Modell wird zur Erfüllung dieser Anforderungen davon ausgegangen, dass die Software in mehreren Produkt Versionen entwickelt wird. Für die Erstellung der ersten Version werden nur die Muss-Anforderungen herangezogen, um die wesentlichen Funktionalitäten bereitzustellen. Mit dieser Grundversion können praktische Erfahrungen gesammelt werden, während bereits an der nächsten Softwareversion mit der Erweiterung der Funktionalität entsprechend ausgewählter Kann-Anforderungen gearbeitet wird. Die gesammelten Erfahrungen fließen dabei in die Entwicklung ein. So wird die Software stufenweise weiterentwickelt, bis das vollständige Endprodukt vorliegt.[61]

Evolutionäre Modelle sind insbesondere für Situationen geeignet, in denen der Auftraggeber zu Beginn noch keine vollständige Definition der gesamten Anforderungen vornehmen kann. Durch die Möglichkeit der Sammlung von Erfahrungen mit ersten Grund Versionen können diese Anforderungen später konkretisiert werden. Das Ergebnis lässt sich so auf die sich entwickelnden Bedürfnisse abstimmen. Fehlentwicklungen einzelner Versionen lassen sich durch

---

[60] vgl. Freund, 2007, S. 53

[61] vgl. Freund, 2007, S. 53f

den beschränkten Umfang der Entwicklungsstufen leichter korrigieren als bei einer kompletten Entwicklung in einem großen Schritt. Die ersten lauffähigen Versionen sind darüber hinaus schneller einsatzbereit, als es bei einem vollständigen System der Fall wäre. Die Risiken des evolutionären Vorgehens sind umfangreiche Überarbeitungen der Systemarchitektur, die aufgrund von Fehlern in der Definition der ursprünglichen Muss-Anforderungen notwendig werden. Außerdem kann die erste Version durch den begrenzten Umfang so eingeschränkt in ihrer Flexibilität sein, dass eine Anpassung an spätere Änderungen nicht möglich und eine erneute Entwicklung notwendig ist.[62]

Vorteile des evolutionären Vorgehensmodells:[63]

- Die einsatzfähigen Produkte werden dem Auftraggeber regelmäßig vorgestellt.
- Die Anzahl der kleinen Arbeitsschritte ermöglicht die Überschaubarkeit des Produktes und die Bestimmung der Entwicklungsrichtung.
- Erfahrungen aus vorherigen Produktversionen werden berücksichtigt.
- Zwischenergebnisse ermöglichen eine frühere Erkennung der Termineinhaltung.

Nachteile des evolutionären Vorgehensmodells:[64]

- Es besteht die Gefahr, dass bei einer fehlerhaften Entwicklung der Kerneigenschaften die komplette Arbeit wieder von Anfang gestartet wird.
- Außerdem besteht die Gefahr, dass die Nullversion in der Entwicklung nicht so flexibel ist, wie es erwartet wird.

---

[62] vgl. Freund, 2007, S. 54f

[63] vgl. Koord & Krauter, 2009

[64] vgl. Koord & Krauter, 2009

Inkrementelle Modelle

Die Nachteile der evolutionären Modelle werden bei inkrementellen Modellen vermieden, indem die Anforderungen vor der Entwicklung möglichst vollständig erfasst werden. Aus diesen Anforderungen werden die wichtigsten ausgewählt und ähnlich dem evolutionären Vorgehen zunächst eine erste Version mit eingeschränkter Funktionalität entwickelt. Die Erfahrungen des Auftraggebers mit den ersten Versionen werden in weiteren Entwicklungsschritten eingearbeitet und die Software bis zur vollständigen Version weiterentwickelt.[65]

Durch dieses Vorgehen bleibt der Vorteil der schnellen Erstellung einer lauffähigen Version erhalten. Die vollständige Erfassung der Anforderungen vor Beginn der Entwicklung ermöglicht, dass die entworfene Systemarchitektur mit ihren Schnittstellen für später folgende geplante Erweiterungen angepasst entworfen werden kann. Dadurch lassen sich spätere umfangreiche Änderungen der ersten Version vermeiden. Einen Nachteil der inkrementellen Modelle stellt der zusätzliche Aufwand für die Erfassung und Bearbeitung der vollständigen Anforderungen vor Beginn der Entwicklung dar. Außerdem besteht durch die frühe Anforderungsermittlung das Risiko, dass sich wesentliche Anforderungen nach den ersten Versionen ändern bzw. anfangs falsch eingeschätzt wurden und aus diesem Grund umfangreiche Anpassungen notwendig werden, was durch das Vorgehensmodell gerade vermieden werden sollte.[66]

---

[65] vgl. Freund, 2007, S. 54f

[66] vgl. Freund, 2007, S. 55

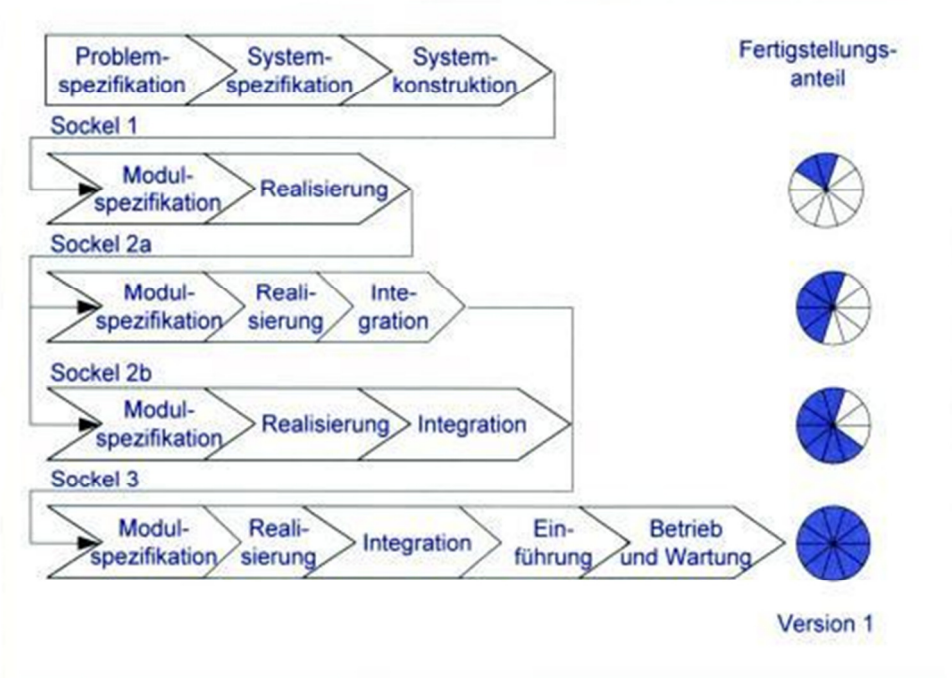

Abbildung 12: Inkrementelles Modell[67]

10.6.1 Rational Unified Process (RUP)

Der Rational Unified Process (RUP) ist ein objektorientiertes Vorgehensmodell zur Softwareentwicklung. Erdacht wurde es von der Firma Rational Software, die seit 2002 Teil des IBM Konzerns ist. Der Unified Process wurde ursprünglich als Metamodell für andere Entwicklungsmodelle parallel zur Unified Modelling Language (UML) entwickelt, welche der späteren konkreten Prozessimplementierung (RUP) dann auch als Notationssprache diente. Als objektorientierte Methode basiert der Rational Unified Process dabei auf Prinzipien wie Use Cases oder inkrementellem und iterativem Vorgehen.[68]

---

[67] Quelle: Koord & Krauter, 2009

[68] vgl. Creative Acts GmbH, 2009

Die Aspekte werden in statische und dynamische unterteilt.

Statische Aspekte

Der RUP legt dabei für die Entwicklung die folgenden grundlegenden Arbeitsschritte fest:[69]

- Kernarbeitsschritte
  - Business Modelling (Geschäftsprozessmodellierung)
  - Requirements (Anforderungsanalyse)
  - Analysis & Design
  - Implementation
  - Testing
  - Operation & Support
  - Deployment (Auslieferung)

- Unterstützende Arbeitsschritte
  - Configuration & Change Management (Konfigurations- und Änderungsmanagement)
  - Projektmanagement
  - Environment (Infrastruktur)

[69] vgl. Creative Acts GmbH, 2009

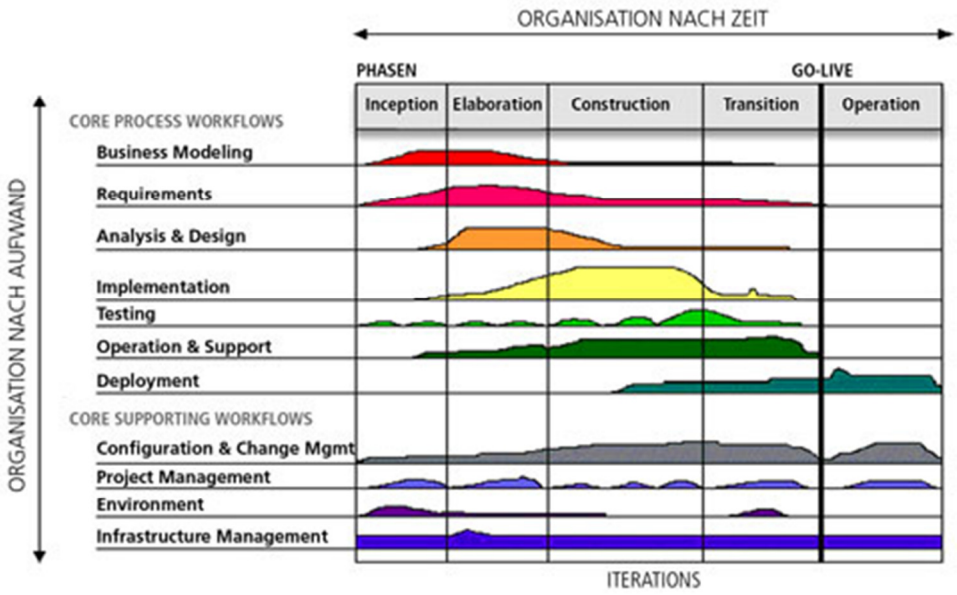

Abbildung 13: Arbeitsschritte beim RUP[70]

Dynamische Aspekte

Orthogonal dazu gibt es im RUP vier Phasen, in welchen jeder der Arbeitsschritte mehr oder weniger intensiv zur Anwendung kommt:[71]

- Inception Phase (Konzeptionsphase)
- Elaboration Phase (Entwurfsphase)
- Construction Phase (Konstruktionsphase)
- Transition Phase (Übergabephase)

Resultate der Phasen sind die folgenden Meilensteine:[72]

- Inception: Lifecycle objectives milestone (Vision inklusive rudimentäres Anwendungsfallmodell, provisorische Architektur, Identifikation der wesentlichsten Risiken, Planung der Ausarbeitungsphase)

---

[70] Quelle: Creative Acts GmbH, 2009

[71] vgl. Creative Acts GmbH, 2009

[72] vgl. Creative Acts GmbH, 2009

- Elaboration: Lifecycle architecture milestone (Architekturprototyp, detailliertes Anwendungsfallmodell, Planung der Konstruktionsphase)
- Construction: Initial operational capability milestone (Entwurfsmodelle und Beta-Release der Software)
- Transition: Product release milestone (Release in Produktionsqualität)

Diese Phasen sind in Iterationen unterteilt. Nach Abschluss jeder Phase wird kontrolliert, ob der zuvor definierte Meilenstein erreicht wurde – falls nicht, kann ein weiterer Durchgang der Phase (Iteration) angehängt werden, was oft bei der Construction Phase großer Projekte eintreten kann.[73]

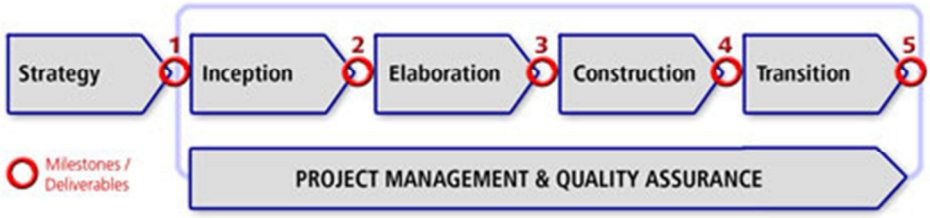

Abbildung 14: Phasen und Meilensteine beim RUP[74]

10.7 Agile Vorgehensmodelle

Agile Vorgehensmodelle benötigen auf Grund ihres inkrementell-iterativen Aufbaus grundsätzlich ein komplexeres Projektmanagement. Sie haben jedoch den Vorteil, dass bei Beginn der Durchführungsphasen nicht alle Details bekannt sein müssen. Daher bietet sich dieses Vorgehen insbesondere für unübersichtlichere Projekte mit einer hohen Änderungswahrscheinlichkeit während der Realisierung an.[75]

---

[73] vgl. Creative Acts GmbH, 2009

[74] Quelle: Creative Acts GmbH, 2009

[75] vgl. Nielsen, 2008, S. 37

Die Grundsätze für agile Vorgehensmodelle sind:[76]

- Gut qualifizierte Personen und eine effiziente Zusammenarbeit sind wichtiger als Prozessdefinitionen und Werkzeuge.
- Eine gut funktionierende, für den Anwender passende Software ist bedeutend mehr wert als eine ausführliche Dokumentation.
- Eine stetige, aktive Zusammenarbeit mit den Endanwendern ist gegenüber einem ausführlichen Vertragswerk vorzuziehen.
- Mut und Offenheit für notwendige Änderungen sind wichtiger als das sture Befolgen eines festgelegten Plans.

Der gemeinsame Nenner aller agilen Vorgehensmodelle ist **Iteration**, dh. es gibt keine herkömmlichen Projektphasen, sondern nur relativ kurze Iterationen. Die Dauer der Iterationen kann frei gewählt werden und liegt zwischen täglich und quartalsmäßig. Am Ende jeder Iteration steht ein für den Kunden greifbares Resultat, dh. eine lauffähige Software. Der Umgang mit den Anforderungen ist in leichtgewichtigen Modellen grundsätzlich ein anderer als in schwergewichtigen Modellen. Es gibt keinen Redaktionsschluss für Veränderungen in den Anforderungen, die zu Change Requests (CR) führen. Bei jeder neuen Iteration suchten die Projektmitarbeiter die Anforderungen und entscheiden neu, welche davon in dieser Iteration realisiert werden sollen.[77]

10.7.1 Extreme Programming (XP)

XP wurde von Kent Beck in seinem Manifest veröffentlicht und ist eines der ältesten agilen Vorgehensmodelle. Hier teilt der Anwendervertreter seine Anforderungen an das neue System mittels User Stories mit. Dh., er schreibt in seiner Sprache auf, was er vom neuen System bezüglich der Lösung seines Problems erwartet. Diese bilden die Grundlage für die Planung der nächsten

[76] vgl. Tremp & Ruggiero, 2001, S. 16

[77] vgl. Tremp & Ruggiero, 2001, S. 16

Iteration, die als Ziel die Lieferung eines neuen Releases der Software hat, in welcher die Funktionalität für die zur Realisierung ausgewählten User Stories umgesetzt ist. Bei technischen Unsicherheiten entwickeln die Programmierer Prototypen, um die Machbarkeit und den Aufwand besser abschätzen zu können.[78]

Weiters beschreiben die Entwickler zusammen mit dem Anwendervertreter eine für die Freigabe verbindliche und vollständige Reihe von Testszenarien für den Abnahmetest. Damit wird das gegenseitige Verständnis über die zu implementierende Funktionalität gefördert. Entwickelte Features werden erst dann für die Produktion freigegeben, wenn die Anwender alle dazugehörigen Testszenarien positiv testen konnten.[79]

Abbildung 15: Vorgang in XP[80]

Für das Gelingen von Softwareentwicklungsprojekten liegt der Schwerpunkt bei XP nicht so sehr auf dem Prozessmodell, sondern in der offenen und konstruktiven Zusammenarbeit im Team. Der entwickelte Code gehört immer dem ganzen Team. Die Programmierung selbst soll immer im Zweier-Team erfolgen. Überstunden soll es grundsätzlich nicht geben. Was in einer Iteration nicht erledigt werden kann, wird für eine nächste Iteration geplant. Täglich findet ein Meeting im Stehen statt, in dem sich das gesamte Entwicklungsteam kurz und offen austauscht Das

---

[78] vgl. Tremp & Ruggiero, 2001, S. 17

[79] vgl. Tremp & Ruggiero, 2001, S. 17

[80] Quelle: Tremp & Ruggiero, 2001, S. 17

Meeting wird stehend durchgeführt, damit es möglichst effizient bleibt. Während der ganzen Entwicklungszeit muss der Anwendervertreter immer mit dabei sein.[81]

10.7.2 Scrum

Im Mittelpunkt von Scrum steht das selbst organisierte Entwicklerteam, das ohne Projektleiter auskommt. Um dem Team eine störungsfreie Arbeit zu ermöglichen, sorgt der Scrum Master als Methodenfachmann dafür, dass der Entwicklungsprozess reibungslos ablauft. Der Scrum Master ist auch die Schnittstelle zum Product Owner, der die Anforderungen im Product Backlog spezifiziert. Bei Scrum heißen die Iterationen Sprint. Bei der Planung eines Sprints priorisiert der Product Owner die Anforderungen im Backlog und das Entwicklerteam verpflichtet sich zu einer Reihe von Features, die es in dieser Zeit umsetzen kann. Während eines Sprints darf der Product Owner das Entwicklerteam nicht mit geänderten oder neuen Anforderungen stören. Das Entwicklerteam trifft sich täglich zum Daily Scrum, bei dem jedes Teammitglied folgende Fragen beantwortet:[82]

- Was habe ich seit dem letzten Daily Scrum getan?
- Was hat mich dabei gestört?
- Was werde ich bis zum nächsten Daily Scrum tun?[83]

Am Ende eines Sprints stellt das Entwicklerteam dem Product Owner die neue Funktionalität bzw. die neue Softwareversion vor. Wird diese akzeptiert, kann sie produktiv gesetzt werden (als Software Release). Während einer Sprint-

---

[81] vgl. Tremp & Ruggiero, 2001, S. 18

[82] vgl. Tremp & Ruggiero, 2001, S. 18

[83] vgl. Tremp & Ruggiero, 2001, S. 18

Retrospektive reflektiert das Team den Sprintprozess, um daraus Lehren für den nächsten Sprint zu ziehen.[84]

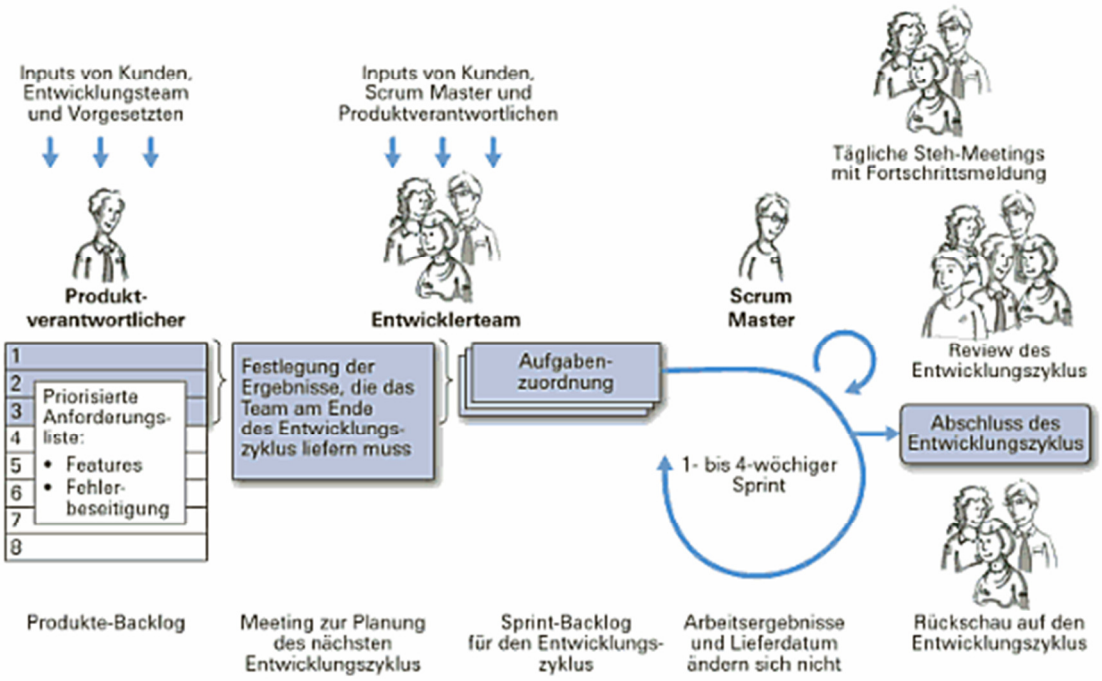

Abbildung 16: Scrum – Prozess und Grundsätze[85]

[84] vgl. Tremp & Ruggiero, 2001, S. 18

[85] Quelle: Tremp & Ruggiero, 2001, S. 19

11. Probleme bei Lifecycle-Modellen

Probleme können durch den Anwender, aber auch durch die fehlende Unterstützung von Experten entstehen. Probleme können wie folgt sein:

durch den Anwender:[86]

- Vorgehensmodell als Vorschrift
- Kreativitätseinschränkung, Demotivation, Bürokratismus
- Vertragsgrundlage (z.B. ISO-Normen)
- aus sachlichen Gründen, z.B. zu geringe Effizienz
- Einführungsaufwand

fehlende nötige Unterstützungsressourcen:[87]

- Kontrollorgane zur Durchsetzung (Personalressourcen)
- Hotline für Nachfragen (Personalressourcen)
- Qualitätssicherung der Vorgehensmodelle

[86] vgl. Apitz & Rasch, 2011, S. 39

[87] vgl. Apitz & Rasch, 2011, S. 39

12. Lifecycle-Modelle im Vergleich

Fragestellung	Wasserfall-modell	Spiral-modell	V-Modell	RUP	XP
Seit wann gibt es das Vorgehens-modell? Wer hat es entwickelt?	*1970, Winston Royce*	*1986, Barry W. Boehm*	*1992, deutsches Bundes-ministerium für Verteidigung*	*1999, Rational Software*	*2000, Kent Beck, Ward Cunningham und Ron Jeffries*
Wie groß ist der Aufwand das Vorgehensmodell einzusetzen?	*klein*	*mittel*	*groß*	*groß*	*klein/ mittel*
Wo liegt der Fokus des Vorgehensmodells?	*methodische Vorgehensweise*	*Ziele, Rahmen-bedingungen und Lösungen*	*Unter-nehmens-prozess*	*Projekt-prozess*	*Ent-wicklungs-prozess*
Wie ist die Bandbreite des VMs? Zu welchem Zeitpunkt eines Projekts soll das VM zum Einsatz kommen?	*Analyse, Design, Implementierung, Test, Einsatz*	*Zielfestlegung, Risikoanalyse, Entwicklung und Test, Planung*	*Anforderungsspezifikation, Analyse, Design, Implementierung, Rollout, Wartung*	*Geschäftsprozessmodellierung, Anforderungsspezifikation, Analyse, Design, Implementierung, Rollout, Wartung*	*Analyse, Design, Implementierung*
Wie ist das Vorgehensmodell verfügbar (frei, verfügbar, kommerziell, unternehmensspezifisch)?	*Frei verfügbar*	*Frei verfügbar*	*Frei verfügbar*	*Frei verfügbar, jedoch kommerzielle Tools*	*Frei verfügbar*
Für welche Projektarbeiten/Branchen ist das VM geeignet?	*Keine Projekt- od. Branchenbeschränkung*	*Keine Projekt- od. Branchenbeschränkung*	*Keine Projekt- od. Branchenbeschränkung*	*Keine Projekt- od. Branchenbeschränkung*	*Keine Projekt- od. Branchenbeschränkung*
Für welche Projektgrößen ist das VM geeignet	*Klein*	*Mittel*	*Mittel/Groß*	*Mittel/Groß*	*Klein/Mittel*

(Kriterien wie Größe, Laufzeit, Budget)?					
Sind projektspezifische Anpassungen des VMs möglich?	*Nein, da Vorgangsweise vorgegeben ist.*	*Ja, Anpassung ist möglich.*	*Ja, Anpassung ist explizit berücksichtigt.*	*Ja, Anpassung ist explizit berücksichtigt.*	*Ja, Anpassung ist explizit berücksichtigt.*
Wird das Projektmanagement aktiv durch das VM unterstützt? Ist Projektmanagement eine Komponente des VMs?	*Nein, Projektmanagement ist keine Komponente.*	*Nein, Projektmanagement ist keine Komponente.*	*Ja, Projektmanagement ist ausführlicher Bestandteil des VMs.*	*Ja, Projektmanagement ist ausführlicher Bestandteil des VMs*	*Nein, wenig Aussagen enthalten.*
Wird der Kunde in den Projektablauf einbezogen?	*Ja, Kunde wird am Ende einbezogen.*	*Ja, Kunde wird mit einbezogen.*	*Ja, Kunde wird aktiv einbezogen.*	*Ja, Kunde kann durch Rollenzuweisung einbezogen werden.*	*Ja, Kunde wird aktiv einbezogen.*
Welche Applikationsarten/Einsatzgebiete/Arten von Entwicklungsvorhaben werden durch das VM unterstützt?	*alle Paradigmen*	*alle Paradigmen*	*alle Paradigmen*	*Objektorientierte Einsatzgebiete*	*Objektorientierte Einsatzgebiete*
Gibt es eine Qualitätssicherung/ ein Qualitätsmanagement innerhalb des VMs?	*Nur über Resultate nach den Phasen.*	*Nur über Resultate nach den Phasen.*	*Extra Prozess/Phase*	*Extra Prozess/Phase*	*Integriert in sämtliche Phasen*
Gibt es ein Testmanagement innerhalb des VMs?	*Extra Phase*	*Integriert im Zyklus*	*Extra Prozess/Phase*	*Extra Prozess/Phase*	*Integriert in sämtliche Phasen*
Wird das VM durch Tools unterstützt?	*Ja*	*Ja*	*Ja*	*Ja*	*Ja*

Tabelle 1: Vergleich der Vorgehensmodelle[88]

[88] vgl. Hähne, 2007, S. 31f

13. Entwicklung eines Lifecycle-Modells zur Data-Warehouse-Entwicklung

In Kombination zu meiner derzeit ausgeübten Tätigkeit als Controller und Data-Warehouse-Entwickler, stelle ich hier das Vorgehensmodell, welches beruflich entwickelt und erfolgreich in der Controlling Abteilung am eigens entwickelten Data Warehouse angewandt wurde, vor. Ziel unserer Datenbank und Softwarelösung war es, alle Daten des Unternehmens in einem System physisch abzubilden. Einerseits werden Daten direkt in der Datenbank abgelegt, andererseits gibt es Live-Sichten auf Daten in anderen Systemen, welche durch Verknüpfungen von Tabellen oder Views sichtbar sind.

Das Vorgehensmodell eines Data Warehouse unterscheidet sich durch die Verschiedenheit der Ziele und Datenstrukturen von dem Vorgehensmodell der klassischen Softwareentwicklung. Die klassische Systementwicklung startet mit Anforderungen und endet mit den gewünschten Daten (anforderungsorientiert). Die Data-Warehouse-Entwicklung hingegen ist genau gegensätzlich, es wird mit Daten begonnen und endet mit Benutzeranforderungen (datenorientiert).[89]

Das Data-Warehouse bildet auch keine operationalen Geschäftsprozesse ab, was eine nachfolgende Prozessmodellierung unnötig macht und Entwicklungsstufen entfallen lässt.[90]

Die genannten Unterschiede werden in einem traditionellen Vorgehensmodell nur unzureichend berücksichtigt.[91]

---

[89] vgl. Inmon, 1996, S. 24f

[90] vgl. Poe & Reeves, 1997, S. 80

[91] vgl. Institut für Wirtschaftsinformatik, 2011

13.1.1 Phasen

Die Vorgehensweise die hier gewählt wurde, ist eine Kombination aus dem Spiralmodell und einem agilen Vorgehensmodell. Grundsätzlich kann gesagt werden, dass es drei Phasen bei der Entwicklung einer Data-Warehouse-Lösung gibt.

Erste Phase: *Analyse*

Bezogen auf die Qualität, die Granularität und die zukünftigen Verwendbarkeit der Daten wurden hier die relevanten operativen aber auch die systemtechnischen Daten bestimmt. Ein besonderes Augenmerk gilt hier den Schnittstellen zu den Vorsystemen. Diese unternehmensinternen und –externen Datenquellen sind gründlich zu analysieren und zu definieren, damit zukünftig eine eventuelle weitere Anbindung möglich ist. Somit ist für eine qualitativ ausgezeichnete Datenübernahme und Datentransfer gesorgt.

Zweite Phase: *Design*

Nach erfolgreicher Analyse der Daten ist der nächste Schritt die tatsächliche Entwicklung der Data-Warehouse-Lösung. Der erste Schritt in dieser Phase ist die Datenmodellierung. Anhand dieses visuellen Hilfsmittels werden die entsprechenden Tabellen mit deren Eigenschaften angelegt. Besonders auf die zukünftige Erweiterbarkeit ist zu achten. Weiters ist die Granularität der Daten ein relevanter Faktor bei der Datenmodellierung.

Dritte Phase: *Implementierung*

Hier findet der Übergang von der systemtechnischen Ebene zur Anwenderebene statt. Durch entsprechende Abfragemöglichkeiten und Berichtsarten wird dem Anwender eine freie Möglichkeit zur Datenanalyse und Datenauswertung gegeben. Hier gilt es auch das System in Bezug auf Zugriffsrechte, Performance und Benutzeranforderungen zu überwachen und zu verbessern.

13.1.2 Ablauf

Da nicht alle Vorsysteme zeitgleich analysiert werden können, wird ein System nach dem anderen analysiert und im Data Warehouse integriert. Somit ergibt sich eine Vorgangsweise wie bei einem Spiralmodell. Auch durch die Weiterentwicklung und Bearbeitung der bereits integrierten Daten ergibt sich eine immer wiederkehrende Prozedur, die dem Spiralmodell ähnelt.

Der Übergang zwischen den Phasen ist starr. Erst wenn die Analyse abgeschlossen ist kann mit dem Design begonnen werden. Nach Abschluss der Phase „Design“ ist die Implementierung möglich. Jedoch ist wie beim Spiralmodell eine Wiederaufnahme eines bereits implementierten Moduls möglich und der Durchlauf durch die einzelnen Phasen kann ohne Qualitätsverlust fortgesetzt werden. Weiters lässt es eine Integration eines neuen Moduls zu ohne eines der bereits integrierten Module überarbeiten zu müssen. Somit ist dieses Vorgehensmodell durch Flexibilität gekennzeichnet.

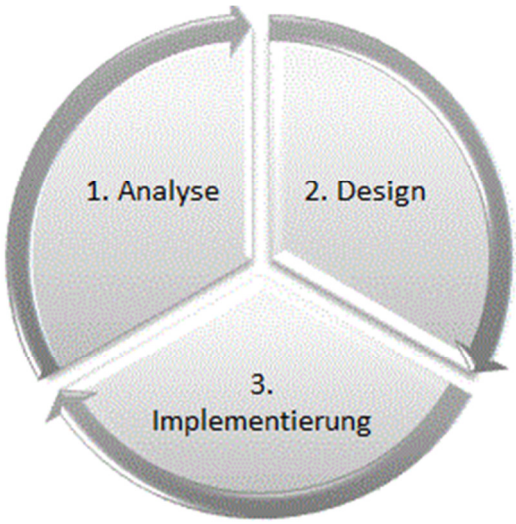

Abbildung 17: Phasen des VM bei einer Data-Warehouse-Entwicklung

Zu Beginn wird täglich durch ein 5-minütiges Meeting eine Bestandsaufnahme getätigt. Alle Beteiligten geben ihre Erfahrungen und Anforderungen weiter. Durch kurze intensive Diskussionen werden die weiteren Vorgangsweisen ausgearbeitet und in den darauffolgenden Tagen umgesetzt.

Zusammengefasst ergibt dies einen ständigen Durchlauf durch die bereits oben erwähnten Phasen und periodisierte Absprachen mit den beteiligten Personen.

13.1.3 Data-Warehouse-Schichten

Durch die Möglichkeit Daten verschiedener Granularitäten zu speichern, ist die Entwicklung einer Data-Warehouse-Lösung in mehreren Schichten zu überlegen. Die einzelnen Schichten sollten in separaten Datenbanken abgebildet werden. Keine Schicht sollte mit einen weiteren Schicht in einer Datenbank zusammengefasst werden. Das führt unter Umständen zu Komplikationen bei der Datenanalyse. Über festgelegte periodische Abläufe werden die Daten der verschiedenen Schichten aktualisiert.

Die erste Schicht sollte die Rohdaten beinhalten. Dh. die Daten sind auf unterster Granularität abgespeichert. In weiteren Schichten kann die Granularität der Daten modifiziert werden, sodass zum Beispiel die Daten nicht mehr auf unterster Ebene zur Verfügung stehen, sondern als summierter Wert. Durch diese Varianten mit Schichten können, wenn es der Bedarf erfordert, mehrere Aggregationsebenen der selben Daten gespeichert und weiter verarbeitet werden. Dies ergibt vor allem einen Performancegewinn bei der Berichterstellung und den Abfragen, da die Daten nicht jedes Mal neu aggregiert werden müssen.

14. Schluss

In dieser Arbeit wurden die verschiedenen Vorgehensmodelle kurz und bündig vorgestellt, sowie die einzelnen Phasen und Eigenschaften miteinander verglichen. Die Vor –und Nachteile wurden herausgearbeitet, damit ersichtlich ist welches Modell für welche Softwareart geeignet ist. Dies spielt für Programmierer und Softwareprojektmanager oftmals eine relevante Rolle bei der Entwicklung und Einführung der Software.

Es wurde der Begriff „Lifecycle-Modell“ detailliert erklärt, sowie einzelne Eigenschaften aus diesem Gebiet näher vorgestellt. Das beinhaltet unter anderem die Begriffe „Sichten auf Vorgehensmodelle“, „Anforderungen“, „Tätigkeitsbereiche“ oder auch „Arten von Lifecycle-Modellen“. In diesem letzten Punkt erfolgte eine Auflistung der verschiedensten Modelle und eine detaillierte Beschreibung eines bekannten Vertreters dieser Art.

Der letzte Abschnitt dieser Arbeit befasst sich mit einem eigen erstellten Vorgehensmodell, das sich hauptsächlich aus den positiven Eigenschaften der bereits erwähnten Modelle zusammensetzt und speziell auf die Entwicklung von Data-Warehouse-Systemen abgestimmt ist.

Für die Zukunft stellt sich die Frage, wie sich die Weiterentwicklung der Softwarearten auf die einzelnen Vorgehensmodelle auswirken wird, welche davon in Zukunft keine bedeutende Rolle mehr spielen und welche durch Modifikationen in Verwendung bleiben.

Literaturverzeichnis

Academic. (2010). *Academic*. Retrieved April 4, 2011, from Academic: http://de.academic.ru/dic.nsf/dewiki/1490895

Apitz, M., & Rasch, P. (2011, Februar 22). *Universität Leipzig.* Retrieved April 3, 2011, from Universität Leipzig: http://bis2.informatik.uni-leipzig.de/studium/seminare/2004_ss/ise/2-Vorgehensmodelle.pdf

Biskup, H., & Fischer, T. (2005, Mai 2). Vorgehensmodelle - Versuch einer begrifflichen Einordnung. München, Bayern, Deutschland.

Creative Acts GmbH. (2009). *Creative Acts GmbH*. Retrieved April 5, 2011, from Creative Acts GmbH: http://www.creativeacts.com/en/node/29

Fettke, P., Intorsureanu, I., & Loos, P. (2011). Komponentenorientierte Vorgehensmodelle im Vergleich. Chemnitz, Deutschland. Retrieved April 5, 2011, from Universität Augsburg - Wirtschaftsinformatik System Engineering: http://www.wi2.info/downl/gi-files/WKBA4/p05.pdf

Freund, T. (2007). *Software Engineering durch Modellierung wissensintensiver Entwicklungsprozesse.* Berlin: GITO Verlag.

Hähne, T. (2007). *Vorgehensmodelle zur Einführung von Individual- und Standardsoftware: Bewertung der Praxisrelevanz der Accenture Delivery Methods (ADM).* Hannover: Universität Hannover.

Helmke, H., Höppner, F., & Isernhagen, R. (2007). *Einführung in die Sofwareentwicklung - Vom programmieren zur erfolgreichen Software-Projektarbeit.* München, Wien: Carl Hanser Verlag.

Hoffmann, D. (2010). *Data Warehouse im Rahmen der Business Intelligence.* Hamburg: Diplomica Verlag GmbH.

Informatik Forum Simon GmbH. (2009, September 20). *Informatik Forum Simon GmbH*. Retrieved April 30, 2011, from Informatik Forum Simon GmbH: http://www.infforum.de/themen/anwendungsentwicklung/grafik_SE_v-modell_XT.htm

Informatik Forum Simon GmbH. (2011, April 1). *Informatik Forum Simon GmbH*. Retrieved April 6, 2011, from Informatik Forum Simon GmbH: http://www.infforum.de/themen/anwendungsentwicklung/thema_SE-vorgehensmodelle.htm

Inmon, W. (1996). *Building the Data Warehouse. 2nd Edition.* New York: Wiley Computer Publishing.

Institut für Wirtschaftsinformatik. (2011). *Institut für Wirtschaftsinformatik*. Retrieved Mai 1, 2011, from Institut für Wirtschaftsinformatik: http://www.iwi.uni-hannover.de/lv/seminar_ws03_04/www/Sandner/webseite/41.html#62

Institut für Wirtschaftsinformatik Software Engineering. (2011). *Institut für Wirtschaftsinformatik Software Engineering*. Retrieved April 4, 2011, from Institut für Wirtschaftsinformatik Software Engineering: http://www.swe.uni-linz.ac.at/teaching/lva/ws03-04/se_uebung/05_gruppen/g1_ploesch/VModell_01.pdf

Keller, W. (2002). *Technische Universität Wien.* Retrieved April 1, 2011, from Technische Universität Wien: http://www.objectarchitects.de/tu2002/slides/vl02proz.pdf

Kirk, W. (2010). *Gestaltung von Dienstleistungen im allgemeinen Interesse - Prozessmanagement.* Norderstedt: Books on Demand GmbH.

Kolditz, J. (2009). *Vorgehensmodell zur Erstellung von Fachkonzepten für selbststeuernde produktionslogistische Prozesse.* Berlin: GITO Verlag.

Koord, Y., & Krauter, V. (2009, Jänner 24). *WinfWiki*. Retrieved April 3, 2011, from WinfWiki: http://winfwiki.wi-fom.de/index.php/%C3%9Cberblick_Vorgehensmodelle_im_Projektmanagement

Nielsen, L. (2008). *Vorgehensmodell zur ERP-Einführung in kleinen und mittelständischen Unternehmen (KMU).* Norderstedt: GRIN Verlag.

Poe, V., & Reeves, L. (1997). *Aufbau eines Data Warehouse.* München: Prentice Hall.

Pomberger, G., & Pree, W. (2004). *Software Engineering - Architektur-Desgin und Prozessorientierung.* München: Carl Hanser Verlag.

Ruf, W., & Fittkau, T. (2008). *Ganzheitliches IT-Projektmanagement.* München: Oldenbourg Wissenschaftsverlag GmbH.

Tremp, H., & Ruggiero, M. (2001). *Application Engineering - Grundlagen für die objektorientierte Softwareentwicklung mit zahlreichen Beispielen, Aufgaben und Lösungen.* Zürich: Compendio Bildungsmedien AG.

University of Oldenburg. (n.d.). *University of Oldenburg*. Retrieved April 4, 2011, from University of Oldenburg: http://www-cg-hci.informatik.uni-oldenburg.de/~da/eden/Inhalt/Kapitel_4/4.2prozessmodell_fuer_das_fallbeispiel.htm

Wallmüller, E. (2001). *Softwarequalitätsmanagement in der Praxis - Softwarequalität durch Führung und Verbesserung von Software-Prozessen.* München, Wien: Carl Hanser Verlag.

Wehlitz, P. A. (2001). *Nutzenorientierte Einführung eines Produktdatenmanagement-Systems*. München: Herbert Utz Verlag GmbH.

Weiss, C. (1998, Mai 30). *Objektorientiertes Software-Engineering*. Retrieved April 4, 2011, from Objektorientiertes Software-Engineering: http://programmierer.freepage.de/cgi-bin/feets/freepage_ext/41030x030A/rewrite/c.weiss/Diplomarbeit/disp_projektab.htm

Wikipedia. (2011, April 1). *Wikipedia*. Retrieved April 4, 2011, from Wikipedia: http://de.wikipedia.org/wiki/Rational_Unified_Process

Abbildungsverzeichnis

Abbildung 1: Tätigkeitsbereiche 9

Abbildung 2: Visualisierung Prozessmodell 10

Abbildung 3: Visualisierung Rollenmodell 11

Abbildung 4: Sequentielles Vorgehensmodell 15

Abbildung 5: Das Wasserfallmodell 18

Abbildung 6: Wasserfallmodell mit Rückkopplung 19

Abbildung 7: Inkrementelles Prototyping im Fallbeispiel 21

Abbildung 8: Das Spiralmodell 22

Abbildung 9: Das V-Modell 24

Abbildung 10: Die drei Ebenen der Standardisierung 25

Abbildung 11: Gesamtstruktur des V-Modell XT 28

Abbildung 12: Inkrementelles Modell 32

Abbildung 13: Arbeitsschritte beim RUP 34

Abbildung 14: Phasen und Meilensteine beim RUP 35

Abbildung 15: Vorgang in XP 37

Abbildung 16: Scrum – Prozess und Grundsätze 39

Abbildung 17: Phasen des VM bei einer Data-Warehouse-Entwicklung 45

Tabellenverzeichnis

Tabelle 1: Vergleich der Vorgehensmodelle 42

Anhang

A. Abkürzungsverzeichnis

bzw.	beziehungsweise
ca.	circa
dh.	das heißt
DV-Konzept	Datenverarbeitungskonzept
IT	Informationstechnologie
od.	oder
SW-Lebenszyklus	Software-Lebenszyklus
VM	Vorgehensmodell
z.B.	zum Beispiel