JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN
PROFESSUR BWL – WIRTSCHAFTSINFORMATIK
UNIV.-PROF. DR. AXEL C. SCHWICKERT

Scharch, Manuel

Vorgehensmodelle in der Software-Entwicklung

ARBEITSPAPIERE WIRTSCHAFTSINFORMATIK

Nr. 4 / 2016
ISSN 1613-6667

Arbeitspapiere WI Nr. 4 / 2016

Autoren: Scharch, Manuel

Titel: Vorgehensmodelle in der Software-Entwicklung

Zitation: Scharch, Manuel: Vorgehensmodelle in der Software-Entwicklung, in: Arbeitspapiere WI, Nr. 4/2016, Hrsg.: Professur BWL – Wirtschaftsinformatik, Justus-Liebig-Universität Gießen 2016, 67 Seiten, ISSN 1613-6667.

Kurzfassung: In ihren Anfangszeiten wurde die Software-Entwicklung weniger als eine planvolle Tätigkeit angesehen, sondern eher als eine Kunst interpretiert. Dies führte dazu, dass die Softwareentwicklung im Allgemeinen zu teuer, zu wenig termingetreu, zu fehlerhaft und zu wenig wartungsfreundlich war. In den 1960er-Jahren wurde diese Problematik unter dem Begriff „Software-Krise“ zusammengefasst. Eine Herausforderung lag darin, die Aufgaben der Software-Entwicklung schrittweise und in praktikable Teilaufgaben aufzuspalten, so dass sie von einzelnen Entwicklern in größeren Teams übernommen werden konnten. Arbeitsteilige Planung und Integration großer Systeme waren in Ingenieursdisziplinen wie z. B. im Hoch- und Maschinenbau lange bekannt und gelöst. Es erschien zweckmäßig, bei der Entwicklung von Software in ähnlicher Weise systematisch vorzugehen. Aus der ingenieurmäßigen Vorgehensweise der Entwicklung von Programmsystemen ging 1968 der Begriff „Software Engineering“ hervor. Vorgehensmodelle liefern dabei eine Anleitung für die Realisierung von Softwareprojekten; sie stellen heute die Grundlage für die Planung und Steuerung weitgehend aller Software-Entwicklungsvorhaben dar. Das vorliegende Arbeitspapier gibt einen Überblick über die Vielzahl der verfügbaren Vorgehensmodelle. Dazu werden die am weitesten verbreiteten sequentiellen, evolutionären und agilen Vorgehensmodelle vorgestellt.

Schlüsselwörter: Software Engineering, Software-Entwicklung, Vorgehensmodelle, Wasserfallmodell, V-Modell, Spiralmodell, RUP, Prototyping, sequentiell, evolutionär, agil, Xtreme Programming, Scrum, Crystal

Inhaltsverzeichnis

Seite

Abkürzungsverzeichnis

CE…….Concurrent Engineering

RUP……..Rational Unified Process

SE…….Simultaneous Engineering

UML…….Unified modeling language

XP…….Extreme Programming

Abbildungsverzeichnis

Seite

1 Problemstellung, Ziel und Aufbau

Zu Beginn der Software-Entwicklung wurden Computer mithilfe von sogenannten Maschinenbefehlen programmiert. Bald jedoch traten Probleme bei der Entwicklung und Verwendung der Software auf. Besonders das Bedürfnis nach immer aufwendigeren, komplexeren Anwendungen führte rasch an die Grenzen der Durchführbarkeit.[1] Ein Hauptgrund hierfür war die schnelle, eher revolutionär als evolutionär verlaufende Entwicklung der Hardware-Technologie. Diese Entwicklungen hatten zur Folge, dass die Fortschritte in der Softwaretechnik nicht ausreichten, um mit diesen Veränderungen Schritt zu halten.[2] Aus dieser neuen Technologie ergaben sich Möglichkeiten für eine Vielzahl von neuen und komplexeren Anwendungen, für die die Programmiertechnik jedoch noch keineswegs gerüstet war.[3] Die Software-Entwicklung wurde weniger als eine planvolle Tätigkeit angesehen, sondern eher als eine Kunst interpretiert. Dies führte dazu, dass die Softwareentwicklung im Allgemeinen zu teuer, zu wenig termingetreu, zu fehlerhaft und zu wenig wartungsfreundlich war. In den 1960er-Jahren wurde diese Problematik unter dem Begriff „Softwarekrise“ zusammengefasst. Durch diese Krise rückten erstmals die Strukturen des Entwicklungsprozesses selbst in den Mittelpunkt des Interesses bei der Systementwicklung.[4]

In der Folge setzte sich die Erkenntnis durch, dass ein so komplexes Gebilde, wie es eine Software-Anwendung ist, nicht einfach programmiert werden kann. Die Programmierung musste im weiteren Kontext einer geordneten Produktentwicklung gesehen werden.[5] Aus der gewachsenen Größe der Programme ergab sich die Notwendigkeit zur Arbeitsteilung. Viele Aufgaben ließen sich bedingt durch ihre Größe und Komplexität nicht mehr von isoliert arbeitenden Programmierern oder durch kleine Teams lösen. Eine Herausforderung lag darin, die Aufgaben stufenweise in handliche Teilaufgaben aufzuspalten, so dass sie von einzelnen Entwicklern übernommen werden konnten. Die beauftragten Programmier-Teams sahen sich bei der Entwicklung großer Programmier-

1 Vgl. Chroust, Gerhard: Modelle der Software-Entwicklung, München: Oldenbourg 1992, S. 15.

2 Vgl. Balzert, Helmut: Lehrbuch der Softwaretechnik. Basiskonzepte und Requirements Engineering, 3. Aufl., Heidelberg: Spektrum 2009, S. 10.

3 Vgl. Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, München: Oldenbourg 1992, S. 12.

4 Vgl. Bremer, Georg: Genealogie von Entwicklungsschemata, in: Vorgehensmodelle für die betriebliche Anwendungsentwicklung, Hrsg.: Kneuper, Ralf; Müller-Luschnat, Günther; Oberweis, Andreas, Wiesbaden: Teubner 1998, S. 34.

5 Vgl. Chroust, Gerhard: Modelle der Software-Entwicklung, a. a. O., S. 15.

systeme mit einer ganzen Reihe neuer Probleme konfrontiert. Dazu zählten Probleme hinsichtlich der arbeitsteiligen Planung, des Entwurfs und der Integration großer Systeme. Derartige Problemstellungen waren in den Ingenieursdisziplinen, wie z.B. dem Hochbau, bereits bekannt und gelöst. Es erschien zweckmäßig, bei der Entwicklung von Programmsystemen analog vorzugehen. Aus dieser ingenieurtechnischen Vorgehensweise der Entwicklung von Programmen und Programmsystemen ging 1968 der Begriff des „Software Engineerings“ hervor. Die Notwendigkeit der Arbeitsteilung und damit der Führung großer Entwicklergruppen führte aus organisatorischer Sicht zu weiteren neuartigen Problemen. Auch hierbei bediente man sich der Lösungswege aus den klassischen Ingenieursdisziplinen. Es dominierte die Idee, große Entwicklungsprojekte als eine Folge von Phasen zu betrachten und damit der technischen Struktur großer Systeme eine organisatorische Struktur gegenüberzustellen.[6]

Die steigende Komplexität der kollaborativen Anwendungen, wachsende Ansprüche an das Qualitätsmanagement, die Einsparung von Unternehmensressourcen und die Planungssicherheit der durchzuführenden Projekte sind immer häufiger Gründe für die Verwendung sogenannter Vorgehensmodelle. Als Weiterentwicklung der Phasenmodelle bieten diese Modelle eine Anleitung für die Realisierung von Softwareprojekten. Vielfältige Vorgehens- und Tätigkeitsbeschreibungen helfen den leitenden und ausführenden Gruppen, sämtliche Phasen eines Entwicklungsprozesses erfolgreich zu durchlaufen und auf diesem Weg ein optimales Softwareprodukt zu verwirklichen.[7] Vorgehensmodelle stellen heute die Grundlage für die Planung und Steuerung großer Software-Entwicklungsprojekte. Kaum ein großer industrieller Software-Hersteller kommt noch ohne eine Phasen- oder ähnliche Struktur bei der Projektführung aus.[8]

Die Software-Entwicklung spielt gleichzeitig eine immer größere Rolle in der gesamten volkswirtschaftlichen Wertschöpfung.[9] Falsche bzw. fehlerhafte Software kann bei Projekten zu enormen Kosten führen. Es wird immer schwieriger, die stetig wachsende Komplexität von Software-basierten Projekten und Produkten beherrschbar zu machen.

---

6 Vgl. Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 12 ff.

7 Vgl. Zaenger, Roland: Konzeption eines generischen Vorgehensmodells zur Entwicklung kollaborativer Applikationen und prototypische Implementierung am Beispiel einer J2EE-Plattform, Diplomarbeit, Universität Paderborn, Paderborn, 2005, S. 5.

8 Vgl. Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 14.

9 Vgl. Loos, Peter; Fettke, Peter: Aspekte des Wissensmanagements in der Software-Entwicklung am Beispiel von V-Modell und Extreme Programming, TU Chemnitz, Chemnitz, 2001, S. 1.

Sowohl in den Bereichen betrieblicher oder administrativer Informations- und Websysteme als auch bei Cyber-Physischen Systemen wie Auto, Flugzeug, Produktionsanlagen, E-Health-Technologien und mobilen Systemen wurde in den letzten Jahren ein wirkungsvolles Portfolio an Konzepten, Techniken, Methoden und Vorgehensmodellen entwickelt, welche die Softwaretechnik zu einer erwachsenen Ingenieurdisziplin heranreifen ließ.[10]

Die vorliegende Arbeit verfolgt das Ziel, einen Überblick über die Vielzahl der Vorgehensmodelle zu geben. Dazu werden die am weitesten verbreiteten Vorgehensmodelle vorgestellt und näher beschrieben.

Zunächst werden in Kapitel 2 die wichtigsten Definitionen erläutert sowie die Abgrenzung zu nahestehenden Themenbereichen vorgenommen. Weiterhin werden die konzeptuellen Grundlagen von Vorgehensmodellen dargestellt und der Software-Entwicklungsprozess geschildert. Darauf folgen Grundlagen zu Aufbau, Struktur und Einsatz von Phasenmodellen. Anschließend wird eine Einordnung der unterschiedlichen Vorgehensmodelle nach Ebert (2014), in sequenziell, evolutionär und agil vorgenommen. In den Kapiteln 3–5 erfolgt die Vorstellung der sequenziellen, evolutionären und agilen Vorgehensmodelle. Aus Gründen der besseren Vergleichbarkeit geht der Autor bei den einzelnen Vorgehensmodellen immer nach folgendem Schema vor: Nach einer kurzen Einleitung wird die Herkunft bzw. die historische Entwicklung des Vorgehensmodells geschildert. Danach folgt eine Beschreibung der zentralen Charakteristika und des Ablaufs bei dem jeweiligen Modell. Anschließend werden geeignete Einsatzbereiche für das Vorgehensmodell benannt. Abschließend werden Vor- und Nachteile eines jeden Modells gegenübergestellt. Sollte ein Vorgehensmodell über besondere Eigenschaften verfügen, die in keine der oben genannten Kategorien passen, so sind diese unter Besonderheiten aufgeführt. Mögliche zukünftige Entwicklungen von Vorgehensmodellen werden in der Schlussbetrachtung dargelegt.

10 Vgl. Rumpe, Bernhard: Agile Modellierung mit UML, 2. Aufl., Berlin Heidelberg: Springer 2012, S. 1.

2 Grundlagen von Vorgehensmodellen

2.1 Terminologie und Abgrenzung

Für eine Vorstellung und Beschreibung der verschiedenen Vorgehensmodelle in der Software-Entwicklung ist es sachgerecht, Klarheit über die verwendeten Begriffe zu schaffen. Daher dient dieser Abschnitt der Definition von verwendeten Fachbegriffen. Danach folgt eine Abgrenzung und Einordnung des Themenbereichs.

Software-Entwicklung

Durch die hohe Innovationsgeschwindigkeit und die Praxisnähe ist eine solide, einheitliche und systematische Begriffsbildung von „Software“ schwierig. In der Softwaretechnik gibt es bislang noch keine allgemeingültige Terminologie. Jeder Begriffssystematik liegt eine bestimmte Sicht auf die Softwaretechnik zugrunde. Dennoch sollten bereits etablierte Begriffe nicht durch neue, theoretisch besser geeignete Begriffe substituiert werden. Software kann als Sammelbezeichnung für Programme, die für den Betrieb von Rechnersystemen zur Verfügung stehen, verstanden werden. Außerdem gehört zu Software immer auch eine entsprechende Dokumentation sowie evtl. zugehörige Daten.[11]

Die Software-Entwicklung ist ein Teilgebiet der Softwaretechnik (synonym zu Software Engineering).[12] Unter dem Begriff „Software Engineering“ werden alle ingenieurtechnischen Vorgehensweisen zur Entwicklung von Software-Systemen zusammengefasst. Dabei werden die folgenden Fragen geklärt:[13]

- Was muss getan werden?
- Wann muss es getan werden?
- Wie muss es getan werden?
- Womit muss es getan werden?

Die Software-Entwicklung als Teilgebiet der Softwaretechnik umfasst Methoden und Werkzeuge, die zur Neu- und Weiterentwicklung von Software eingesetzt werden. Der

11 Vgl. Balzert, Helmut: Lehrbuch der Softwaretechnik. Basiskonzepte und Requirements Engineering, a. a. O., S. 3 und Sommerville, Ian: Software Engineering, 9. aktual. Aufl. München: Pearson 2012, S. 31.

12 Vgl. Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 24.

13 Vgl. Abts, Dietmar; Mülder, Wilhelm: Grundkurs Wirtschaftsinformatik. Eine kompakte und praxisorientierte Einführung. 8., überarb. und erw. Aufl. Wiesbaden: Springer Vieweg 2013, S. 295.

Entwicklungsprozess eines Softwareprodukts erstreckt sich von der Entstehung über die Inbetriebnahme und Wartung bis zur Ablösung des Produkts durch ein anderes. Er stellt den Ablauf der Entstehung und Entwicklung eines Software-Systems dar, der alle Maßnahmen und Tätigkeiten einschließt, die während dieser Periode erforderlich sind. Dieser Prozess wird auch als Software-Lebenszyklus (Software Life Cycle) bezeichnet[14] und wird in Abschnitt 2.2 dieses Kapitels näher erläutert.

Vorgehensmodelle

Ausgehend vom Entwicklungsprozess lässt sich der Begriff „Vorgehensmodell“ als Ausprägung eines Entwicklungsschemas definieren. Ein Entwicklungsschema (synonym zu Vorgehensstrategie, Entwicklungsansatz)[15] hat die Funktion, das Aufgabengefüge des zu realisierenden Entwicklungsprozesses im Wesentlichen modellhaft darzustellen.[16] Dabei weist der synonyme Gebrauch des Begriffs „Vorgehensstrategie“ auf die langfristige Ausrichtung und den hohen Abstraktionsgrad hin.[17] Vorgehensmodelle werden aus Entwicklungsschemata abgeleitet und zeichnen sich durch eine konkretere Modellierung der Entwicklungsaufgaben im Lebenszyklus von Softwareprodukten aus.[18]

Das Thema Vorgehensmodelle hat in der Software-Entwicklung, nicht zuletzt durch die steigende Komplexität der Software-Systeme, an Bedeutung gewonnen. Oftmals werden spezialisierte Vorgehensmodelle zur Lösung von spezifischen Problemkomplexen definiert und publiziert. Dies führte in den letzten Jahren zu einem starken, unübersichtlichen Wachstum der zur Verfügung stehenden Vorgehensmodelle sowie zu einer uneinheitlichen Begriffsbildung.[19] Nach Horn stellen Vorgehensmodelle die Arbeitsschritte der Software-Entwicklung in einzelnen Phasen dar. Dabei werden die Voraussetzungen und Ergebnisse (Vor- und Nachbedingungen) der einzelnen Phasen detailliert fest-

14 Vgl. Abts, Dietmar; Mülder, Wilhelm: Grundkurs Wirtschaftsinformatik, a. a. O., S. 295 und Fischer, Thomas; Biskup, Hubert; Müller-Luschnat, Günther: Begriffliche Grundlagen für Vorgehensmodelle, in: Vorgehensmodelle für die betriebliche Anwendungsentwicklung, Hrsg.: Kneuper, Ralf; Müller-Luschnat, Günther; Oberweis, Andreas, Wiesbaden: Teubner 1998, S. 18.

15 Vgl. Fischer, Thomas; Biskup Hubert; Müller-Luschnat, Günther: Begriffliche Grundlagen für Vorgehensmodelle, a. a. O., S. 18.

16 Vgl. Bremer, Georg: Genealogie von Entwicklungsschemata, a. a. O., S. 33.

17 Vgl. Schwickert, Axel C.: Web Site Engineering – Ein Komponentenmodell, in: Arbeitspapiere WI, Nr. 12/1998, Hrsg.: Lehrstuhl für Allg. BWL und Wirtschaftsinformatik, Johannes Gutenberg-Universität: Mainz 1998, S. 18.

18 Vgl. Bremer, Georg: Genealogie von Entwicklungsschemata, a. a. O., S. 32 und 34.

19 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt. 2. Aufl. Heidelberg: Spektrum Akad. Verlag 2008, S. 4.

gelegt.[20] Nach Hesse beschreiben Vorgehensmodelle in idealisierender und von Details abstrahierender Weise den Software-Entwicklungsprozess. Ebenfalls werden die auszuführenden Tätigkeiten und zu erbringenden Ergebnisse beschrieben.[21] Laut Balzert beschreiben Vorgehensmodelle den geplanten Gesamtprozess zur Unterstützung bei der Software-Entwicklung. Vorgehensmodelle legen fest, in welcher Abfolge die Aktivitäten zur Entwicklung durchgeführt werden sollen. Dabei hat jede Aktivität die Erstellung eines Zwischenprodukts (Dokumente wie z.B. Anforderungsspezifikationen oder Wissen der Entwickler) zum Ziel.[22] Das nachfolgende, allgemeine Schema soll zur Einordnung und Definition von Vorgehensmodellen beitragen (siehe Abbildung 1).

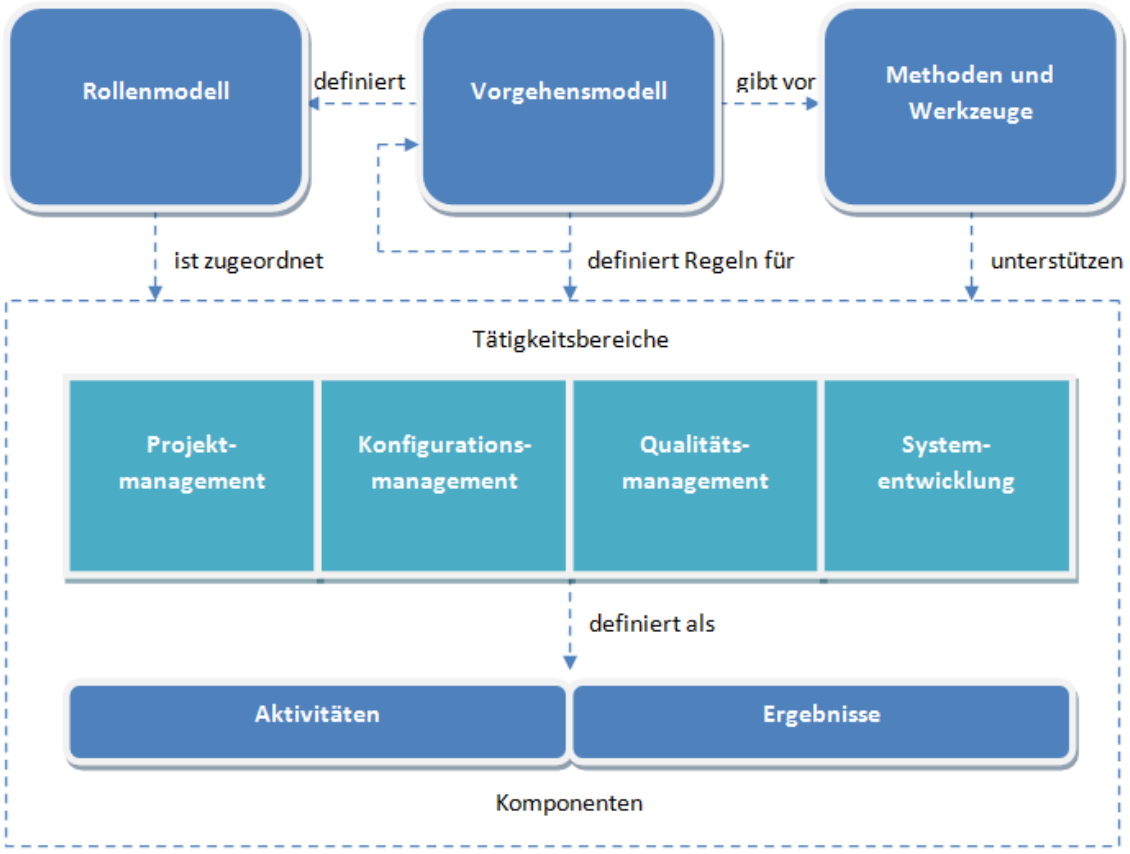

Abbildung 1: Ordnungsschema von Vorgehensmodellen[23]

20 Vgl. Horn, Erika; Schubert, Wolfgang: Objektorientierte Software-Konstruktion. Grundlagen – Modelle – Methoden – Beispiele. München: Carl Hanser Verlag 1993, S. 21.

21 Vgl. Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 104.

22 Vgl. Balzert, Helmut: Lehrbuch der Softwaretechnik. Basiskonzepte und Requirements Engineering, a. a. O., S. 3 und Balzert, Helmut: Lehrbuch der Softwaretechnik. Software-Entwicklung, 2. Aufl., 1. Nachdr. Heidelberg: Spektrum Akad. Verlag 2001, S. 54.

23 In Anlehnung an Fischer, Thomas; Biskup Hubert; Müller-Luschnat, Günther: Begriffliche Grundlagen für Vorgehensmodelle, a. a. O., S. 17.

Entsprechend diesem Schema der Fachgruppe „Vorgehensmodelle“ der Gesellschaft für Informatik[24] strukturiert ein Vorgehensmodell den Software-Entwicklungsprozess in Aktivitäten (Aufgaben) und Ergebnisse (die zu entwickelnden Resultate) und legt Regeln für die Abarbeitung der Aktivitäten und der daraus resultierenden Ergebnisse fest. Weiterhin bestimmt ein Vorgehensmodell Methoden und Werkzeuge, die die Erarbeitung der Ergebnisse unterstützen. Außerdem werden Rollen (die Menge an zusammengehörigen Aufgaben und Befugnissen einer Person) vorgeschrieben, die die Aktivitäten der verschiedenen Tätigkeitsbereiche verantworten und durchführen.[25]

Sequenziell

Bei einem sequenziellen Entwicklungsprozess werden die einzelnen Entwicklungsphasen streng nacheinander, top-down vom Groben zur Verfeinerung, durchgeführt. Dabei wird jede einzelne Phase abgeschlossen und nur einmal durchlaufen.[26]

Inkrementell

Hierbei wird die Software nach und nach realisiert. Die erste Stufe bildet das Kernsystem. Softwareprodukte werden schrittweise weiterentwickelt, wobei nach jedem Abschluss eines Schrittes eine weitere funktionsfähige Softwarekomponente (ein Inkrement) vorliegt.[27]

Iterativ

Unter einem iterativen Entwicklungsprozess wird die Entstehung der Software aus einer Abfolge von unterschiedlichen Entwicklungszyklen verstanden. Die einzelnen Entwicklungsphasen werden mehrfach durchlaufen. Modelle dieses Typs gehen von einem evolutionären Gesamtprozess aus, durch den ein Softwaresystem laufend verbessert wird.[28]

---

24 Die Gesellschaft für Informatik (GI) ist ein Zusammenschluss von Menschen, die einen engen Bezug zur Informatik haben und sich für dieses Fachgebiet mit all seinen Facetten und Anwendungsgebieten interessieren. Weitere Informationen sind unter folgender Website erhältlich: https://www.gi.de/

25 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 4.

26 Vgl. Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, 11. Völlig neu bearb. Aufl. Berlin: de Gruyter 2015, S. 353 und Filß, Christian: Vergleichsmethoden für Vorgehensmodelle, Diplomarbeit, TU Dresden, Dresden, 2005, S. 56.

27 Vgl. Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, a. a. O., S. 353 und Filß, Christian: Vergleichsmethoden für Vorgehensmodelle, a. a. O., S. 55.

28 Vgl. Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, a. a. O., S. 353 und Filß, Christian: Vergleichsmethoden für Vorgehensmodelle, a. a. O., S. 55.

Evolutionär

In einem bereits betriebenen System werden neue Anforderungen deutlich, die den Ausgangspunkt für einen weiteren Entwicklungszyklus bilden. Dieses Vorgehen findet häufig bei Systemen Anwendung, deren Anforderungen nicht von Beginn an genau bestimmt werden können.[29]

Agil

Unter Agilität wird die Fähigkeit verstanden, auf Veränderungen zu reagieren. Agile Vorgehensmodelle werden auch als leichtgewichtige Vorgehensmodelle bezeichnet. Charakteristisch sind kleine, meist unbürokratische Teilprojekte mit greifbaren Ergebnissen, anpassbaren Vorgaben, Teamwork und weitgehender Selbstorganisation.[30]

Vorgehensmodelle erfüllen zwei Aufgaben. Zum einen Abstraktion und Nachbildung: Sie beschreiben die Gemeinsamkeiten vieler realer, bereits durchgeführter Prozesse. Es wird von jenen Details abgesehen, die sich zwischen den Prozessen unterscheiden, und das Augenmerk wird ausschließlich auf allgemeingültige Aspekte gelegt. Das bedeutet die mentale Trennung des Produkts – das bei jedem Prozess anders ist – von dem Prozess. Zum anderen zählen zu den Aufgaben der Vorgehensmodelle die Bereiche Vorbild-Erstellung und Definition. Vorgehensmodelle sind Muster und Vorbild für die Durchführung der Prozesse. Dabei muss ein Vorgehensmodell auch künftige Verallgemeinerungen seiner Verwendung, z.B. für neue Produktklassen, berücksichtigen.[31]

Die Festlegung für ein bestimmtes Vorgehensmodell bedeutet immer die Trennung der Beschreibung des Prozesses von seiner Ausführung. Es wird also zwischen Prozess und Produkt klar unterschieden.[32] Daraus ergeben sich zwei Sichtweisen der Planung und Entwicklung von IT-Systemen:

1. *Ergebnissicht:* Das „Was“ der Entwicklung (die Gestaltung und Darstellung des IT-Systems mit Modellierungsansätzen wie z.B. funktions-, datenfluss- oder objektorientierte Modellierung)
2. *Prozesssicht:* Das „Wie“ der Entwicklung (die Vorgehensweise der Entwicklung, die Prozessgestaltung zur Entwicklung des IT-Systems)

29 Vgl. Filß, Christian: Vergleichsmethoden für Vorgehensmodelle, a. a. O., S. 55.

30 Vgl. Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, a. a. O., S. 357 und Brandstäter, Jonathan: Agile IT-Projekte erfolgreich gestalten. Wiesbaden: Springer Fachmedien 2013, S. 9.

31 Vgl. Chroust, Gerhard: Modelle der Software-Entwicklung, a. a. O., S. 39 f.

32 Vgl. Chroust, Gerhard: Modelle der Software-Entwicklung, a. a. O., S. 32.

Die vorliegende Arbeit deckt den Bereich aus Prozesssicht ab und klammert Fragestellungen zu technischen Aspekten (Programmierung, Sprachen, etc.) aus. Auch werden sogenannte Projektmanagement-Konzepte (wie z.B. PRINCE2) nicht betrachtet.

Eine professionelle Entwicklung von Anwendungssystemen erfolgt in Form von Projekten. Aufgabe des Projektmanagements sind Fragestellungen bezüglich wer was wann zu welchen Kosten auszuführen hat. Demgegenüber beschäftigen sich Vorgehensmodelle mit Fragestellungen bezüglich der Reihenfolge der auszuführenden Aktivitäten und den zum Einsatz kommenden Methoden und Werkzeugen, also wie und womit die Aktivitäten ausgeführt werden.[33] Wie diese Reihenfolge im Allgemeinen aussehen kann und welche Grundeinteilung angewendet wird, ist Bestandteil des folgenden Abschnitts.

2.2 Allgemeines Vorgehensmodell

Jede Produktentwicklung erfolgt in bestimmten Schritten, die unabhängig von der Art des Produkts sind. Zunächst müssen die Anforderungen geklärt werden. Danach werden die Strukturen der Lösung festgelegt, bevor die Komponenten des Produkts angefertigt und zusammengefügt werden. Fällt die Prüfung des Ergebnisses positiv aus, wird das Produkt in Gebrauch genommen. Nachfolgend wird das Produkt benutzt und gewartet, bis es schließlich ausgemustert und durch ein anderes Produkt ersetzt wird. Dieser Ablauf wird im Englischen auch als „Life Cycle“ bezeichnet.[34] Bezieht man das „Life Cycle“-Modell auf die Software-Entwicklung, ergibt sich das Schema aus Abbildung 2 in Form eines „Software Life Cycles“.

Der Begriff des „Software Life Cycle“ wird im amerikanischen Sprachgebrauch verwendet. Er zeigt eine (idealisierte) Einteilung eines Software-Projektes in Phasen.[35] Am Beginn des Software Life Cycle steht ein grober Plan, der in der Analyse konkretisiert und in der Spezifikation dokumentiert wird. Darauf folgen die Festlegung der Struktur und die Anfertigung des Produkts mit Test auf Funktionalität. Schließlich wird das Pro-

33 Vgl. Stahlknecht, Peter; Hasenkamp, Ulrich: Einführung in die Wirtschaftsinformatik, 11. Vollst. Überarb. Auflage. Berlin, Heidelberg: Springer 2005, S. 209; 459.

34 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken. Heidelberg: dpunkt 2007, S. 155 und Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 30.

35 Vgl. Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 30.

dukt an den Kunden übergeben und in Betrieb genommen. Solange ein Produkt in Betrieb ist, werden immer wieder Korrekturen und Modifikationen notwendig.[36]

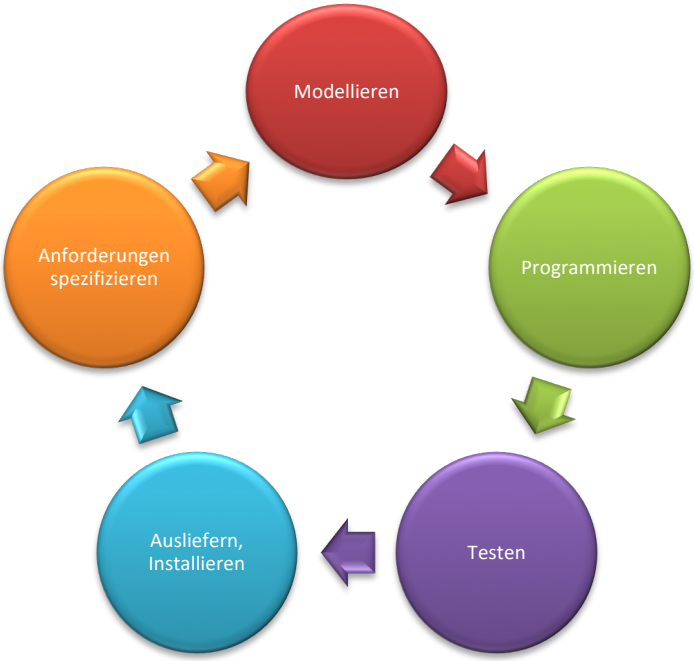

Abbildung 2: Software Life Cycle als geschlossener Kreislauf[37]

Jede Software-Erstellung soll in einem festgelegten organisatorischen Rahmen durchgeführt werden. Ein Vorgehensmodell bildet einen solchen Rahmen.[38] Vorgehensmodelle (auch: Life Cycle Modelle) sind dafür zuständig, den komplexen Prozess der Entwicklung und der anschließenden Wartung in überschaubare Teilaktivitäten zu zerlegen und deren Ergebnisse sowie deren logischen und zeitlichen Zusammenhang zu definieren.[39] Eine solche Gliederung schafft die Voraussetzung, den Mitarbeitereinsatz zu planen und die Zuständigkeiten zu regeln, den Projektfortschritt anhand der erreichten Ergebnisse zu kontrollieren, bei Abweichungen rechtzeitig steuernd einzugreifen und den Aufwand für weitere Aktivitäten abschätzen zu können.[40]

---

36 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 155.

37 In Anlehnung an Abts, Dietmar; Mülder, Wilhelm: Grundkurs Wirtschaftsinformatik, a. a. O., S. 295.

38 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, 2. Aufl. Heidelberg: Spektrum Akad. Verl. 2008, S. 98.

39 Vgl. Gadatsch, Andreas: Management von Geschäftsprozessen. Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker, 2. überarb. und erw. Aufl. Wiesbaden: Vieweg & Teubner Verlag 2002, S. 74.

40 Vgl. Abts, Dietmar; Mülder, Wilhelm: Grundkurs Wirtschaftsinformatik, a. a. O., S. 296.

Ein allgemein verwendbares, jedoch noch recht grobes Vorgehensmodell zeigt Abbildung 3. Zu Beginn steht ein zu lösendes Problem. In der Analysephase wird eine Bestandsaufnahme der aktuellen Situation vorgenommen. Anschließend werden Ziele definiert und Alternativen bewertet. Eine Ausgestaltung des zukünftigen Systems erfolgt in der Realisierungsphase. Das Modell endet mit der Nutzung und Wartung des eingeführten Systems.[41]

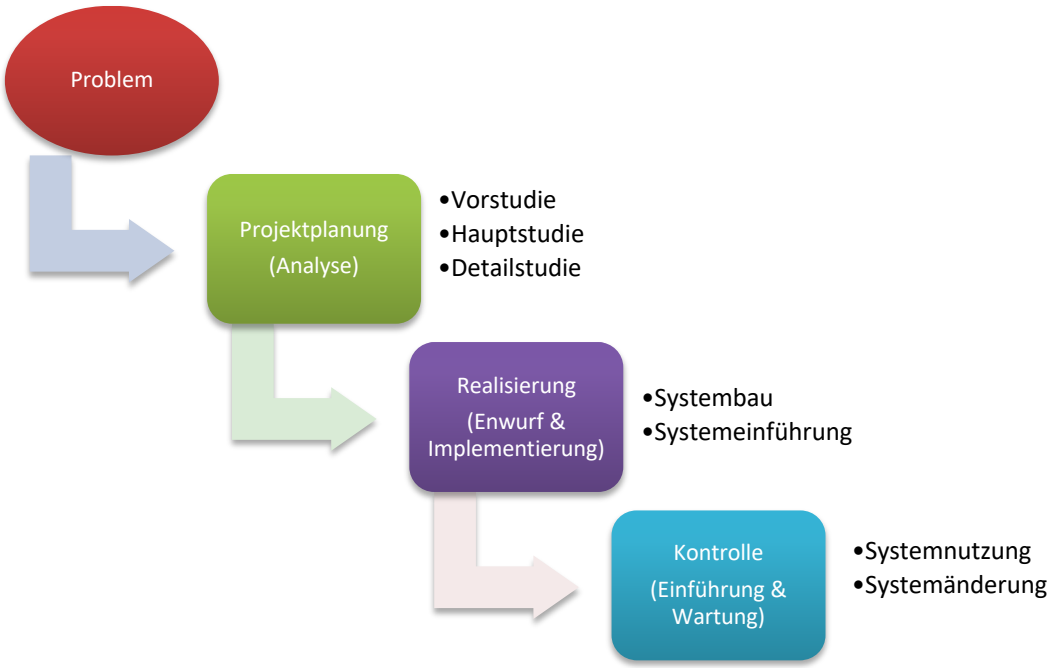

Abbildung 3: Allgemeines Vorgehensmodell[42]

Für den Ablauf eines Projektes ist es von Relevanz, zur Reduzierung der Komplexität, aber auch zur besseren Plan- und Kontrollierbarkeit, einzelne, zeitlich aufeinanderfolgende Phasen einzuführen. Vielen unterschiedlichen Vorgehensmodellen kann eine gemeinsame Grundstruktur in Form der drei Phasen aus Abbildung 3 zugeordnet werden. In jeder Phase werden logisch bzw. sachlich zusammengehörige Aktivitäten ausgeführt. Für jede einzelne Phase werden Ziele, Aktivitäten und Ergebnisse definiert. Die Projektplanungsphase wird unterteilt in die Teilphasen Vor-, Haupt- und Detailstudie. Systembau und -einführung sind Bestandteil der Realisation. In der Nutzungsphase wird das neue System verwendet, dabei bewertet und evtl. geändert. Daher ist hierbei auch

41 Vgl. Abts, Dietmar; Mülder, Wilhelm: Grundkurs Wirtschaftsinformatik, a. a. O., S. 424.

42 Eigene Darstellung in Anlehnung an Abts, Dietmar; Mülder, Wilhelm: Grundkurs Wirtschaftsinformatik, a. a. O., S. 425 und Bea, Franz Xaver; Göbel, Elisabeth: Organisation, 3. neu bearb. Aufl. Stuttgart: Lucius & Lucius 2006, S. 511.

von der „Kontrollphase“ die Rede.[43] Ein Phasenmodell, als Definition der Aneinanderreihung von Tätigkeiten bzw. Phasen, ist folglich ebenfalls ein Submodell eines Vorgehensmodells. Es muss nicht explizit in dieser Form definiert sein. Durch eine Festlegung von bestimmten Aktivitäten und deren Abfolge in einem Vorgehensmodell liegt ein implizites Phasenmodell stets zugrunde.[44]

Als Phase wird ein zusammenhängender Zeitraum verstanden, in dem bestimmte Arbeiten durchgeführt und abgeschlossen werden. Am Ende einer Phase steht ein sogenannter Meilenstein. Dieser gibt bestimmte Kriterien vor, die Auskunft darüber geben, ob eine Phase erfolgreich beendet wurde. Nach Erfüllung dieser Kriterien beginnt die nächste Phase. Die Phasen überlappen sich also nicht (siehe Abbildung 4). In größeren Projekten kann es jedoch verschiedene Entwicklungsstränge geben, die parallel ablaufen und durch unterschiedliche Meilensteine gegliedert sind. Eine Aufteilung in Phasen erleichtert die Kontrolle eines Projekts, da nicht das gesamte Projekt auf einmal betrachtet und finanziert werden muss, sondern nur einzelne Abschnitte. An der Prüfung der Zwischenergebnisse, am Meilenstein, ist der Kunde beteiligt.[45]

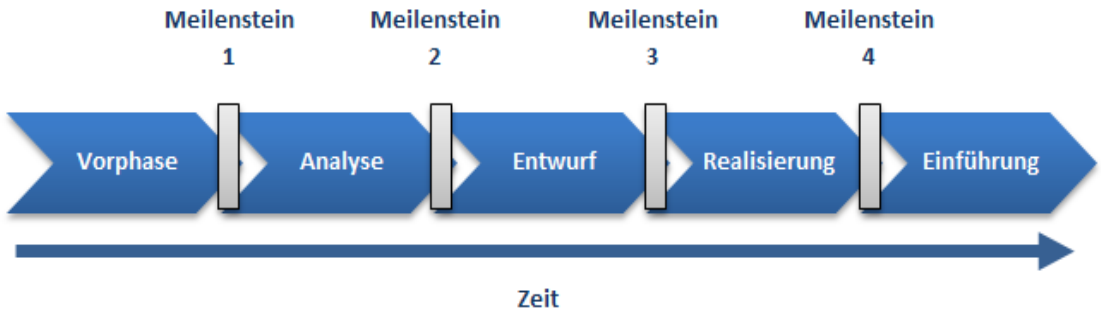

Abbildung 4: Phasenmodell mit Meilensteinen[46]

Konventionelle Phasenmodelle beschränken sich auf eine lineare (sequenzielle) Abfolge von Entwicklungsschritten (Phasen), die eine zeitlich, begrifflich, technisch und/oder organisatorisch begründete Zusammenfassung von Tätigkeiten beinhalten.[47]

43 Vgl. Abts, Dietmar; Mülder, Wilhelm: Grundkurs Wirtschaftsinformatik, a. a. O., S. 424 und Bea, Franz Xaver; Göbel, Elisabeth: Organisation, 3., neu bearb. Aufl. Stuttgart: Lucius & Lucius 2006, S. 511.

44 Vgl. Filß, Christian: Vergleichsmethoden für Vorgehensmodelle, a. a. O., S. 3.

45 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 110.

46 Eigene Darstellung in Anlehnung an Stahlknecht, Peter; Hasenkamp, Ulrich: Einführung in die Wirtschaftsinformatik, a. a. O., S. 210.

47 Vgl. Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 29 und 39.

Es ist notwendig, Projekte in sogenannte Arbeitspakete aufzuteilen, damit das Projekt übersichtlich bleibt. Ursprünglich war eine Phase ein Zeitraum, in dem nur ein einziger Entwicklungsschritt wie z.B. Anforderungen benennen, Programmierung oder Testung ausgeführt wurde. Strenge sequenzielle Abfolgen von Entwicklungsschritten werden heutzutage nicht mehr postuliert, da Projekte mittlerweile flexibler durchgeführt werden. Verschiedene Aktivitäten dürfen sich überlappen. Beispielsweise können einzelne Gruppen bereits testen, während andere noch programmieren.[48]

Moderne Vorgehensmodelle gehen weit über reine Phasenkonzepte hinaus. Die Vielzahl an Vorgehensmodellen, die in den letzten Jahren publiziert wurden, haben unterschiedliche Eigenschaften. Anhand dieser Eigenschaften lassen sich Vorgehensmodelle klassifizieren. Der folgende Abschnitt zeigt eine mögliche Einordnung von Vorgehensmodellen in drei Kategorien.

2.3 Charakterisierung von Vorgehensmodellen

Vorgehensmodelle mit gemeinsamen Eigenschaften lassen sich in sogenannte Familien einordnen. Sequenzielle Vorgehensmodelle beschreiben beispielsweise eine aufeinanderfolgende Abarbeitung der anfallenden Entwicklungsaktivitäten. Innerhalb einer Familie können sich Vorgehensmodelle hinsichtlich der Beziehung einzelner Entwicklungsschritte, der Zuordnung von Aktivitäten und Teilergebnissen zu bestimmten Entwicklungsschritten und bezüglich des Detaillierungsgrades einzelner Aktivitäten unterscheiden.[49]

Sämtliche Vorgehensmodelle haben die Gemeinsamkeit, dass Anforderungen vor ihrer jeweiligen Umsetzung ermittelt und analysiert werden. Allerdings geschieht dies mit unterschiedlichem Tiefgang. In jedem Vorgehensmodell können sich die Anforderungen während des gesamten Entwicklungsprojekts ändern. Daher müssen sie auch währenddessen durchgehend verfolgt werden. Vorgehensmodelle haben sich über die Zeit entwickelt und werden anhand der Randbedingungen von Projekt und Produkt ausgewählt. Die Auswahl von Vorgehensmodellen, Prozessen, Methoden sowie die Dokumentation ist abhängig davon, wie lange ein Produkt gepflegt werden soll, wie viele

48 Vgl. Goll, Joachim: Methoden und Architekturen der Softwaretechnik. Wiesbaden: Vieweg + Teubner Verlag (Studium) 2011, S. 78.

49 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 3.

Kunden damit arbeiten werden, wie lang der Lebenszyklus ist und welche Risiken, Unsicherheiten und Randbedingungen mit dem Projekt in Verbindung stehen.

Abbildung 5 zeigt eine Einteilung von populären Vorgehensmodellen in drei Familien. Die Einteilung und der Einsatz erfolgt anhand von zwei Dimensionen, die durch Anforderungen bestimmt werden. Die horizontale Achse beschreibt die Stabilität der Marktanforderungen und die vertikale Achse Anforderungen an den Projektcharakter.[50]

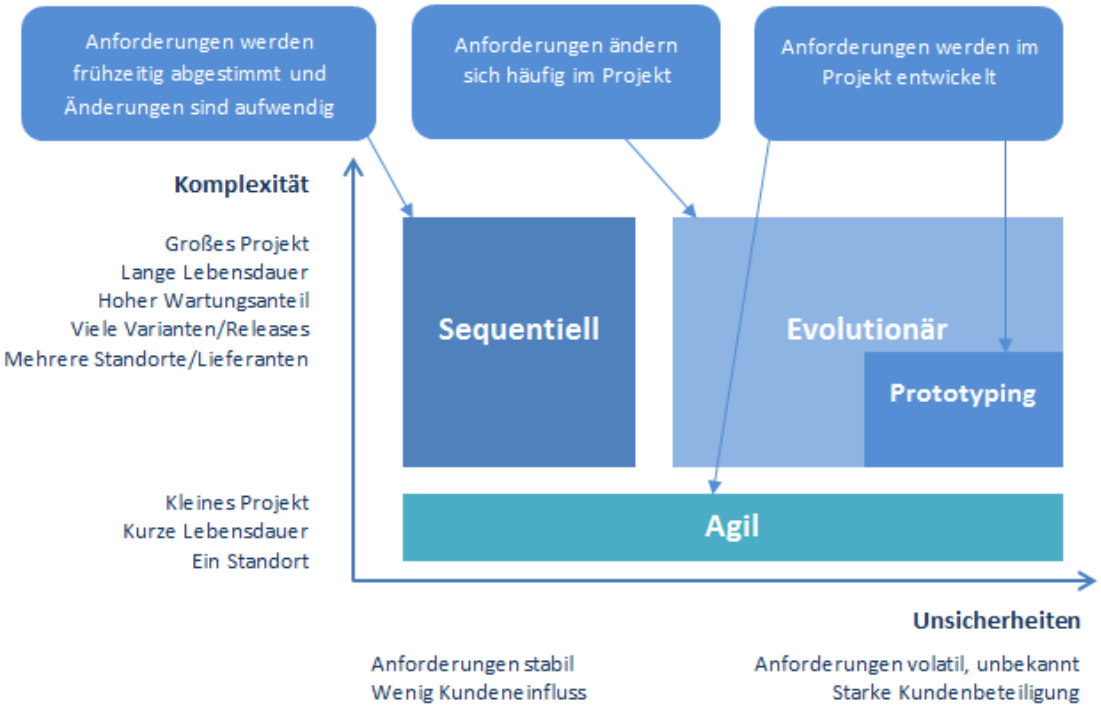

Abbildung 5: Charakterisierung der Vorgehensmodelle nach Ebert[51]

Die erste Familie bilden die sequenziellen Vorgehensmodelle. In diese Gruppe fallen beispielsweise das Wasserfall- und das V-Modell. Vorgehensmodelle dieser Kategorie zeichnen sich dadurch aus, dass die anfallenden Aktivitäten in Phasen eingeteilt und anschließend nacheinander abgearbeitet werden. Dieses Vorgehen folgt ähnlich den Beschreibungen des sequenziellen Phasenmodells aus Abschnitt 2.2 dieses Kapitels.

Die nächste Familie umfasst die Gruppe der evolutionären Vorgehensmodelle. Bei diesen Modellen liefert die Anforderungsanalyse für die Entwicklung keine ausreichenden

50 Vgl. Ebert, Christof: Systematisches Requirements Engineering. Anforderungen ermitteln, spezifizieren, analysieren und verwalten, a. a. O., S. 311 f.

51 In Anlehnung an Ebert, Christof: Systematisches Requirements Engineering. Anforderungen ermitteln, spezifizieren, analysieren und verwalten, 5. überarb. Aufl. Heidelberg: dpunkt Verl. 2014, S. 312.

Resultate. Es wird versucht, durch Zyklen (Iterationen) aus Erprobung und Verbesserung das Produkt so oft zu verändern und zu erweitern, bis es eingesetzt werden kann. Dabei fallen oft mehrere funktionsfähige Softwarekomponenten (Inkremente) an. Der Literatur lässt sich keine präzise Unterscheidung zwischen inkrementeller und evolutionärer Entwicklung entnehmen.[52] Das sogenannte Prototyping ist der evolutionären Entwicklung zwar in manchen Punkten ähnlich, jedoch ist der eigentliche Zweck des Prototypings das Entwerfen, Konstruieren, Bewerten und Revidieren von Prototypen, um Anforderungen zu klären und somit eine lange und teure Entwicklung zu vermeiden.[53] Streng genommen handelt es sich beim Prototyping nicht um ein Vorgehensmodell, sondern eher um eine Technik, die den Fokus auf ein Produkt (vgl. Ergebnissicht aus Abschnitt 2.1) legt. Da das Prototyping jedoch von vielen Autoren im Zusammenhang mit Vorgehensmodellen genannt wird, soll es der Vollständigkeit halber auch in dieser Arbeit Erwähnung finden.

Zur dritten Familie gehören die agilen Vorgehensmodelle. Eine pauschale Charakterisierung dieser Modelle ist nicht möglich. Vorrangig besteht ihre Ähnlichkeit in der Ablehnung von bürokratischen Prozessen. Die im Folgenden behandelten Vertreter – Scrum, Extreme Programming und Crystal – unterscheiden sich zum Teil beträchtlich.
In welcher Weise genau die einzelnen Vorgehensmodelle voneinander abweichen, wird auf den folgenden Seiten herausgearbeitet.

52 Vgl. Gnatz, Michael: Vom Vorgehensmodell zum Projektplan. Saarbrücken: VDM Verl. 2007, S. 21.

53 Vgl. Ebert, Christof: Systematisches Requirements Engineering. Anforderungen ermitteln, spezifizieren, analysieren und verwalten, a. a. O., S. 165.

3 Sequenzielle Vorgehensmodelle

3.1 Wasserfallmodell

Eines der ersten und bekanntesten Vorgehensmodelle in der Software-Entwicklung ist das Wasserfallmodell. Dieses Modell stellt den Software-Lebenszyklus als „Wasserfall“ dar. Dabei stehen die einzelnen Kästchen (siehe Abbildung 6) für Phasen bzw. Aktivitäten, die nacheinander abgearbeitet werden.[54]

Entstehung

Bei dem Wasserfallmodell handelt es sich um eines der ersten Vorgehensmodelle, das im Software Engineering bekannt ist.[55] Die Geschichte des Wasserfallmodells reicht zurück bis auf Herbert Benington, der 1956 erstmals ein Modell beschrieb, das aus neun Phasen bestand und einer Top-Down-Methode entsprechend abgearbeitet wurde. Jede Phase musste vollständig abgeschlossen sein, bevor eine neue Phase beginnen konnte. Am Ende einer jeden Phase musste jeweils ein Dokument fertiggestellt sein. Dieses Phasenmodell wurde in einem Entwicklungsprojekt für das Luftverteidigungssystem SAGE angewendet.[56] Diese Grundstruktur wurde von Winston Royce im Jahre 1970 aufgegriffen und als „Wasserfallstruktur“ dargestellt. Das Wasserfallmodell ist demnach eine Verbesserung des „Neun-Phasen-Modells“ von Benington.[57] Ein Problem von Beningtons Modell war, dass es keinen Informationsfluss (Feedback) entgegen des Phasenverlaufs gab. Diese Erweiterung wurde von Royce 1970 eingeführt und als Rückkopplung bezeichnet. Ebenfalls wurde bereits hier ein erstes Konzept des Prototypings integriert.[58] Einige Konzepte von modernen Vorgehensmodellen wurden also bereits in diesem frühen Modell eingeführt. Erst die Erweiterungen um die sogenannten „Feedback-Schleifen“ verhalfen dem Modell, das durch Barry Boehm 1976 erstmals als Wasserfallmodell[59] bezeichnet wurde, zu großer Popularität und zahlreichen Anwendern.

54 Vgl. Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 33.

55 Vgl. Versteegen, Gerhard: Vorgehensmodelle, in: Software Management. Beherrschung des Lifecycles, Hrsg.: Versteegen, Gerhard, Berlin, Heidelberg: Springer 2002, S. 30.

56 Vgl. Benington, Herbert D.: Production of Large Computer Programs. Reprinted in: Annals of the History of Computing, 5, Nr. 4, Oktober 1983, 1956, S. 350–361.

57 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 99.

58 Vgl. Royce, Winston: Managing the Development of Large Software Systems, in: IEEE WESCON Hrsg.: Institute of Electrical and Electronics Engineers, 26. Aufl., August 1970, S. 328–338.

59 Vgl. Boehm, Barry W.: Software Engineering, in: IEEE Transact. on Computers, C-25, Dezember 1976, S. 1216–1241.

Das Wasserfallmodell ist demzufolge kein rein sequenzielles Vorgehensmodell, allerdings wird dies – fälschlicherweise – oft angenommen. Es lässt sich zwischen zwei Varianten von Wasserfallmodellen unterscheiden. Das rein sequenzielle „Baseline-Model" ohne Rückführschleifen (Benington) und das Wasserfallmodell mit Rückführschleifen (auch Schleifenmodell genannt).[60] Ob Rückschritte lediglich in direkte Vorgängerphasen oder auch in andere Phasen erlaubt sind, hängt von der konkreten Ausprägung des Wasserfall- bzw. Schleifenmodells ab. Zu beachten bleibt jedoch, dass Rückschritte in vorangegangene Phasen, im Gegensatz zu iterativen Vorgehensmodellen, bei diesen Modellen als Ausnahmefall zu betrachten sind.[61]

Zentrale Charakteristika und Ablauf

Das Softwaresystem wird mithilfe der Top-Down-Methode spezifiziert und anschließend schrittweise konkretisiert. Als Grundlage für den gesamten Entwicklungsprozess dient die Spezifikation als Leistungsbeschreibung. Alle wichtigen Anforderungen der späteren Anwender werden also von vornherein ermittelt und festgeschrieben. Die Phasen der Softwareproduktion entsprechen den einzelnen „Arbeitsgängen". Die Ergebnisse einer Phase fallen in die nächste, um dort entsprechend weiterverarbeitet zu werden. Dort dienen die Ergebnisse als Meilensteine, die dafür vorgesehen sind, den Projektfortschritt zu überprüfen.[62] Beim Übergang wird vorausgesetzt, dass die vorhergehende Phase abgeschlossen ist. Zur Verständigung zwischen den Entwicklern und Anwendern werden umfangreiche Dokumente über das zu realisierende Softwaresystem erstellt. Das Softwaresystem wird einmal hergestellt und anschließend gewartet, wobei die Nutzung des Systems während der Entwicklung nicht im Vordergrund steht. Beim Wasserfallmodell bzw. Schleifenmodell werden Rücksprünge zur jeweils vorgelagerten Phase eingeführt. An dieser Stelle beinhaltet jeder Phasenabschluss einen Validations- und Verifikationsschritt zur vorherigen Phase, um unerwünschte Nebeneffekte der sequenziellen Konkretisierung abzufangen. Sämtliche Änderungen an bereits akzeptierten Ergebnissen müssen auf die vorhergehende Phase begrenzt bleiben.[63]

Die wichtigen Phasen des Wasserfallmodells lassen sich auf grundlegende Entwicklungsaktivitäten abbilden (siehe Abbildung 6):

60 Vgl. Goll, Joachim: Methoden und Architekturen der Softwaretechnik, a. a. O., S. 84.
61 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 6.
62 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 5.
63 Vgl. Bremer, Georg: Genealogie von Entwicklungsschemata, a. a. O., S. 39 f.

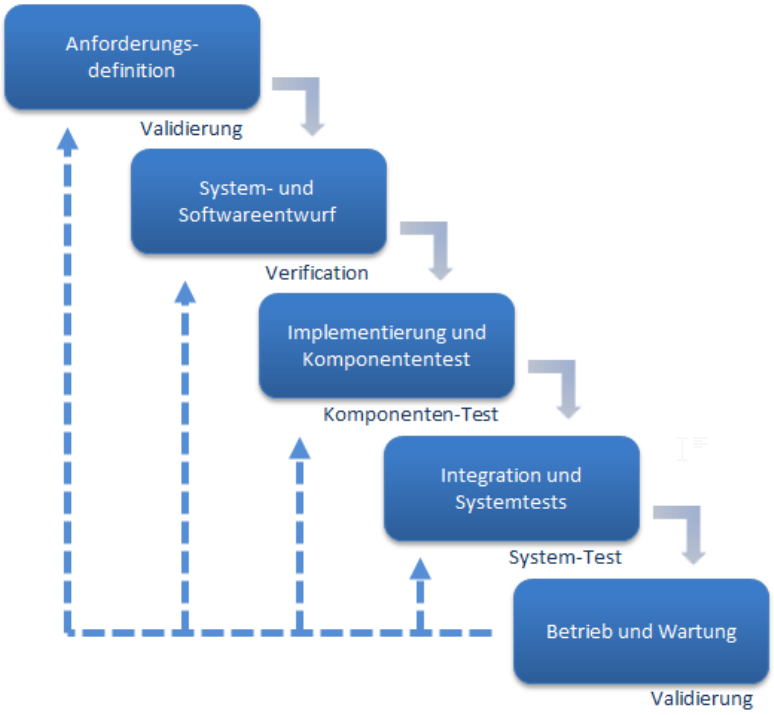

Abbildung 6: Wasserfallmodell mit Rücksprüngen[64]

1. *Analyse und Definition der Anforderungen*: In Zusammenarbeit mit den Systembenutzern werden die Dienstleistungen, Einschränkungen und Ziele aufgestellt. Es folgt eine detaillierte Definition, die als Systemspezifikation dient.
2. *System- und Software-Entwurf*: Während des Systementwurfsprozesses werden die Anforderungen in Hard- und Software-Systeme aufgeteilt. Es wird eine allgemeine Systemarchitektur festgelegt. Das Erkennen und Beschreiben der grundlegenden abstrakten Software-Systeme und ihrer Beziehungen zueinander ist Bestandteil des Software-Entwurfs.
3. *Implementierung und Komponententest*: Der Software-Entwurf wird in eine Reihe von Programmen oder Programmeinheiten umgesetzt. Durch Tests wird sichergestellt, dass jede Einheit ihre Spezifikationen erfüllt.
4. *Integration und Systemtest*: Die einzelnen Programme oder Programmeinheiten werden zusammengeführt bzw. integriert und als Ganzes getestet, um zu gewährleisten, dass die Software-Anforderungen erfüllt werden. Nach erfolgreichen Tests folgt die Auslieferung des Software-Systems an den Kunden.

64 In Anlehnung an Sommerville, Ian: Software Engineering, a. a. O., S. 96.

5. *Betrieb und Wartung*: In der Regel ist dies die längste Phase innerhalb des Lebenszyklus. Das System wird installiert und zum Gebrauch freigegeben. Zur Wartung zählen die Korrektur von Fehlern, die in den frühen Phasen nicht entdeckt wurden, die Verbesserung der Implementierung von Systemeinheiten und die Verbesserung des Systems bei neuen Anforderungen.[65]

Einsatzbereiche

Trotz seiner langen Geschichte ist das Wasserfallmodell auch heutzutage noch dazu geeignet in unterschiedlichen Projekten eingesetzt zu werden, auch z. B. zur Entwicklung von Prototypen. Prototypen zeichnen sich durch kurze Entwicklungszeiten aus und dienen lediglich zu Demonstrationszwecken. Daher ist das Wasserfallmodell durchaus für ihre Entwicklung geeignet. Des Weiteren kann das Modell bei der Entwicklung von Kleinprogrammen seine Stärken ausspielen. Hierbei handelt es sich um Projekte, bei denen ein oder zwei Entwickler über einen kurzen Zeitraum an einer kleineren Anwendung arbeiten. Bei Kleinprojekten sind die Anforderungen meist klar definiert und die Notwendigkeit von Änderungen ist nahezu ausgeschlossen. Die Programmierung von Hardware im Embedded-Bereich lässt sich ebenfalls durch das Wasserfallmodell abdecken, da hier Änderungsanträge im Laufe der Entwicklung eher als unwahrscheinlich gelten und die Entwicklungsdauer begrenzt ist.[66]

Der Einsatz eines Wasserfallmodells ist immer dann zu empfehlen, wenn die Anforderungen an ein zu entwickelndes System leicht verständlich sind. Sie müssen beschreibbar und möglichst stabil sein. Die häufige Änderung der Anforderungen führt im schlimmsten Fall dazu, dass die Analysephase nicht verlassen wird. Bei Anforderungsänderungen nach Beendigung der Analysephase müssen alle bereits fertiggestellten Ergebnisse überarbeitet werden, da das System basierend auf der beschriebenen Menge von Anforderungen vollständig entworfen und implementiert wird.[67]

Vor- und Nachteile

Die Vorteile des Wasserfallmodells liegen in der klaren Struktur des Vorgehens und seiner einfachen Darstellung. Es gibt aus der Sicht des Projektmanagements klare, einfach verständliche, lineare und gut kontrollierbare Abläufe vor, die sich auf einer Zeit-

65 Vgl. Sommerville, Ian: Software Engineering, a. a. O., S. 96 f.

66 Vgl. Versteegen, Gerhard: Vorgehensmodelle, Software Management. Beherrschung des Lifecycles, a. a. O., S. 32.

67 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 7.

achse abbilden lassen[68]. Außerdem wird projektbegleitend in jeder Phase die Dokumentation erzeugt. Die Phase der Systemdokumentation fällt somit weg.

Das Wasserfallmodell passt aufgrund seiner überschaubaren Struktur zu anderen Vorgehensmodellen und lässt sich in diese integrieren.[69]

Zu Problemen können allerdings Änderungen führen, die im Nachhinein umgesetzt werden sollen, da ein iteratives Vorgehen (zumindest in der ursprünglichen Form) nicht unterstützt wird. Es fehlt die Möglichkeit, die äußeren Eigenschaften einer Anwendung frühzeitig auszuprobieren, da die Implementierung erst sehr spät erfolgt.

In der Phase des Entwurfs ist eine Interaktion mit dem Endbenutzer mangels Prototypen äußerst schwierig. Das bedeutet, dass der Benutzer eine tatsächliche Vorstellung von dem Produkt erst bei seiner Fertigstellung bekommt.[70]

Die starre Phasen-Aufteilung des Wasserfallmodells kann außerdem dazu führen, dass zu einem frühen Zeitpunkt Verpflichtungen eingegangen werden, die es erschweren, sich auf neue Anforderungen einzustellen. Dadurch kann ein fester Auslieferungstermin womöglich nicht eingehalten werden.[71]

Die fortlaufende Dokumentation birgt die Gefahr, dass die Dokumentation einen größeren Stellenwert einnimmt als das eigentliche System.[72]

3.2 V-Modell

Dieses Vorgehensmodell wird in Form eines „V“ dargestellt. Es stellt eine Weiterentwicklung des Wasserfallmodells unter besonderer Berücksichtigung von Qualitätssicherungsaktivitäten dar. Dies erfolgt durch Maßnahmen zur Verifikation und Validierung.[73]

68 Vgl. Chroust, Gerhard: Modelle der Software-Entwicklung, a. a. O., S. 161.

69 Vgl. Sommerville, Ian: Software Engineering, a. a. O., S. 98 und Goll, Joachim: Methoden und Architekturen der Softwaretechnik, a. a. O., S. 85.

70 Vgl. Chroust, Gerhard: Modelle der Software-Entwicklung, a. a. O., S. 161.

71 Vgl. Sommerville, Ian: Software Engineering, a. a. O., S. 98 und Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 6.

72 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 101.

73 Vgl. Abts, Dietmar; Mülder, Wilhelm: Grundkurs Wirtschaftsinformatik, a. a. O., S. 429 und Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 101.

Entstehung

Die erste Erwähnung fand das V-Modell in einer Arbeit von Barry Boehm im Jahre 1979.[74] In dieser Publikation zur Qualitätssicherung bei der Entwicklung von Software und Systemen stellte Boehm dem Wasserfallmodell ein V-förmiges Phasenmodell gegenüber, das gleichzeitig als Namensgeber für das V-Modell diente. Um die Qualität bei der Entwicklung zu verbessern, wurden Maßnahmen zur Verifikation und Validierung eingeführt.[75] Mit diesem V-Modell als Basis wurde ein Vorgehensmodell für die Bundeswehr und im Anschluss für Behörden entwickelt. Dieses diente der Standardisierung der Software-Bearbeitung im Bereich der Bundesverwaltung und wurde vom Bundesinnenminister im Bundesanzeiger veröffentlicht. Inzwischen wird dieser Standard auch in der Industrie angewendet.[76] Innerhalb der folgenden Jahre wurde das V-Modell stetig fortgeschrieben. Dies endete mit der Fertigstellung des V-Modells 97 im Jahr 1997. Neuere Entwicklungen in Methodik und Technologie waren im V-Modell 97 unzureichend berücksichtigt und das Modell spiegelte den aktuellen Stand in der Softwaretechnik nicht wider. Projekte, die mit dem V-Modell 97 durchgeführt wurden, konnten somit den Stand der Technik nicht in dem Maße nutzen, wie es wünschenswert gewesen wäre.[77] Im Jahre 2004 wurde als Ergebnis einer weiteren Revision das V-Modell XT vorgestellt. XT steht in diesem Kontext für „extreme tailoring“ („extremes Maßschneidern“) und drückt die angestrebte Flexibilität des Modells aus.[78] Das V-Modell XT stellt die neueste Version des V-Modells dar und wird fortwährend aktualisiert.

Zentrale Charakteristika und Ablauf

Grundsätzlich wird beim V-Modell davon ausgegangen, dass Qualitätssicherungsmaßnahmen aus den Querbezügen zwischen den Ergebnissen der Integrations- und Konkretisierungsschritte abgeleitet werden können. Bei der Erstellung des Softwaresystems ist festzustellen, ob das erzielte Produkt den gestellten Anforderungen entspricht (Verifika-

74 Vgl. Boehm, Barry W.: Guidelines for verifying and validating software requirements and design specifications, EURO IFIP, North-Holland 1979, S. 711–719.

75 Vgl. Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 37 und Bartelt, Christian; Bauer, Otto; Beneken, Gerd; et al.: V-Modell XT. Das deutsche Referenzmodell für Systementwicklungsprojekte. Version 2.0, Hrsg.: Verein zur Weiterentwicklung des V-Modell XT e.V., München: 2006, S. 7.

76 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 103.

77 Vgl. Höhn, Reinhard; Höppner, Stephan; Rausch, Andreas: Das V-Modell XT. Anwendungen, Werkzeuge, Standards, Berlin, Heidelberg: Springer 2008, S. 3.

78 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 190.

tion: Hat der Auftragnehmer das System richtig entwickelt?). Zum anderen ist bei der Nutzung des Systems zu prüfen, ob das erstellte Produkt für die vorgesehene Aufgabe geeignet ist (Validierung: Haben wir das richtige System in Auftrag gegeben?). Konstruktive und analytische Tätigkeiten werden zueinander in Beziehung gesetzt (siehe Abbildung 7). Im linken Bereich werden Tätigkeiten, die zu einer schrittweisen Konkretisierung des Anwendungssystems in Teilsystemen führen, dargestellt. Der rechte Bereich repräsentiert Tätigkeiten, die zu einer schrittweisen Integration des Anwendungssystems aus Komponenten führen.[79]

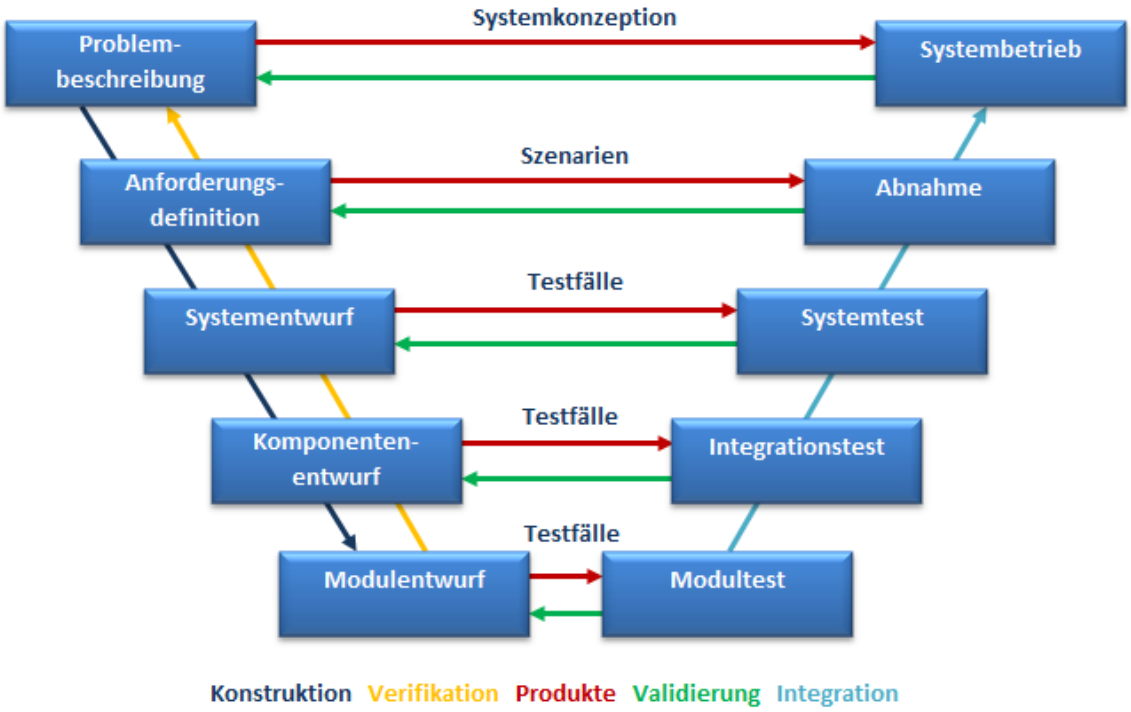

Abbildung 7: Ablauf des V-Modells 97[80]

Das V-Modell XT ist in vier Submodelle gegliedert. Dadurch ist es möglich, neben den Tätigkeiten zur Erstellung des IT-Systems auch begleitende Tätigkeiten wie Qualitätssicherung, Konfigurationsmanagement und das Projektmanagement zu betrachten. Die vier Submodelle sind eng miteinander verbunden und beeinflussen sich gegenseitig.[81]

79 Vgl. Bremer, Georg: Genealogie von Entwicklungsschemata, a. a. O., S. 41 und Friedrich, Jan; Kuhrmann, Marco; Sihling, Marc; Hammerschall, Ulrike: Das V-Modell XT. Für Projektleiter und QS-Verantwortliche kompakt und übersichtlich, Berlin: Springer 2009, S. 107 f.

80 In Anlehnung an Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 101.

81 Vgl. Dröschel, Wolfgang: Einführung in das V-Modell, in: Das V-Modell 97. Der Standard für die Entwicklung von IT-Systemen mit Anleitung für den Praxiseinsatz, Hrsg.: Dröschel, Wolfgang, München: Oldenbourg 2000, S. 8.

Das V-Modell XT legt im Detail fest, wer was wann in einem Entwicklungsprojekt zu tun hat. Hierfür werden folgende Projekttypen unterschieden:

- Systementwicklungsprojekt eines Auftraggebers (AG)
- Systementwicklungsprojekt eines Auftragnehmers (AN)
- Systementwicklungsprojekt eines Auftraggebers mit Auftragnehmer in der gleichen Organisation (ohne Vertrag)
- Einführung und Pflege eines organisationsspezifischen Vorgehensmodells

In der neuesten Version XT wird im Gegensatz zum V-Modell 97 das Produkt stärker in eine zentrale Position gerückt. Wie andere Vorgehensmodellen auch, beschreibt das V-Modell XT die Abläufe im Verlauf des Entwicklungsprojekts über Produkte, Rollen und Aktivitäten. Die Besonderheit liegt hier bei den Vorgehensbausteinen, die eine Modularisierung der Abläufe und somit eine flexible Zusammenstellung je nach Projekttyp erlauben.[82] Die Projektdurchführungsstrategie legt fest, welche Bausteine vorkommen müssen bzw. können und in welcher Reihenfolge sie abgearbeitet werden sollen.[83]

Das V-Modell XT soll für unterschiedliche Produktentwicklungen einsetzbar sein. Dazu ist eine Anpassung an konkrete Entwicklungen erforderlich. Dies geschieht durch das sogenannte Tailoring („Maßschneidern“).[84] Zu Projektbeginn sind nicht alle Vorgehensbausteine notwendig. Durch das Tailoring werden nur die Bausteine ausgewählt, die auch tatsächlich gebraucht werden.[85] Auf diese Weise ist es möglich, Aktivitäten, Rollen und Produkte herauszufiltern und zu einem schlanken Vorgehen im Projekt beizutragen.

Einsatzbereiche

Das V-Modell ist geeignet für größere Entwicklungsprojekte und enthält ein umfangreiches Regelwerk. Bei kleineren Projekten kann hierdurch ein gewisses Maß an Bürokratie anfallen. Ursprünglich wurde das V-Modell für IT-Projekte des Bundes und von Behörden erarbeitet. Sollte der Auftraggeber ein Bundesministerium oder eine ähnliche staatliche Institution sein, gilt das V-Modell als verbindlich anzuwenden. Daher wird es

82 Vgl. Friedrich, Jan; Kuhrmann, Marco; Sihling, Marc; Hammerschall, Ulrike: Das V-Modell XT. Für Projektleiter und QS-Verantwortliche kompakt und übersichtlich, a. a. O., S. 3.

83 Vgl Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, a. a. O., S. 355.

84 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 109.

85 Vgl. Friedrich, Jan; Kuhrmann, Marco; Sihling, Marc; Hammerschall, Ulrike: Das V-Modell XT. Für Projektleiter und QS-Verantwortliche kompakt und übersichtlich, a. a. O., S. 6.

auch als das Vorgehensmodell des Bundes bezeichnet.[86] Inzwischen wird es jedoch auch in der Wirtschaft eingesetzt.[87] Das V-Modell XT ist ausgerichtet auf die Planung und Entwicklung von IT-Systemen. Die Organisation und Durchführung des laufenden Betriebs, die Instandhaltung, sowie die Aussonderung von Systemen wird nicht abgedeckt. Die Konzeption dieser Aufgaben im Rahmen der Entwicklung ist jedoch im V-Modell XT geregelt.[88]

Vor- und Nachteile

Die Vorteile des V-Modells liegen in der integrierten und detaillierten Darstellung von Systemerstellung, Qualitätssicherung, Konfigurations- und Projektmanagement. Es ist gut geeignet für große Projekte, insbesondere für eingebettete Systeme.[89]

Das Modell ist generisch und lässt sich unternehmens- und projektspezifisch anpassen. Im Sinne eines Baukastens ist das V-Modell XT ein sinnvoller Ausgangspunkt bei der Erstellung eines unternehmensspezifischen Vorgehensmodells. Das Risiko, wichtige Komponenten außer Acht zu lassen, wird minimiert.

Eine Verwendung des V-Modells ohne erhebliche Anpassungen macht es aufgrund seiner Größe schwerfällig in der Anwendung. Die Menge der zu erstellenden Produkte bzw. Dokumente ist auch nach einem Tailoring für ein kleines Projekt erheblich. Daher muss die Wirtschaftlichkeit vieler Dokumente bezweifelt werden.[90]

Des Weiteren besteht die Gefahr, dass Vorgehenskonzepte, die für große eingebettete Systeme sinnvoll sind, unkritisch auf andere Anwendungstypen übertragen werden.

Ohne eine entsprechende Case-Unterstützung[91] ist das V-Modell kaum handhabbar.[92]

---

86 Vgl. Brandt-Pook, Hans; Kollmeier, Rainer: Softwareentwicklung kompakt und verständlich. Wie Softwaresysteme entstehen, 2. Aufl., Wiesbaden: Springer Vieweg 2015, S. 25.

87 Vgl. Abts, Dietmar; Mülder, Wilhelm: Grundkurs Wirtschaftsinformatik, a. a. O., S. 430.

88 Vgl. Bartelt, Christian; Bauer, Otto; Beneken, Gerd; et al.: V-Modell XT. Das deutsche Referenzmodell für Systementwicklungsprojekte. Version 2.0, a. a. O., S. 7.

89 VglBalzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 113.

90 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 201.

91 Das Akronym Case steht für „Computer Aided (oder Assisted) Software Engineering“ und drückt aus, dass die Entwicklung von Software mithilfe computergestützter Software-Entwicklungswerkzeuge oder Umgebungen erfolgt. Für weitere Informationen siehe: https://www.gi.de/service/informatiklexikon/detailansicht/article/case-computer-aided-software-engineering.html.

3.3 Nebenläufige Modelle

Nebenläufige Vorgehensmodelle (auch parallele Vorgehensmodelle genannt) ermöglichen eine beschleunigte Bearbeitung von anfallenden Arbeiten, indem die Überlappung einzelner Phasen genutzt wird. Die einzelnen Phasen werden parallel abgearbeitet. Zu den wichtigsten Vertretern dieser Klasse zählen das „Simultaneous Engineering“ (SE) und das „Concurrent Engineering“ (CE).[93]

Entstehung

Die nebenläufigen Modelle haben ihren Ursprung in der Fertigungsindustrie, die früher stark zwischen der Entwicklung und Fertigung von Produkten trennte. Eine Überprüfung hinsichtlich Qualität, Fertigungstauglichkeit und Wartbarkeit war erst nach der Entwicklung eines Prototyps möglich. Oftmals führte dies zu einer Produktüberarbeitung mit entsprechenden Zeitverzögerungen. Um die termingerechte Fertigstellung zu gewährleisten, wurden nebenläufige Modelle eingeführt. Alle an einer Entwicklung beteiligten Abteilungen einschließlich der Fertigung, des Marketings und des Vertriebs wurden in einem Team vereint, um von Anfang an alle relevanten Gesichtspunkte berücksichtigen zu können. Durch diese Parallelisierung konnten kostspielige Überarbeitungen eingespart werden. Bei der Entwicklung vom Wasserfallmodell zu den evolutionären Vorgehensmodellen stellt das nebenläufige Modell den logischen nächsten Schritt dar.[94]

Zentrale Charakteristika und Ablauf

Die parallele Abarbeitung kann entweder nach dem SE oder nach dem CE erfolgen. Das SE zeichnet sich dadurch aus, dass unterschiedliche Aktivitäten überlappend und parallel ausgeführt werden. Diese Aktivitäten wurden ursprünglich rein sequenziell abgearbeitet (siehe Abbildung 8).[95] In der Software-Entwicklung kann dieses Konzept wie folgt aussehen: In der Definitionsphase werden nach der Erstellung des Pflichtenheftes parallel das OOA-Modell[96], die Benutzungsoberfläche und das Benutzerhandbuch ent-

92 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 113.

93 Vgl. Timinger, Holger: Wiley Schnellkurs Projektmanagement, Hoboken: Wiley 2015, S. 59.

94 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 127 und Gnatz, Michael: Vom Vorgehensmodell zum Projektplan, a. a. O., S. 23.

95 Vgl. Neutschel, Bernd; Vajna, Sandor: Organisations- und Prozessintegration, in: Integrated Design Engineering, Hrsg.: Vajna, Sandor, Berlin: Springer Vieweg 2014, S. 351.

96 Das objektorientierte Analysemodell ist eine Variante des Analyseprozesses. Es geht dabei um die Frage was ein System leisten soll.

wickelt. Ist schon ein Teil des OOA-Modells entwickelt, beginnt bereits der Entwurf. Ausgehend von diesem Entwurf wird anschließend mit der Implementierung begonnen.[97] Beim CE werden einzelne Aufgaben auf mehrere Personen aufgeteilt und parallel abgearbeitet.

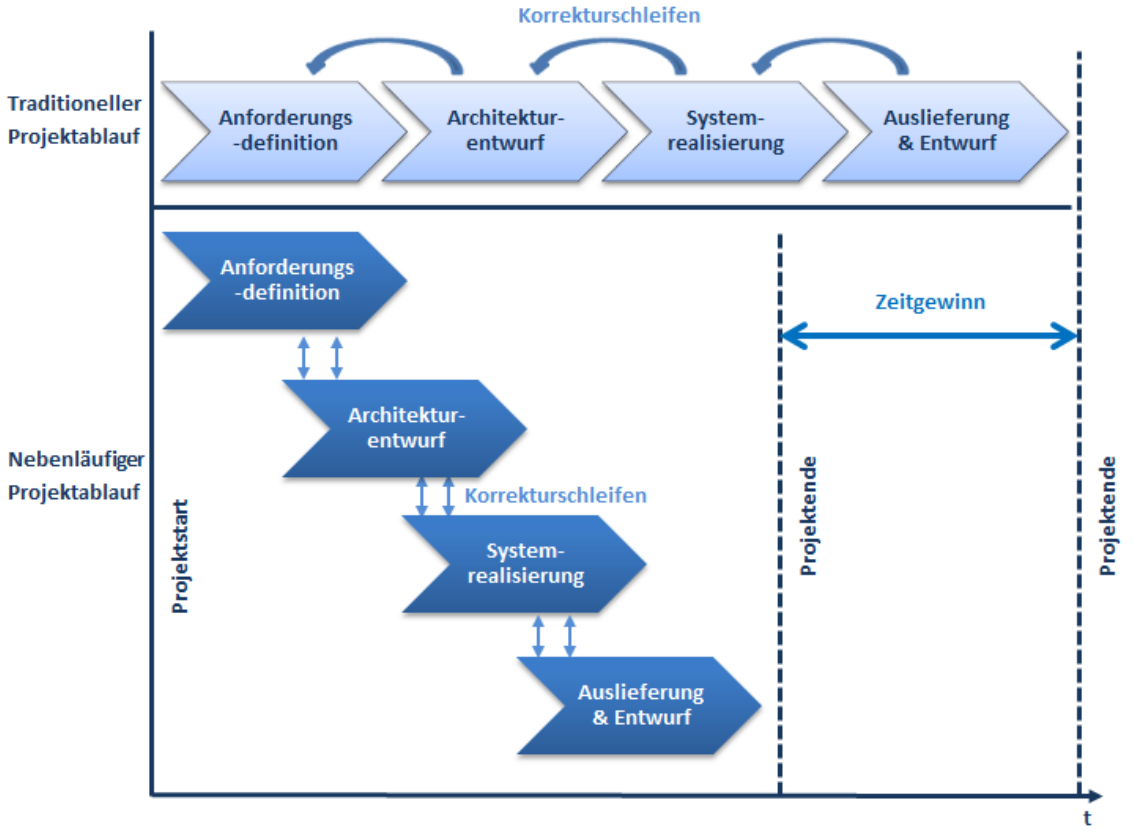

Abbildung 8: Nebenläufige Entwicklung[98]

Einsatzbereiche

Der Einsatz eines nebenläufigen Vorgehensmodells ist nicht bei allen Arten von Projekten zu empfehlen. Ist ein schnell sichtbares Ergebnis gewünscht, lohnt sich allerdings der Einsatz von nebenläufigen Modellen, insbesondere wenn Funktionen, die im Rahmen der Terminvorgabe nicht zu realisieren sind, ohne dass größere Probleme auf eine Folgeversion des betreffenden Produkts verschoben werden können.

Bei schwankenden und unberechenbaren Kundenanforderungen sowie bei unerfahrenen Anwendern ist ein Einsatz des SE zielführend, da die Anwender direkt in die Entwick-

97 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 127.

98 In Anlehnung an Neutschel, Bernd; Vajna, Sandor: Organisations- und Prozessintegration, a. a. O., S. 351.

lung miteinbezogen werden können. Bleiben die externen Rahmenbedingungen dagegen stabil, ist ein systematisches Vorgehen zweckmäßig. Je höher der Bedarf an Investitions- und Ressourcenausstattung eines Projekts ist und je genauer dieser Bedarf abgeschätzt werden kann, desto eher ist die Verwendung eines phasenorientierten, streng sequenziellen Vorgehensmodells zu empfehlen.[99]

Vor- und Nachteile

Die Vorteile der nebenläufigen Vorgehensmodelle liegen in der Chance auf Zeitgewinn durch die parallele Bearbeitung der einzelnen Phasen. Im Gegensatz zu rein sequenziellen Vorgehensmodellen ermöglichen nebenläufige Modelle eine frühzeitigere Rückmeldung über mögliche Probleme in späteren Phasen durch Vorwegnahme der dortigen Bearbeitung.

Durch die Beteiligung aller betroffenen Personengruppen ist eine frühe Erkennung und Vermeidung von Risiken sowie eine optimale Zeitausnutzung gegeben.

Allerdings steigen auch die Risiken eines Mehraufwands, falls parallel begonnene Arbeiten späterer Phasen korrigiert werden müssen, weil sich in früheren Phasen Änderungen ergeben haben. Dies kann auch zur Überlastung der Mitarbeiter führen.[100]

Iterationen können notwendig werden, wenn die grundlegenden und kritischen Entscheidungen zu spät getroffen werden.

Um mögliche Probleme tatsächlich frühzeitig antizipieren zu können, ist ein hoher Planungs- und Personalaufwand notwendig.[101]

99 Vgl. Seibert, Siegfried: Technisches Management. Innovationsmanagement, Projektmanagement, Qualitätsmanagement, Stuttgart: Teubner 1998, S. 311.

100 Vgl. Timinger, Holger: Wiley Schnellkurs Projektmanagement, a. a. O., S. 66.

101 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 129.

4 Evolutionäre Modelle

4.1 Spiralmodell

Beim Spiralmodell durchläuft der Software-Entwicklungsprozess einen aus vier Schritten bestehenden Zyklus mehrfach. Das Ziel ist es, Risiken frühzeitig zu erkennen und zu vermeiden. Dabei kann das Spiralmodell als eine Art Vorgehensmodell-Generator bezeichnet werden, da pro Zyklus ein oder mehrere unterschiedliche Vorgehensmodelle zum Einsatz kommen können. Sind die Anforderungen im Vorfeld bekannt und das Risiko gering, nähert sich das Modell beispielsweise dem Wasserfallmodell an. Bei unbekannten Anforderungen und Risiken wird verstärkt inkrementell entwickelt bzw. Prototyping angewendet.[102]

Entstehung

Sequenzielle Vorgehensmodelle wie das Wasserfallmodell standen vor dem Problem, dass die früheren Phasen auf Vertrauensbasis beendet werden mussten. Bei der Erstellung eines Fachkonzepts wurde beispielsweise ohne weitere Überprüfung davon ausgegangen, dass ein Entwurf erarbeitet worden ist. In der Entwurfsphase musste anschließend z. B. darauf vertraut werden, dass der Entwurf auch tatsächlich implementierbar war. Solange die zu entwickelnden Anwendungen gut verstanden wurden, war ein solches Vorgehen unproblematisch. Bei unsicheren Projekten, die nicht vollständig verstanden wurden, konnte man sich jedoch kaum darauf verlassen, dass die antizipierte Anwendung realisierbar bzw. profitabel war.[103] Daher wurde es in den 1980er-Jahren nötig, Vorgehensmodelle zu entwickeln, die eine aufeinander aufbauende Software-Entwicklung, in Form von mehreren Versionen, ermöglichte. Das Wasserfallmodell war unzureichend, wenn Software in Stücken (Inkrementen) entwickelt werden musste. Dies war immer dann nötig, wenn die Anforderungen an ein Projekt nicht bis ins Detail geklärt waren bzw. sich im Laufe des Entwicklungsprozesses veränderten.[104] Auf die anhaltenden Diskussionen über das Wasserfallmodell reagierte Barry Boehm 1986 bzw. 1988 mit der Veröffentlichung des Spiralmodells. Eigentlich handelte es sich hierbei

102 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 134 und Goll, Joachim: Methoden und Architekturen der Softwaretechnik, a. a. O., S. 124.

103 Vgl. Chroust, Gerhard: Modelle der Software-Entwicklung, a. a. O., S. 169 f.

104 Vgl. Hanser, Eckhart: Agile Prozesse: Von XP über Scrum bis MAP, Berlin, Heidelberg: Springer 2010, S. 3 f.

nicht um ein konkretes, sondern um ein generisches Vorgehensmodell, das eine Anleitung zur konkreten Ausprägung eines Vorgehensmodells bereitstellte.[105]

Zentrale Charakteristika und Ablauf

Vorgehensmodelle mit geplanten Rücksprüngen multiplizieren im Prinzip die einzelnen Phasen klassischer Wasserfallmodelle. Wie ein phasenweises Vorgehen nicht sequenziell, sondern rekursiv aufgebaut sein kann und dabei einen Prototyp schrittweise entwickelt, zeigt das von Boehm entwickelte Spiralmodell.[106] Grundsätzlich wird davon ausgegangen, dass die Anwendungsentwicklung sowohl aus der Erstellung als auch aus der Weiterentwicklung eines Softwaresystems besteht. Dieses Vorgehen wird in Form einer Spirale dargestellt.[107] Dabei besteht jede Spiralrunde aus den folgenden vier Schritten:[108]

1. *Ziele aufstellen*: Es werden spezielle Ziele für diese Phase definiert. Weiterhin werden Rahmenbedingungen, denen der Prozess bzw. das Produkt unterliegt, bestimmt und es wird ein Managementplan erstellt. Eine Aufstellung der Projektrisiken, anhand derer alternative Strategien geplant werden können, bildet den Abschluss dieses Schrittes.

2. *Risiken einschätzen und verringern*: Jedes erkannte Projektrisiko wird intensiv analysiert und es werden Schritte zur Risikoreduktion unternommen. Wenn beispielsweise das Risiko darin besteht, dass die Anforderungen nicht angemessen sind, kann ein Prototyp des Systems entwickelt werden.

3. *Entwicklung und Validierung*: Nach Auswertung der Risiken wird ein Entwicklungsmodell ausgewählt. Liegen die Risiken beispielsweise hauptsächlich bei der Benutzeroberfläche, ist die evolutionäre Herstellung eines Prototyps ein angemessenes Entwicklungsmodell. Bei Sicherheitsrisiken ist dagegen eine Entwicklung mithilfe formaler Umformungen geeignet. Das Wasserfallmodell ist angebracht, wenn das größte Risiko bei der Integration von Subsystemen liegt.

105 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 176.

106 Vgl. Schwickert, Axel C.: Web Site Engineering – Modelltheoretische und methodische Erfahrungen aus der Praxis, in: Arbeitspapiere WI, Nr. 8/1997, Hrsg.: Lehrstuhl für Allg. BWL und Wirtschaftsinformatik, Johannes Gutenberg-Universität: Mainz 1997, S. 8.

107 Vgl. Bremer, Georg: Genealogie von Entwicklungsschemata, a. a. O., S. 43.

108 Vgl. Abts, Dietmar; Mülder, Wilhelm: Grundkurs Wirtschaftsinformatik, a. a. O., S. 430.

4. *Planung*: Es folgt die Überprüfung des Projekts. Es wird entschieden, ob mit der nächsten Windung der Spirale fortgefahren werden kann. Wird sich dafür entschieden, werden die Pläne für die nächste Phase aufgestellt.[109]

Das mehrfache Durchlaufen dieser Schritte stellt den Risikobezug des Modells („risk-driven-model“)[110] dar. Die Fläche der Spirale repräsentiert die akkumulierten Kosten, die bisher bei der Entwicklung angefallen sind. Der Winkel der Spirale zeigt den Fortschritt der Entwicklung des jeweiligen Zyklus an und verbindet gleichartige Tätigkeiten in mehreren aufeinanderfolgenden Durchläufen der Spirale (siehe Abbildung 9).[111]

Der Risikobezug ist fundamental für das Spiralmodell. Es soll sicherstellen, dass Risiken erkannt und möglichst früh im Projekt umgangen werden. Daraus leitet Boehm ein einfaches Vorgehen ab. Folgender Zyklus soll entweder bis zum erfolgreichen Abschluss oder bis zum Scheitern eines Projektes wiederholt werden:

1. Suche nach allen Risiken, von denen das Projekt bedroht ist. Sind keine Risiken mehr vorhanden, ist das Projekt erfolgreich abgeschlossen.
2. Bewertung der erkannten Risiken, um das größte Risiko zu identifizieren.
3. Suche nach Möglichkeiten, um das größte Risiko zu eliminieren. Sollte sich das größte Risiko nicht beseitigen lassen, ist das Projekt gescheitert.[112]

Einsatzbereiche

Das Spiralmodell ist ein offenes System mit verschiedenen Ausprägungen in der Anwendung. Durch seine Flexibilität lässt es sich an unterschiedliche Projektsituationen anpassen.[113] Da der Fokus des Spiralmodells auf der Ziel-Definition, der Risiko-Analyse und den Planungsaktivitäten liegt, ist es besonders für Großprojekte der Software-Entwicklung geeignet.[114] Insbesondere bei unklaren oder instabilen Anforderungen an das zu entwickelnde System empfiehlt sich der Einsatz eines Spiralmodells. Zu Beginn

109 Vgl. Sommerville, Ian: Software Engineering, a. a. O., S. 103.

110 Vgl. Boehm, Barry: A Spiral Model of Software Development and Enhancement. In: IEEE Computer, Vol.21, Ausg. 5, Mai 1988, pp 61–72.

111 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 131 und Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 73 f.

112 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 178.

113 Vgl. Gnatz, Michael: Vom Vorgehensmodell zum Projektplan, a. a. O., S. 24.

114 Vgl. Informatik Forum Simon (Hrsg.): Spiralmodell, Online im Internet: http://www.infforum.de/themen/anwendungsentwicklung/se-spiral-modell.htm, 11.01.2016.

wird der klare und stabile Teil der Anforderungen betrachtet. Während der Entwicklung des ersten Inkrements können sich weitere Teile der Anforderungen, die in folgenden Inkrementen realisiert werden können, festigen.[115]

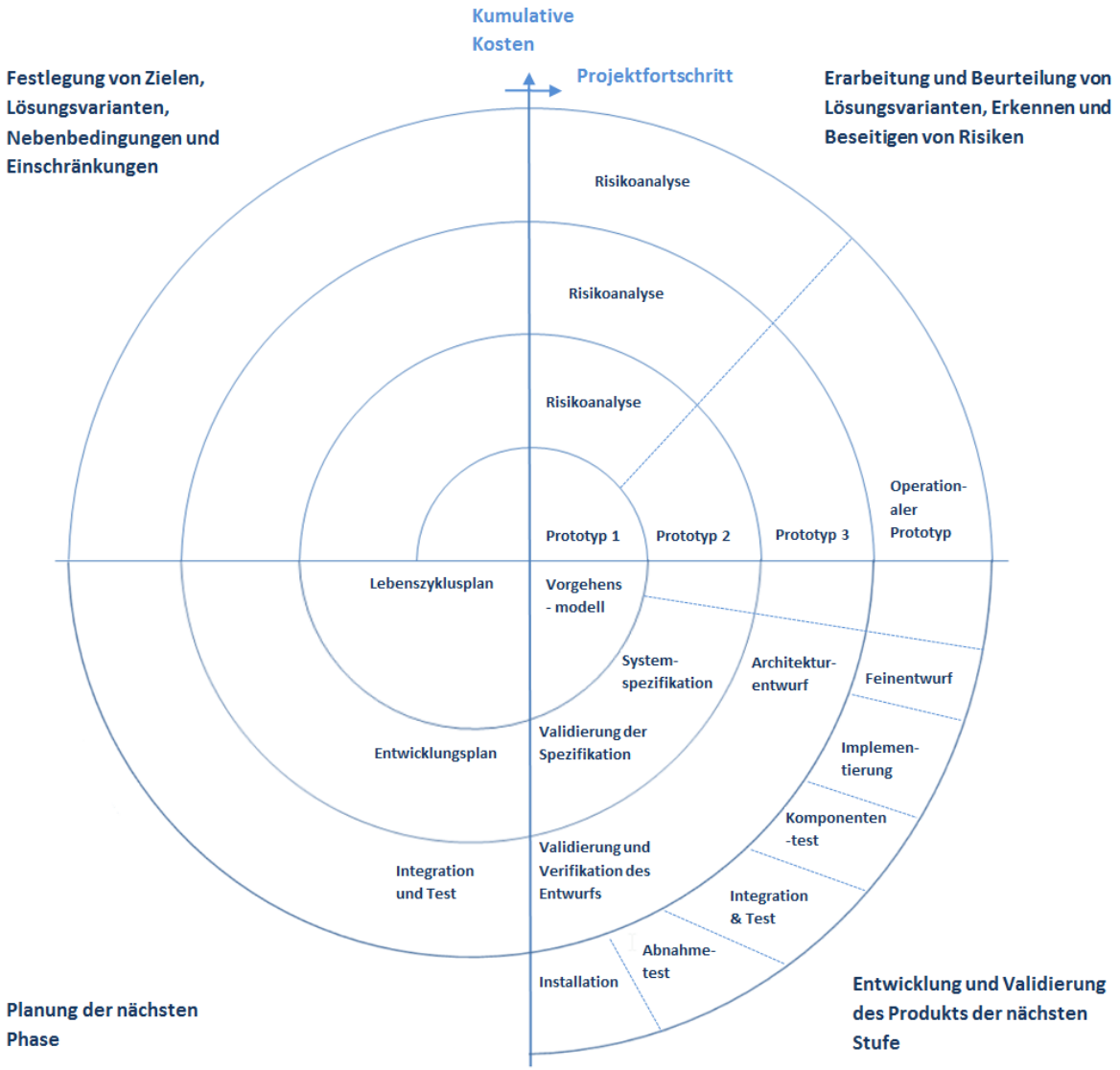

Abbildung 9: Ablauf des Spiralmodells[116]

Vor- und Nachteile

Ein wichtiger Vorteil dieses Vorgehensmodells ist, dass auf der einen Seite Entwicklungsphasen wiederholt (iterativ) mit einem höheren Wissensstand durchlaufen werden

115 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 14.

116 In Anlehnung an Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, a. a. O., S. 354.

können. Gleichzeitig können allerdings auch geänderte Anforderungen beim wiederholten Durchlauf berücksichtigt werden.[117]

Weitere wesentliche Vorteile des Spiralmodells sind das frühe Erkennen von Fehlern bzw. Risiken und das Abwägen von Lösungsalternativen. Für die Endnutzer ist es durch die konsequente Orientierung am Prototyping möglich, bereits in frühen Phasen in den Entwicklungsprozess eingebunden zu werden.[118]

Die erforderliche Erfahrung bei der Risikobewertung kann als Nachteil angesehen werden, da das entsprechende Know-how der Projektverantwortlichen als Basis für den Erfolg bei der Anwendung eines Spiralmodells dient.[119]

Ebenfalls führen die oft neuen Entscheidungen über den weiteren Prozessablauf zu einem hohen Managementaufwand. Dieser macht das Modell für kleine und mittlere Projekte eher ungeeignet.[120]

4.2 Rational Unified Process

Beim Rational Unified Process (RUP) handelt es sich um ein umfassendes, iteratives und inkrementelles Vorgehensmodell für Software-Entwicklungsprozesse. Die Software-Architektur steht im Mittelpunkt und der RUP geht von einer modellgetriebenen Software-Entwicklung aus. Dabei orientiert er sich eng an der Modellierungssprache UML (Unified modeling language).[121] Der RUP trennt Phasen und Arbeitsabläufe, wobei Phasen dynamisch sind und Ziele vorgeben. Arbeitsabläufe sind dagegen statisch und können als technische Aktivitäten angesehen werden, die nicht mit einer einzelnen Phase verknüpft sind, sondern in der Entwicklung dazu verwendet werden, die Ziele der einzelnen Phasen zu erreichen.[122]

---

117 Vgl. Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, a. a. O., S. 353.

118 Vgl. Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, a. a. O., S. 354 und Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 75.

119 Vgl. Pressman, Roger: Software engineering. A practitioner's approach. 5th ed., Boston: Mass: McGraw Hill 2001, S. 38.

120 Vgl. Balzert, Helmut; Ebert, Christof: Lehrbuch der Softwaretechnik. Softwaremanagement, a. a. O., S. 133.

121 Vgl. Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, a. a. O., S. 356.

122 Vgl. Sommerville, Ian: Software Engineering, a. a. O., S. 115.

Entstehung

Der RUP ist ein kommerziell vermarktetes Vorgehensmodell für die objektorientierte Entwicklung und wurde 1998 von der Firma Rational Software vorgestellt. Der Architekt des RUP war Philipp Kruchten.[123] Der RUP ging aus zwei Ansätzen hervor: Zum einen aus dem von der Firma Objectory AB entwickelten objektorientierten Ansatz mit Use-Cases, zum anderen aus dem methodischen Ansatz der Firma Rational.[124] Nach der Übernahme von Objectory durch Rational wurden beide Ansätze 1996 zum Rational Objectory Process zusammengeführt und 1998 zum RUP weiterentwickelt.[125] Mittlerweile gehört Rational Software zu IBM. Der RUP soll die Vorteile konventioneller Phasenmodelle und nichtlinearer Vorgehensmodelle vereinen. Dabei erleichtern Phasen die Planung und das Management von Projekten. Iterationen und inkrementelles Entwickeln identifizieren Risiken frühzeitig und helfen, sie zu beseitigen.[126]

Zentrale Charakteristika und Ablauf

Prinzipiell besteht der RUP aus vier Phasen mit beliebig vielen Iterationen.[127] Abbildung 10 zeigt auf der x-Achse die Abfolge der Phasen Inception, Elaboration, Construction und Transition.

Die einzelnen Phasen lassen sich nach Sommerville wie folgt beschreiben:

1. *Konzeption (Inception)*: In dieser Phase erfolgt der Einstieg in das Projekt. Es wird ein Geschäftsfall für das System ausgearbeitet. Alle externen Einheiten, die mit dem System zusammenwirken, müssen festgelegt werden. Anschließend wird die Geschäftsrelevanz des Systems beurteilt. Fällt diese gering aus, kann das Projekt annulliert werden.
2. *Entwurf (Elaboration)*: In der Entwurfsphase wird versucht, ein Verständnis für den Problembereich zu entwickeln. Des Weiteren wird ein Architekturrahmen für das System festgelegt und ein Projektplan erstellt. Dadurch lassen sich die größten Risiken des Projekts ermitteln.

123 Vgl. Goll, Joachim: Methoden und Architekturen der Softwaretechnik, a. a. O., S. 101.

124 Vgl. Kruchten, Philippe: The rational unified process. An introduction, 2. ed., Boston: Addison-Wesley 2003, S. 32.

125 Vgl. Noack, Jörg; Schienmann, Bruno: Objektorientierte Vorgehensmodelle im Vergleich, in: Informatik-Spektrum, 3/1999, S. 168.

126 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 203.

127 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 57.

3. Konstruktion (Construction): Diese Phase umfasst den Systementwurf, die Programmierung sowie Tests. Teile des Systems werden parallel entwickelt und integriert. Nach Abschluss dieser Phase sollte ein funktionierendes Softwaresystem mit entsprechender Dokumentation vorliegen, das an den Benutzer ausgeliefert werden kann.
4. Übergabe (Transition): In der letzten Phase erfolgt die Übergabe des Systems. Nach Abschluss sollte der Auftraggeber über ein Softwaresystem mit entsprechender Dokumentation verfügen, das in seiner realen Arbeitsumgebung ordnungsgemäß funktioniert.[128]

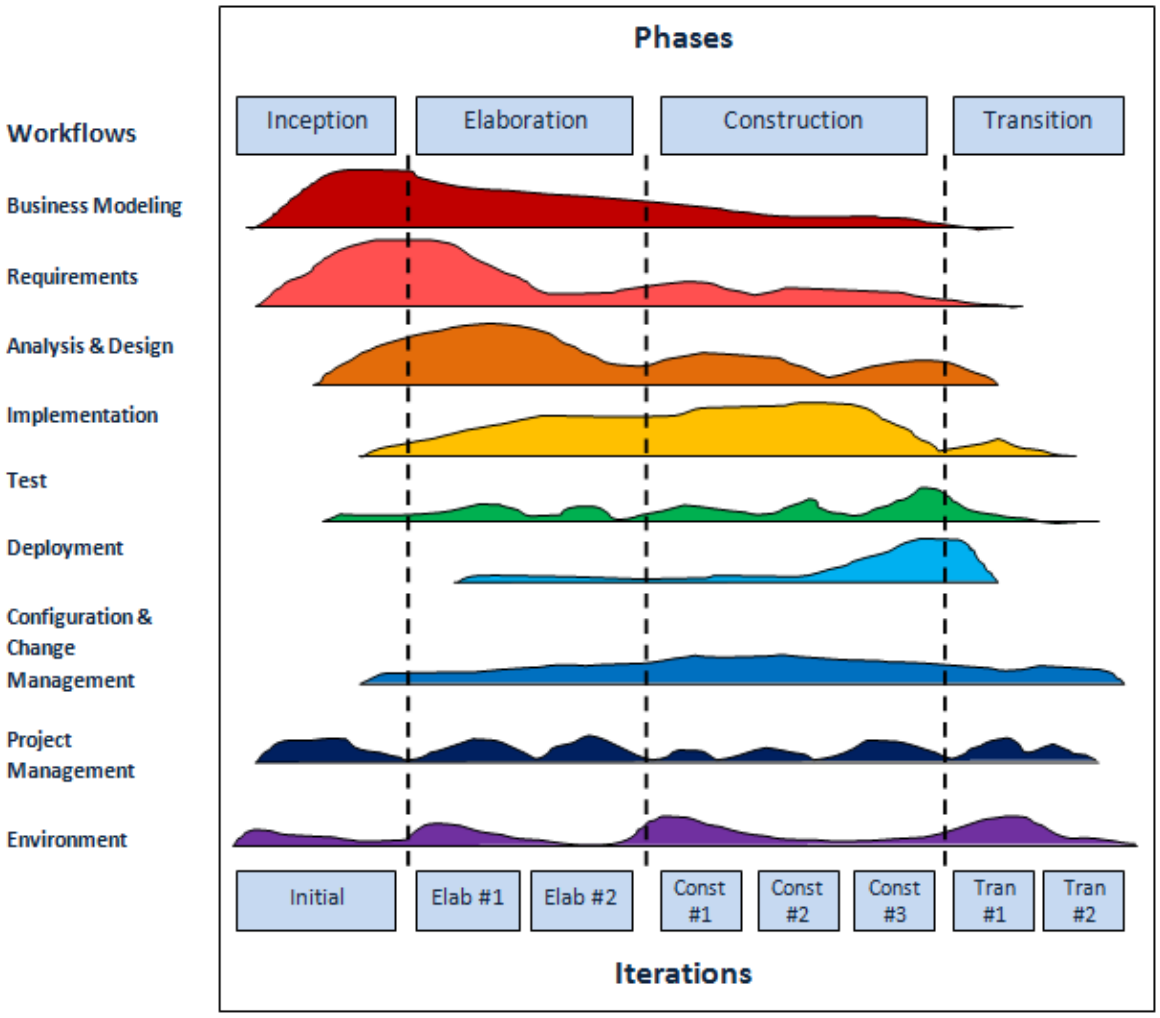

Abbildung 10: Darstellung des RUP[129]

128 Vgl. Sommerville, Ian: Software Engineering, a. a. O., S. 113.

129 In Anlehnung an Kruchten, Philippe: The rational unifiedprocess, a. a. O., S. 46.

Jede dieser Phasen wird durch einen Meilenstein abgeschlossen. Sind noch nicht alle Aufgaben absolviert, wird eine weitere Iteration dieser Phase durchgeführt. Die y-Achse zeigt die sogenannten „Workflows“ (Kern-Arbeitsprozesse). Diese statischen Elemente stehen für die inhaltlichen Tätigkeiten als Artefakte, Aktivitäten und Rollen.[130]

Kernidee des RUP ist es, ein Produkt durch zeitlich beschränkte Iterationen weiterzuentwickeln. Dabei stellt das Ergebnis einer jeden Iteration eine inkrementelle Verbesserung des Produkts gegenüber der letzten Version dar. Das Produkt liegt also nach jeder Iteration in verbesserter Weise vor. Die einzelnen Prozessschritte werden nach zwei Prinzipien ausgestaltet: der Architekturzentriertheit und der Use-Case-Zentriertheit. Das Ziel besteht darin, schrittweise eine ausführbare Architektur zu entwickeln, die modellgetrieben definiert wird (entlang der x-Achse). Diese Modelle sind das zentrale Element zur Visualisierung der Software-Architektur. Zur Abstimmung und Integration der Abläufe werden Use-Cases (Anwendungsfälle) eingesetzt (entlang der y-Achse). Diese werden verwendet, um die Software-Architektur zu validieren, um das Vorgehen beim Test zu definieren und um Iterationen zu planen.[131]

Einsatzbereiche

Aufgrund der großen Anzahl an beschriebenen Rollen und Aktivitäten ist der Einsatz des RUP erst ab einer Teamgröße von ca. zehn Personen sinnvoll. Diese „Schwerfälligkeit“ macht ein entsprechendes Tailoring notwendig, um eine Ausrichtung auf die jeweiligen Projektziele zu gewährleisten und den Verwaltungsaufwand zu reduzieren.[132]

Vor- und Nachteile

Der RUP liegt in einer HTML-Fassung vor. Daher ist es für Anwender möglich, einfach durch den Prozess zu navigieren und Informationen zu finden. Jeder Mitarbeiter kann die Prozessbeschreibung direkt am Arbeitsplatz einsehen. Dadurch sind Mitarbeiter eher dazu geneigt, in der Prozessbeschreibung zu lesen und anfallende Probleme zu lösen.

130 Vgl. Höhn, Reinhard; Höppner, Stephan; Rausch, Andreas: Das V-Modell XT. Anwendungen, Werkzeuge, Standards, a. a. O., S. 279.

131 Vgl. Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, a. a. O., S. 356 f.

132 Vgl. Informatik Forum Simon (Hrsg.): Rational Unified Process (RUP), Online im Internet: http://www.infforum.de/themen/anwendungsentwicklung/se-rup.htm, 21.01.2016.

Die detaillierte Prozessbeschreibung bietet zu den Aktivitäten nicht nur eine Auflistung der Arbeitsschritte, sondern ebenfalls Arbeitsanleitungen. Die Arbeitsergebnisse werden inhaltlich beschrieben und enthalten jeweils ein Musterdokument, das die Struktur des zu erstellenden Dokuments aufweist. Dieser feine Detaillierungsgrad sorgt dafür, dass die Prozessbeschreibung meist ausreichend ausführliche Informationen enthält, um den Ablauf und die definierten Tätigkeiten zu verstehen.

Allerdings muss der RUP in der Regel an spezielle Gegebenheiten einer Entwicklungsorganisation angepasst werden. Dies betrifft insbesondere die Tätigkeiten der notwendigen Rollen und die geforderten Arbeitsergebnisse. Aufgrund der starken Vernetzung der Prozesselemente untereinander ist eine Anpassung nicht trivial und nur mit erheblichem Aufwand möglich.

Problematisch ist auch, dass die modellierten Arbeitsabläufe den Eindruck erwecken können, dass eine Durchführung der Tätigkeiten in der vorgeschriebenen Reihenfolge garantiert zu einem guten Ergebnis führt. Es genügt jedoch nicht, nur formal nach der Prozessvorgabe zu arbeiten. Es ist wichtig, dass die in den Aktivitäten erzielten Arbeitsergebnisse von ausreichender Qualität sind.[133]

4.3 Prototyping

Unter einem Prototyp wird laut Hansen eine demonstrierbare Vorabversion eines Programmsystems verstanden.[134] Mithilfe von Prototypen lassen sich wesentliche Merkmale einer Software frühzeitig untersuchen.[135] Der Begriff „Prototyping“ beschreibt eine Vorgehensweise der Software-Entwicklung, bei der Prototypen entworfen, konstruiert, bewertet und revidiert werden, um die Anforderungen von Kunden bzw. Nutzern rechtzeitig herauszufinden. Dadurch lässt sich eine lange und kostspielige Entwicklung vermeiden, die am Ende an den Bedürfnissen der Kunden/Nutzer vorbeiläuft.[136]

133 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 210 f.

134 Vgl. Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf: Wirtschaftsinformatik. Grundlagen und Anwendungen, a. a. O., S. 344.

135 Vgl. Chroust, Gerhard: Modelle der Software-Entwicklung, a. a. O., S. 164.

136 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 165.

Entstehung

Die Vorgehensweise des Prototypings wurde erstmals zu Beginn der 1980er-Jahre in der Literatur erwähnt.[137] Allerdings beinhalteten bereits die ersten sequenziellen Vorgehensmodelle (z. B. Wasserfallmodell Kap. 3) Ansätze zur Entwicklung von Prototypen.

Ende der 1970er-Jahre herrschte ein starker Bedarf an alternativen Entwicklungsstrategien, da insbesondere bei Großprojekten zunehmend Probleme zwischen den Anforderungen der Kunden/Nutzer und den entwickelten Endprodukten auftraten. Infolgedessen wurde 1979 eine amerikanische Studie in Auftrag gegeben, die zur Aufklärung dieser Problematik beitragen sollte. Es zeigte sich, dass bei der Verwendung herkömmlicher Vorgehensmodelle oftmals eine mangelnde Kommunikation zwischen Auftraggebern und Entwicklern die Ursache für das Scheitern von Software-Entwicklungsprojekten war.[138] Das wohl bekannteste Vorgehensmodell dieser Zeit war das Wasserfallmodell. Bei diesem Modell wurden die zukünftigen Nutzer lediglich zu Beginn der Anforderungsanalyse konsultiert und hinsichtlich ihrer Vorstellungen befragt. Vom weiteren Entwicklungsprozess wurden sie weitestgehend ausgeschlossen. Auftretende Fehler bzw. Unklarheiten, die zuvor in der Anforderungsanalyse entstanden waren, konnten sich auf diese Weise bis in das Endprodukt durchziehen, wodurch dieses im schlimmsten Fall unbrauchbar wurde. Eine Alternative, um diesen Schwachpunkt zu umgehen und die Kommunikation zwischen Auftraggebern und Entwicklern zu verbessern, bot das Prototyping.

Zentrale Charakteristika und Ablauf

Grundsätzlich unterscheidet sich das Prototyping nicht von traditionellen Vorgehensmodellen, die auf dem „Life Cycle"- Prinzip basieren. Es kann als eine Art Ergänzung zu herkömmlichen Modellen angesehen werden, die den Benutzer während der gesamten Entwicklung miteinbezieht.[139] Der Unterschied liegt primär darin, dass die einzelnen Phasen zwar beibehalten werden, sich jedoch hinsichtlich der Anforderungsanalyse und Systemspezifikation zeitlich stark überlappen. Der Spezifikationsprozess und die Entwurfsphase werden durch Prototyping-Aktivitäten ergänzt (siehe Abbildung 11).

137 Vgl. Leimbach, Timo: Die Geschichte der Softwarebranche in Deutschland. Entwicklung und Anwendung von Informations- und Kommunikationstechnologie zwischen den 1950ern und heute, Stuttgart: Fraunhofer Verlag 2011, S. 30 und 369.

138 Vgl. Budde, Reinhard; Kautz, Karlheinz; Kuhlenkamp, Karin; Züllighoven, Heinz: Prototyping. An Approach to Evolutionary System Development, Berlin, Heidelberg: Springer 1992, S. 6.

139 Vgl. Pomberger, Gustav; Weinreich, Rainer: Qualitative und quantitative Aspekte prototypingorientierter Software-Entwicklung – Ein Erfahrungsbericht, in: Informatik-Spektrum, 20/1997, S. 33.

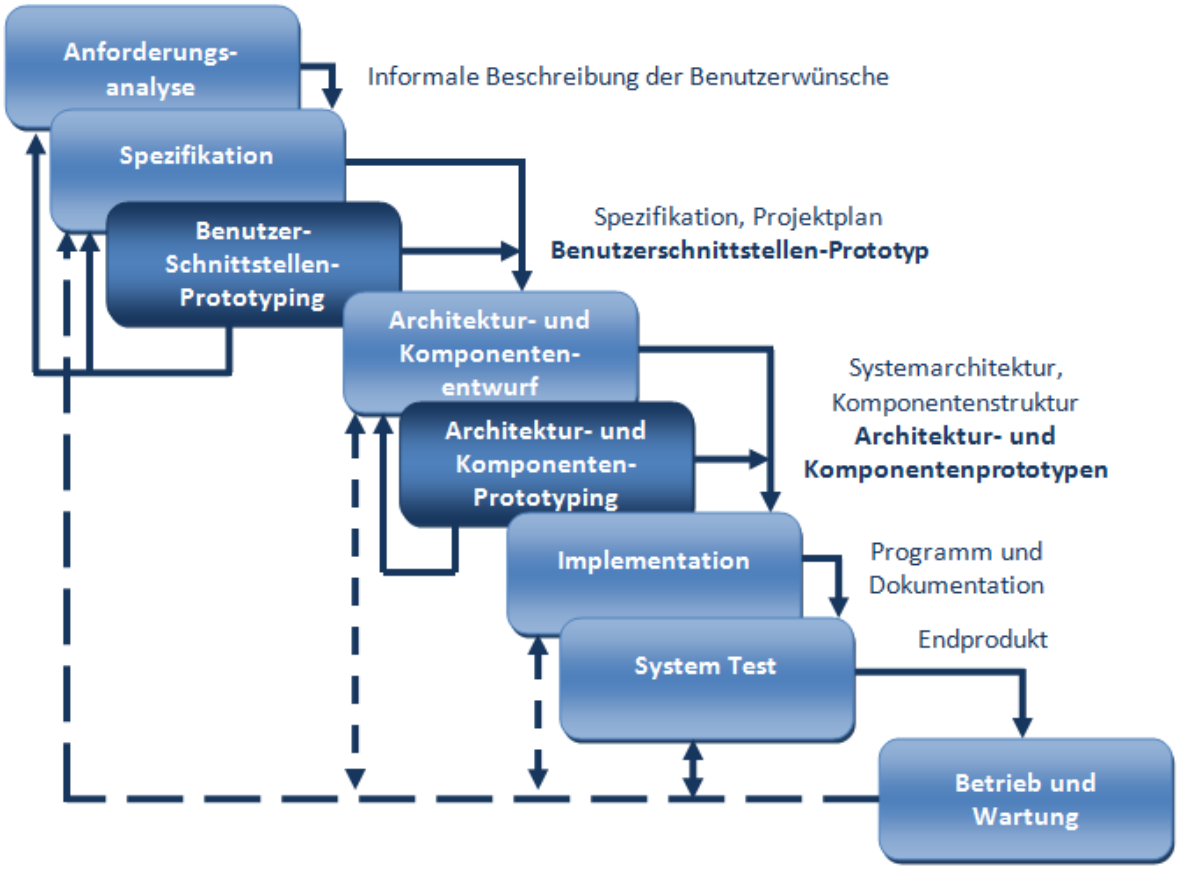

Abbildung 11: Prototyp-orientierter Software-Lebenszyklus[140]

Die einzelnen Phasen bilden hierbei keine chronologischen Abschnitte mehr, sondern entsprechen eher Aktivitäten, deren exakte Trennung gemäß dem Software Life Cycle aufgeweicht wird.[141]

Beim Prototyping wird versucht, möglichst frühzeitig im Entwicklungsprozess funktionsfähige Prototypen von Teilfunktionen der Software zu entwickeln.[142] Dabei lassen sich nach Floyd drei Arten der Prototyp-Entwicklung unterscheiden:

1. *Explorativ*: Hier liegt der Schwerpunkt auf der Klärung der Anforderungen und den gewünschten Eigenschaften des Zielsystems. Ebenfalls werden unterschiedliche Lösungsmöglichkeiten diskutiert.

140 In Anlehnung an Pomberger, Gustav; Pree, W.; Stritzinger, Alois: Methoden und Werkzeuge für das Prototyping und ihre Integration, in: Informatik Forschung und Entwicklung, Vol. 7, 2/1992, S. 51.

141 Vgl. Pomberger, Gustav; Pree, W.; Stritzinger, Alois: Methoden und Werkzeuge für das Prototyping und ihre Integration, a. a. O., S. 51.

142 Vgl. Pomberger, Gustav; Weinreich, Rainer: Qualitative und quantitative Aspekte prototypingorientierter Software-Entwicklung – Ein Erfahrungsbericht, a. a. O., S. 34.

2. *Experimentell*: Für eine bereits bekannte Zielsetzung und feststehende Anforderungen werden Lösungsmöglichkeiten untersucht, bevor in größerem Maße in das Zielsystem investiert wird.

3. *Evolutionär*: Das System wird nach und nach an sich ändernde Anforderungen angepasst. Diese Anforderungen können in frühen Phasen nicht zuverlässig festgelegt werden.[143]

Bei explorativen und experimentellen Prototypen handelt es sich im Wesentlichen um Wegwerfprototypen, die lediglich zur Sammlung von Erfahrungen benutzt werden. Auf Basis dieser Erfahrungen wird dann ein völlig neues, endgültiges System erstellt. Dieses Vorgehen wird auch als „Rapid Prototyping" bezeichnet. Evolutionäre Prototypen werden schrittweise verbessert, indem bereits entwickelte Teilsysteme weiterbenutzt werden.[144] Der Endbenutzer wird in die Entwicklung eingebunden, bis ein aus seiner Sicht zufriedenstellendes Ergebnis vorliegt.[145] Ob die einzelnen Prototypen weggeworfen oder in das System integriert werden, spielt für das allgemeine prototypische Vorgehen keine Rolle. Allerdings ist eine Integration der Prototypen in das System wünschenswert.[146]

Einsatzbereiche

Prototyping ist dann sinnvoll, wenn die Anforderungen bzw. Teile der Anforderungen an das zu entwickelnde System unklar sind. Ein Analyse-Prototyp kann dazu beitragen, die Anforderungen oder die kritischen Bereiche mit dem Benutzer/Kunden zu klären und anschließend zu dokumentieren. Dadurch lassen sich instabile Anforderungen reduzieren. Treten im weiteren Entwicklungsverlauf Änderungen an den Anforderungen auf, führen diese zu einer Überarbeitung aller bereits erzielten Ergebnisse. Entwurfs-Prototypen dienen der Überprüfung, ob nicht-funktionale Anforderungen mit der gewählten Architektur umgesetzt werden können, bzw. ob diese überhaupt realisierbar sind.

143 Floyd, Christiane: A systematic look at Prototyping, in: Approaches to prototyping, Hrsg.: Budde, Reinhard; Kuhlenkamp, Karin; Mathiassen, Lars; Züllighoven, Heinz, Berlin, Heidelberg, New York, Tokyo: Springer 1984, S. 6.

144 Vgl. Stahlknecht, Peter; Hasenkamp, Ulrich: Einführung in die Wirtschaftsinformatik, a. a. O., S. 219.

145 Vgl. Fritzsche, Martin; Keil, Patrick: Kategorisierung etablierter Vorgehensmodelle und ihre Verbreitung in der deutschen Software-Industrie, Institut für Informatik der Technischen Universität München, München 2007, S. 22.

146 Vgl. Pomberger, Gustav; Pree, W.; Stritzinger, Alois: Methoden und Werkzeuge für das Prototyping und ihre Integration, a. a. O., S. 52.

Müssen feste Auslieferungstermine für das zu entwickelnde System eingehalten werden, so sollte Prototyping nicht eingesetzt werden. Zwar entstehen bereits relativ früh erste Prototypen, allerdings werden die folgenden Entwicklungsschritte vollständig durchgeführt. Ein Endergebnis entsteht demnach erst später im Projektverlauf. Dies kann dazu führen, dass zum geplanten Termin lediglich ein ausführbarer Prototyp zur Verfügung steht.[147]

Vor- und Nachteile

Durch eine präzise Systemspezifikation wird eine genaue Abschätzung des notwendigen Entwurfs-, Implementierungs- und Testaufwands ermöglicht. Dies führt zu einer verbesserten Planbarkeit und somit zu erhöhter Termintreue von Softwareprojekten.

Ebenfalls unterstützt das Prototyping die arbeitsteilige Software-Entwicklung. Zum einen werden Probleme bei der Kommunikation zwischen Anwendern und Entwicklern reduziert, zum anderen werden auch Kommunikationsprobleme unter den Entwicklern verringert. Eine experimentell abgesicherte Überprüfung eines Modells, wie sie beim Prototyping angewandt wird, reduziert die Anzahl an unsicheren Annahmen in einem Softwareprojekt. Dadurch wird die Gefahr des Scheiterns verringert.

Trotz einiger Vorteile bringt das Prototyping auch neue Risikofaktoren mit sich. Der Spezifikationsprozess wird oftmals unnötig aufgebläht und in die Länge gezogen.

Auch besteht die Gefahr, dass anstatt von fertigen Produkten Systemprototypen ausgeliefert werden. Dies führt zu erheblichen Qualitätseinbußen.[148]

Entwickler können den Unterschied zwischen Prototyp und eigentlichem Produkt aus den Augen verlieren. Es kann vorkommen, dass Entwickler dadurch weniger diszipliniert und zielorientiert in der Produktentwicklung arbeiten und sich die Arbeitsweise wieder einem unstrukturierten Vorgehen („code & fix“) annähert.[149]

Besonderheiten

Da das Prototyping bei unterschiedlichen Vorgehensmodellen wie z.B. dem Wasserfallmodell oder dem Spiralmodell zum Einsatz kommt, kann hier weniger von einem eigenständigen Vorgehensmodell die Rede sein. Vielmehr handelt es sich beim Prototy-

147 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 10 f.

148 Vgl. Pomberger, Gustav; Weinreich, Rainer: Qualitative und quantitative Aspekte prototypingorientierter Software-Entwicklung – Ein Erfahrungsbericht, in: Informatik-Spektrum, 20/1997, S. 37.

149 Vgl. Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer: Software-Entwicklung, a. a. O., S. 67.

ping um eine Technik,[150] die als eine Art „Add-on“ bei verschiedenen Vorgehensmodellen angewendet werden kann. In der Literatur wird der prototypische Ansatz oft fälschlicher Weise als eigenständiges Vorgehensmodell bezeichnet. Es findet aufgrund seines verwandtschaftlichen Verhältnisses[151] zu den iterativen bzw. evolutionären Vorgehensmodellen, der Vollständigkeit halber, auch in dieser Rubrik Erwähnung.

5 Agile Vorgehensmodelle

5.1 Scrum

Bei Scrum handelt es sich um ein Managementframework zur Entwicklung von Software, das aus wenigen klaren Regeln besteht. Es verkörpert als agiles Framework die Werte des agilen Manifests[152]. Demnach steht der Mensch im Mittelpunkt der Software-Entwicklung.[153] Die Planung wird in Scrum zu Beginn nur teilweise und grob vorgenommen. Diese Planung wird dann phasenweise pro Sprint verfeinert. Die Architektur und das Produkt wachsen somit evolutionär. So kann schnell und unter Berücksichtigung aller Konsequenzen auf Veränderungen reagiert werden.[154]

Entstehung

Der Begriff „Scrum“ stammt aus dem Rugby-Sport und bedeutet zu Deutsch in etwa „Gedränge“. Bei diesem „Gedränge“ stehen sich zwei Mannschaften in einem kreisförmigen Gebilde gegenüber und versuchen, den Gegner gemeinschaftlich am Raumgewinn zu hindern.[155] Der Rugby-Sport ist zwar dafür bekannt, dass es sehr rau zugeht,

150 Vgl. Pomberger, Gustav; Weinreich, Rainer: Qualitative und quantitative Aspekte prototypingorientierter Software-Entwicklung – Ein Erfahrungsbericht, in: Informatik-Spektrum, 20/1997, S. 37.

151 Vgl. Ludewig, Jochen; Färberböck, Hannes; Lichter, Horst; Matheis, Hans: Software-Entwicklung durch schrittweise Komplettierung, in: Requirementsengineering‘87, GMD-Studien 121, Hrsg.: Schmitz, Paul, St. Augustin: GMD 1987, S. 115.

152 Das agile Manifest beinhaltet die wesentlichen Gemeinsamkeiten aller agilen Verfahren. Dazu gehören beispielsweise Extreme Programming, Scrum, Crystal oder das Adaptive Software Development. Weitere Informationen zum agilen Manifest sind unter folgender Website erhältlich: www.agilemanifesto.org.

153 Vgl. Pichler, Roman: Scrum – Agiles Projektmanagement erfolgreich einsetzen, Heidelberg: dpunkt Verlag 2008, S. 1.

154 Vgl. Schiffer, Bernd: Scrum-Einführung bei einem Internet Service Provider – Leben und Werk eines ScrumMasters, in: Agile Projekte mit Scrum, XP und Kanban im Unternehmen durchführen, Hrsg.: Wolf, Henning, Heidelberg: dpunkt Verlag 2012, S. 30 f.

155 Vgl. Pichler, Roman: Scrum – Agiles Projektmanagement erfolgreich einsetzen, a. a. O., S. 2 und Gloger, Boris: Scrum – Produkte zuverlässig und schnell entwickeln, 3. Aufl., München: Hanser 2008, S. 10.

jedoch gibt es auch einige wenige Regeln, die strikt einzuhalten sind. Dies lässt sich auf Scrum übertragen, da hier ebenfalls nur wenige Regeln zu beachten sind, diese aber sehr genau umgesetzt werden.[156] Sowohl beim Rugby als auch bei der Software-Entwicklung steht das kooperative, selbstverantwortliche und selbstorganisierte Verhalten eines vernetzten Teams im Mittelpunkt. Die Idee für Scrum resultiert aus der Beobachtung, dass kleine, eigenverantwortliche Teams, deren Mitglieder alle Aufgaben, die für die Produktentwicklung notwendig sind, beherrschen, eine höhere Effizienz haben als heterogene Teams mit Overheads in einer Matrixdarstellung.[157]

Die Ursprünge von Scrum liegen in der japanischen Konsumgüterbranche. Als eine neue Methode zur Produktentwicklung wurde Scrum 1986 von Takeuchi und Nonaka in der Veröffentlichung „The New Product Development Game“ vorgestellt. Schwaber leitete 1995 das heutige Scrum für die Systementwicklung her.[158] Seitdem wird Scrum durch Erfahrungen von tausenden Menschen korrigiert und erweitert.[159]

Zentrale Charakteristika und Ablauf

Scrum-Projekte beinhalten drei klar getrennte Rollen, die von den Projektbeteiligten begleitet werden müssen:

1. *Product-Owner:* Der Produktverantwortliche übernimmt die Sichtweise des Kunden. Er arbeitet eng mit dem Entwicklungsteam zusammen und steuert die Software-Entwicklung. Auch ist er verantwortlich für das Product-Backlog.[160]

2. *Team:* Das Entwicklungsteam ist dafür verantwortlich, ein nützliches Produkt zu liefern. Es besteht aus mehreren Mitarbeitern, die sich selbst organisieren und selbstständig Entscheidungen treffen. Das Team sollte möglichst aus Mitgliedern unterschiedlicher Disziplinen zusammengesetzt sein.[161]

156 Vgl. Gloger, Boris: Scrum – Produkte zuverlässig und schnell entwickeln, a. a. O., S. 10.

157 Vgl. Goll, Joachim: Methoden und Architekturen der Softwaretechnik, a. a. O., S. 118 f.

158 Vgl. Schwaber, Ken: SCRUM Development Process, in: Business Object Design and Implementation, OOPSLA ´95 Workshop Proceedings 16 October 1995, Austin, Texas, Hrsg.: Sutherland, Jeff; Casanave, Cory; Miller, Joaquin; Patel, Philip; Hollowell, Glenn, London: Springer 1997, S. 118 und Brandstäter, Jonathan: Agile IT-Projekteerfolgreichgestalten, a. a. O., S. 10 f.

159 Vgl. Goll, Joachim; Hommel, Daniel: Mit Scrum zum gewünschten System, Wiesbaden: Springer Vieweg 2015, S. 83 f.

160 Vgl. Hanser, Eckhart: Agile Prozesse: Von XP über Scrum bis MAP, a. a. O., S. 62 und Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, Bonn: mitp-Verl. 2005, S. 105.

161 Vgl. Goll, Joachim; Hommel, Daniel: Mit Scrum zum gewünschten System, a. a. O., S. 90 und Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 105.

3. *Scrum-Master:* Der Scrum-Master ist verantwortlich für ein intaktes und produktives Scrum-Team. Er koordiniert alle Aktivitäten bezüglich Entwicklung und Planung. Außerdem ist er dafür verantwortlich, dass die Regeln von Scrum eingehalten werden.[162]

Der Entwicklungszyklus in Scrum besteht im Wesentlichen aus drei Phasen (siehe Abbildung 12):

1. *Pre-Game* bzw. *Pre-Sprint-Phase:* In dieser Phase liegt der Schwerpunkt auf der Planung und dem Design der Grobarchitektur. Anforderungen an das Produkt werden in einer Liste, dem „Product-Backlog“, gepflegt, aktualisiert, erweitert und priorisiert. Monatlich entnimmt das Team ein definiertes Arbeitspaket aus dem Product-Backlog und setzt dies komplett in Funktionalität um.[163]

2. *Game* bzw. *Sprint-Phase:* Ein Arbeitspaket entspricht einem Inkrement, das während der laufenden Iteration (dem Sprint) nicht weiter modifiziert wird, da dies die Fertigstellung gefährden könnte. Alle weiteren Teile auf dem Product-Backlog können vom Auftraggeber in Vorbereitung auf den nächsten Sprint verändert werden. Aus dem Inkrement werden weitere kleinere Arbeitspakete erstellt, die dann in einer täglich aktualisierten Liste (dem Sprint-Backlog) festgehalten werden. Während des Sprints, der in der Regel ca. 30 Tage in Anspruch nimmt, werden diese Aspekte dann in vollständig und produktiv einsetzbare Anwendungen umgesetzt. Ein tägliches Meeting dient dabei dem Abgleich der Teams untereinander (Daily Scrum).[164]

3. *Post-Game* bzw. *Post-Sprint-Phase:* Nach mehreren Sprints kann die Software idealerweise an den Anwender übergeben werden. Vor diesem Schritt wird die Software ausgiebig getestet. Eventuelle Fehler werden als Bugfix in das Product-Backlog aufgenommen und durch einen neuen Sprint behoben.[165]

162 Vgl. Goll, Joachim; Hommel, Daniel: Mit Scrum zum gewünschten System, a. a. O., S. 91 und Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 129.

163 Vgl. Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 106 und Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 126.

164 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 126.

165 Vgl. Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 107.

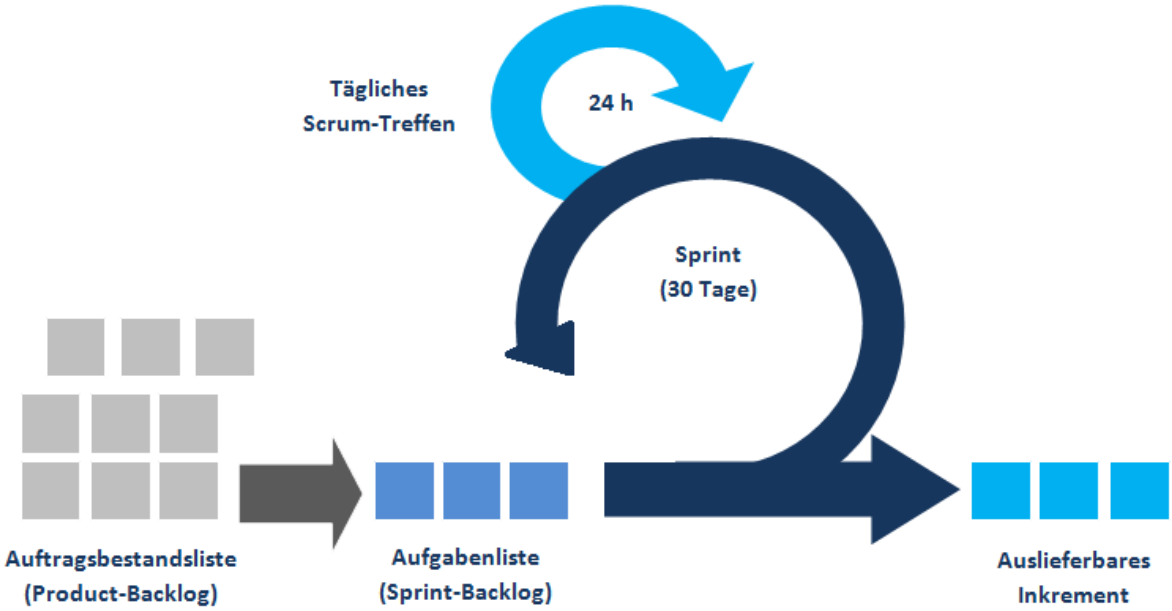

Abbildung 12: Übersicht Scrum-Entwicklungsprozess[166]

Einsatzbereiche

Scrum eignet sich besonders für Entwicklungsprojekte, bei denen die Anforderungen kaum bekannt sind. Der Kunde kann zu Projektbeginn noch keine genaue Auskunft darüber erteilen, was er eigentlich möchte.[167] Ebenfalls ist Scrum gut geeignet für Projekte, die im Laufe der Zeit aus dem Ruder gelaufen sind.[168]

Um Scrum einsetzen zu können, sollte eine geringe Zahl an hierarchischen Strukturen bestehen. Eher sollte ein kooperativer Führungsstil gewählt werden, da die Entwicklerteams eigenständig und eigenverantwortlich arbeiten.[169] In der Regel sollte die Anzahl der Teammitglieder nicht größer sein als fünf bis zehn Personen, da das Team am selben Ort arbeiten muss. Bei größeren Teams muss das Projekt aufgeteilt werden. Dann kommen zusätzliche Organisationselemente hinzu, die gleichzeitig in mehreren Scrum-Teilprojekten koordiniert werden müssen. Dies wird auch als „Scrum of Scrums“ bezeichnet. Allerdings sollte nur dann verteilt werden, wenn dies unumgänglich ist.[170]

166 In Anlehnung an Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 125.

167 Vgl. Goll, Joachim; Hommel, Daniel: Mit Scrum zum gewünschten System, a. a. O., S. 102.

168 Vgl. Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 113.

169 Vgl. Goll, Joachim; Hommel, Daniel: Mit Scrum zum gewünschten System, a. a. O., S. 102.

170 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 233.

Vor- und Nachteile

Durch die inkrementelle Auslieferung von Produktkomponenten in kurzen Zeitspannen kann eine lange Planungszeit vermieden werden. Hinsichtlich der sich schnell ändernden Anforderungen bringen kurze Planungsphasen Vorteile.

Die Selbstorganisation der Teams fördert die Identifikation des Einzelnen mit dem Projekt und erhöht somit den Teamgeist. Tägliche Meetings und kleine Teams führen dazu, dass jeder erfährt, woran gearbeitet wird und welche Probleme bestehen. Durch schnelle Feedbacks der Kunden kann verhindert werden, dass die Entwicklung in die falsche Richtung geht.[171]

Unter Umständen kann es vorkommen, dass sich ein Team auf die funktionalen Anforderungen fokussiert und darüber hinaus vergisst, zentrale Aspekte wie die Architektur oder die rechtzeitige Entwicklung von Prototypen voranzutreiben.

Inhalte des Product- und Sprint-Backlog werden oft nicht ausreichend visualisiert.[172] Bei Scrum tragen die Teams eine enorme Verantwortung. Umso mehr müssen die einzelnen Mitglieder fähig und gewillt sein, zusammenzuarbeiten.[173]

5.2 Extreme Programming

Extreme Programming (XP) ist die wohl am häufigsten verwendete agile Methode. Es ist eine Vorgehensweise zur prototypischen Systementwicklung, bei der versucht wird, Ideen so früh wie möglich praktisch umzusetzen. Hierbei werden keine ausführlichen theoretischen Analysephasen durchgeführt, sondern es erfolgt ein kontinuierliches Kundenfeedback anhand des erstellten Prototyps.[174]

Entstehung

XP geht zurück auf Kent Beck, der als Berater im später bekannt gewordenen C3-Projekt bei Daimler Chrysler engagiert worden war. In diesem Projekt sollte ein System entwickelt werden, mit dem die gesamte Lohnbuchhaltung bei Daimler Chrysler abgewickelt werden sollte. Es gab allerdings enorme Probleme bei der Entwicklung des gewünschten Systems und so stand es kurz vor dem Abbruch. Daraufhin überzeugte Beck

171 Vgl. Goll, Joachim; Hommel, Daniel: Mit Scrum zum gewünschten System, a. a. O., S. 108.
172 Vgl. Goll, Joachim; Hommel, Daniel: Mit Scrum zum gewünschten System, a. a. O., S. 109.
173 Vgl. Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 113.
174 Vgl. Goll, Joachim: Methoden und Architekturen der Softwaretechnik, a. a. O., S. 116.

die damalige CIO des Unternehmens jedoch davon, das Projekt noch einmal neu zu beginnen – mit überarbeiteten Arbeitsweisen. Beck führte viele neuartige Ideen, u.a. solche von Ward Cunningham und Ron Jeffries, ein und so konnten Erfolge erzielt werden, die vorher nicht für realistisch gehalten wurden. Bereits nach drei Wochen konnte beispielsweise ein Gehaltsscheck ausgedruckt werden. Dies war zuvor in 18 Monaten Entwicklung nicht erreicht worden. Auf der Grundlage der Erfahrungen, die Beck mit dem C3-Projekt gemacht hatte, erweiterte er die Methodik, bis im Jahr 1996 XP als Vorgehensmodell entwickelt wurde.[175]

Zentrale Charakteristika und Ablauf

XP bietet konkrete Praktiken rund um die Programmierung und die Einbindung des Kunden. Somit stehen der Entwickler und der Kunde im Vordergrund. Das Vorgehensmodell versucht, organisatorischen Ballast möglichst zu vermeiden. Der Fokus liegt auf dem Erstellen des Programmcodes. Das Ziel von XP ist die effiziente Entwicklung qualitativ hochwertiger Software unter Berücksichtigung des Zeit- und Kostenbudgets. Zur Erreichung dieses Ziels hält XP eine Reihe von Werten, Prinzipien und Entwicklungstechniken bereit.[176] Die vier zentralen Werte mit zugehörigen Prinzipien sind die folgenden:

1. *Kommunikation:* Die Ursachen für Probleme liegen meist in mangelnder bzw. schlechter Kommunikation. Daher soll das Projektteam nicht räumlich getrennt arbeiten, so dass der Kunde stetig in den Entwicklungsprozess eingebunden werden kann.[177]
2. *Einfachheit:* Die Arbeitsprodukte sollen nicht nur leicht verständlich sein, sondern auch nur jene Aspekte umfassen, die tatsächlich zum aktuellen Zeitpunkt zu berücksichtigen sind. Erreicht werden kann dies durch ständiges Refactoring.[178]
3. *Feedback:* Das Projekt-Team soll so schnell und so oft wie möglich Feedback über die erzielten Arbeitsergebnisse erhalten. Kleine und häufige Releases unterstützen die inkrementelle Entwicklung.[179]

175 Vgl. Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 86 und Hanser, Eckhart: Agile Prozesse: Von XP über Scrum bis MAP, a. a. O., S. 13.

176 Vgl. Rumpe, Bernhard: Agile Modellierung mit UML, a. a. O., S. 14 und Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 222.

177 Vgl. Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 85 und Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 222.

178 Vgl. Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 85 f.

4. *Eigenverantwortung und Courage:* Die Mitarbeiter müssen in der Lage sein, schlechte Arbeitsergebnisse zu verwerfen, anstatt mit aller Gewalt auf ihnen aufzubauen. Durch Pair Programming und einer gemeinsamen Verantwortung für den Code liegt der Fokus auf den handelnden Personen.[180]

In XP kommen laut Wolf, Rook und Lippert folgende bewährte Entwicklungstechniken zum Einsatz:

Entwicklungstechnik	**Beschreibung**
Managementtechniken	
Kunde vor Ort	Kunde gibt Anforderungen vor und steht stets als Ansprechpartner vor Ort zur Verfügung.
Planspiel	Kunde formuliert Anforderungen auf Story-Cards. Entwickler schätzen deren Aufwand.
Kurze Release-Zyklen	Systemkomponenten werden Anwendern in kurzen Abständen zur Verfügung gestellt.
Teamtechniken	
Metapher	Metaphern beschreiben die Kernideen des Systems.
Gemeinsame Verantwortung	Entwickler sind gemeinsam für das System verantwortlich.
Fortlaufende Integration	Änderungen am System werden möglichst schnell integriert.
Programmierstandards	Der Quellcode ist einheitlich gestaltet.
40-Stunden-Woche	Entwickler arbeiten nicht länger als die Regelarbeitszeit es vorsieht.
Programmiertechniken	
Testen	Alles was programmiert wurde, wird getestet.
Einfaches Design	Das Softwaresystem ist möglichst einfach gestaltet.
Refactoring	Strukturdefizite im Entwurf der Software behindern die Weiterentwicklung. Daher werden sie sofort behoben.
Programmieren in Paaren	Es sitzen stets zwei Entwickler an einem Rechner.[181]

Tabelle 1: Entwicklungstechniken bei XP

Das XP-Vorgehensmodell kann grundsätzlich in zwei Phasen unterteilt werden. Die Planungsphase dient dem Verständnis und der Dokumentation der Anforderungen, die iterative Realisierungsphase dient der Entwicklung des Systems. Jede Phase enthält wei-

179 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 222.

180 Vgl. Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 86 und Rumpe, Bernhard: Agile Modellierung mit UML, a. a. O., S. 30.

181 Vgl. Wolf, Henning; Rook, Stefan; Lippert, Martin: eXtreme Programming. Eine Einführung mit Empfehlungen und Erfahrungen aus der Praxis, 2. Aufl., Heidelberg: dpunkt Verlag 2005, S. 12 ff.

tere Aktivitäten zur Erstellung von Zwischenergebnissen (siehe Abbildung 13).[182] Zu Beginn der Planungsphase entscheidet der Kunde zusammen mit den Entwicklern in einem Planspiel über die notwendigen Funktionalitäten des Systems. Dazu werden alle Anforderungen auf sogenannte Story-Cards geschrieben und im nächsten Release via paarweiser Programmierung verwirklicht. Kommen während der Entwicklung Rückfragen der Entwickler auf, so können diese vor Ort mit dem Kunden geklärt werden. Nach Abschluss einer Iteration wird die entwickelte Software getestet und die Entwicklungsphase beendet oder eine neue Iteration begonnen. Anschließend wird die Software gewartet. Sollte der Kunde keine weiteren Modifikationswünsche haben, wird das Projekt beendet.[183]

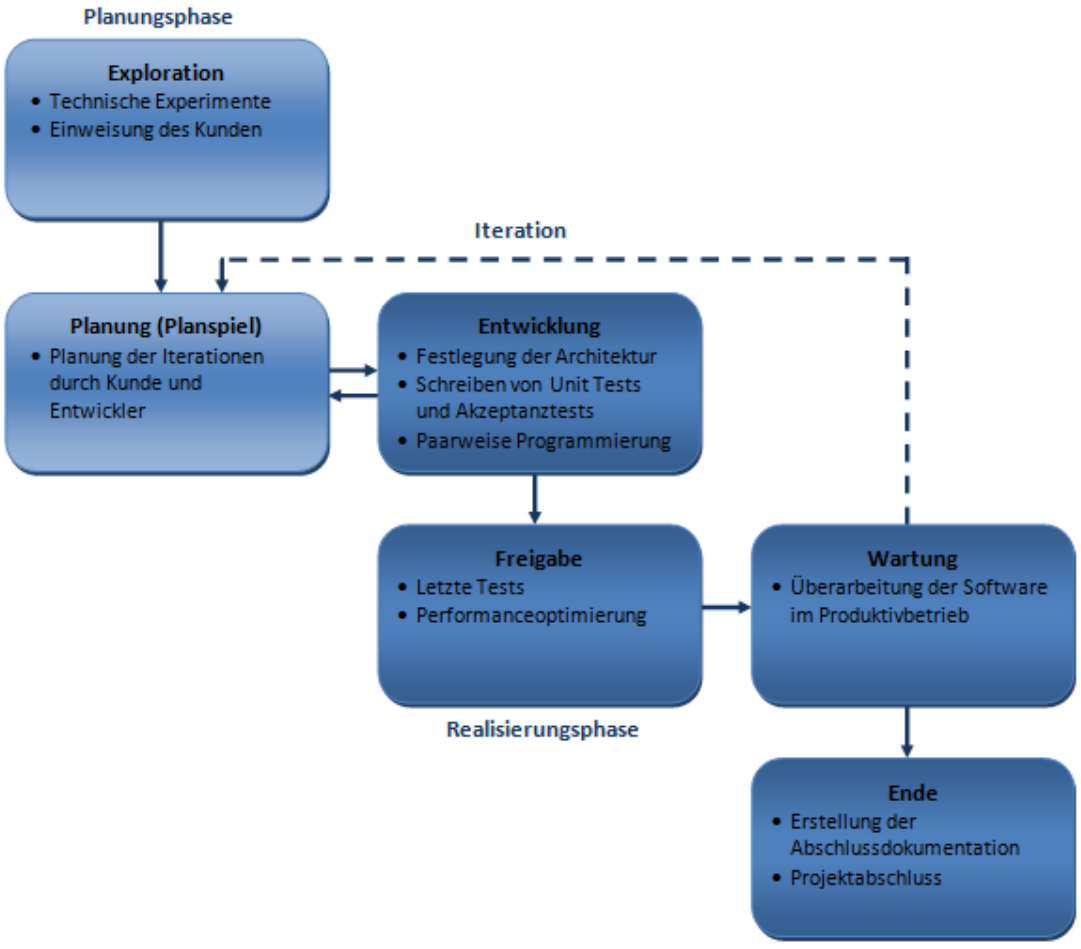

Abbildung 13: XP-Prozess[184]

182 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 116.

183 Vgl. Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 86 und Rumpe, Bernhard: Agile Modellierung mit UML, a. a. O., S. 89 ff.

184 In Anlehnung an Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 89.

Einsatzbereiche

Der Einsatz von XP empfiehlt sich vor allem bei Projekten, deren Anforderungen zu Beginn unklar bzw. unbekannt sind und sich zudem häufig ändern. Gerade Projekte, die einen kurzen Zeitraum für die Implementierung vorsehen, sind hierbei geeignet. Beispielsweise kann XP bei der Entwicklung von Web-Applikationen eingesetzt werden. Für den erfolgreichen Einsatz von XP sollte ein Projektumfeld vorhanden sein, das ein schnelles Feedback ermöglicht. Das Projektteam sollte eine Größe von zwölf Mitarbeitern nicht überschreiten.[185]

Vor- und Nachteile

Aufgrund der Kürze der einzelnen Entwicklungszyklen lassen sich XP-Projekte gut planen und vorhersagen. Darüber hinaus zeichnen sich Software-Systeme, die mit XP entwickelt werden, durch eine einfache und effiziente Implementierung aus, da der Quellcode und die Architektur des Systems ständig optimiert werden (Refactoring).[186] Da die ersten Ergebnisse früh für den Kunden einsehbar sind, kann er Rückschlüsse für seine tatsächlichen Anforderungen an das System ziehen.[187]

Durch die begrenzte Anzahl der Teammitglieder ist gleichzeitig die Größe des gesamten Projekts und damit auch der Umfang des zu realisierenden Systems begrenzt. Ebenfalls fordert XP von allen Beteiligten des Projekts ein hohes Maß an Disziplin, damit auch alle Prinzipien über den gesamten Projektverlauf angewendet werden.[188]

5.3 Crystal

Crystal ist eine Familie von agilen Vorgehensmodellen, die von Alistair Cockburn entwickelt wurde.[189] Bei den Crystal-Methoden stehen die Menschen im Mittelpunkt.[190] Durch die Zusammenarbeit in Teams haben die handelnden Personen mit ihren Interaktionen und ihrer Kommunikation einen signifikanten Einfluss auf den Erfolg eines Software-Projekts.[191]

185 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 116.

186 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 116 f.

187 Vgl. Goll, Joachim: Methoden und Architekturen der Softwaretechnik, a. a. O., S. 117.

188 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 117.

189 Vgl. Hanser, Eckhart: Agile Prozesse: Von XP über Scrum bis MAP, a. a. O., S. 47.

190 Vgl. Hruschka, Peter; Rupp, Chris; Starke, Gernot: Agility kompakt. Tipps für erfolgreiche Systementwicklung, 2. Aufl., Heidelberg: Spektrum Akad. Verl. Springer 2009, S. 52.

191 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 131.

Entstehung

Die Familie der Crystal-Methoden geht zurück auf Alistair Cockburn, der sich in den 1990er-Jahren mit der Entwicklung eines alternativen Vorgehensmodells in der Software-Entwicklung beschäftigte. Dabei stellte er fest, dass alle gängigen Methoden über 13 Kernelemente verfügen. Darunter fallen beispielsweise Prozesse, Standards, Techniken und Aktivitäten. Je mehr Elemente in einer Methode angewendet werden und je strikter diese im Ablauf befolgt werden, desto „starrer" und „schwerer" wird die jeweilige Methode. Cockburn beobachtete, dass bei klassischen Methoden mindestens 90% über die Kernelemente Aktivitäten, Rollen und Produkte verantwortlich für die Gestaltung von Prozessen waren. Dabei kam die menschliche Komponente, das Team, meist zu kurz.[192] Nachdem Cockburn eine Umfrage in verschiedenen Entwicklerteams durchgeführt hatte, kam er zu dem Ergebnis, dass gerade die kommunikativen und gemeinschaftlichen Aspekte verantwortlich für die erfolgreiche Entwicklung von Software waren.[193] Daher forderte er eine Verschiebung der Gewichtung bei den Methoden zugunsten der menschlichen Faktoren.[194] Durch diese Abkehr von den klassischen Elementen konnte ein „ultraleichter" Prozess entstehen, der den Menschen im Mittelpunkt sah: Crystal.

Die Analogie zum Kristall stellt dabei die Unterteilung der einzelnen Vertreter der Crystal-Familie hinsichtlich Farbe und Härtegrad dar. Die Farbe steht für die Team- bzw. Projektgröße. Unter dem Härtegrad versteht Cockburn die sogenannte Kritikalität, also die Schwere des Schadens, den die Software bei Fehlern anrichten kann.[195]

Zentrale Charakteristika und Ablauf

Eine Einteilung gemäß Projektgröße und Kritikalität eines Projekts zeigt Abbildung 14. Die Farben der einzelnen Entwicklungsmethoden stehen für die „Agilität der Methode". Je heller die Farbe, desto leichtgewichtiger die Methode.[196] Die Methoden Crystal Clear, Crystal Orange und Crystal Orange Web wurden bisher entwickelt und dokumentiert.

192 Vgl. Hanser, Eckhart: Agile Prozesse: Von XP über Scrum bis MAP, a. a. O., S. 47 und Hruschka, Peter; Rupp, Chris; Starke, Gernot: Agility kompakt. TippsfürerfolgreicheSystementwicklung, a. a. O., S. 53.

193 Vgl. Cockburn, Alistair: Agile Software Development: The Cooperative Game, 2nd Edition, Boston: Addison-Wesley 2006, S. 6.

194 Vgl. Hruschka, Peter; Rupp, Chris; Starke, Gernot: Agility kompakt. Tipps für erfolgreiche Systementwicklung, a. a. O., S. 53.

195 Vgl. Hanser, Eckhart: Agile Prozesse: Von XP über Scrum bis MAP, a. a. O., S. 47.

196 Vgl. Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt, a. a. O., S. 131.

Die weiteren Methoden stehen noch aus. Dies gilt auch für sämtliche Projekte der Kritikalitätsklasse L (Projekte, deren fehlerfreies Funktionieren lebensnotwendig ist), da Cockburn über keinerlei Erfahrungen in diesem Bereich verfügt.[197]

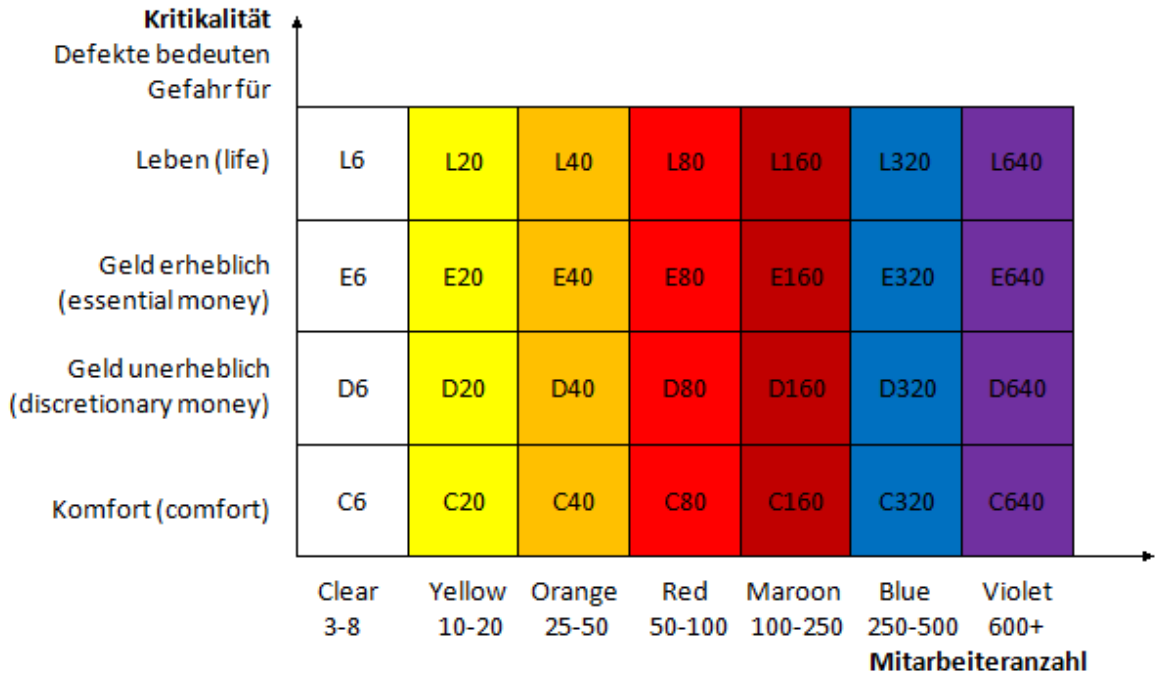

Abbildung 14: Überblick über die Crystal-Methodiken[198]

Im Folgenden wird die Methode Crystal Clear exemplarisch für die gesamte Familie näher erläutert. Cockburn beschreibt Crystal Clear als einen einfachen und toleranten Regelsatz, der das Projekt in sicheres Fahrwasser bringt. Im Mittelpunkt stehen dabei eine gute Kommunikation und eine hohe Toleranz gegenüber variierenden Arbeitsstilen. Ein solcher Toleranzspielraum lässt Platz für individuelle Anforderungen bei unterschiedlichen Projekten, da Crystal Clear lediglich drei Projekteigenschaften[199] fordert:

1. *Regelmäßige Lieferung:* Der entwickelte und getestete Code soll regelmäßig an den Kunden ausgeliefert werden, damit der Kunde in gleichbleibenden Abständen sein Feedback zum aktuellen Projektstand abgeben kann.
2. *Reflektierte Verbesserung:* In sogenannten Reflexionsbesprechungen listet ein Team auf, was im Projekt als positiv und was als negativ erachtet wurde. Da-

197 Vgl. Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 145.

198 In Anlehnung an Cockburn, Alistair: Agile Software Development: The Cooperative Game, a. a. O., S. 196 und Dogs, Carsten; Klimmer, Timo: Agile Software-Entwicklung kompakt, a. a. O., S. 145.

199 Vgl. Hanser, Eckhart: Agile Prozesse: Von XP über Scrum bis MAP, a. a. O., S. 53.

raufhin werden Verbesserungsvorschläge diskutiert. Die Ergebnisse werden in die nächste Entwicklungsphase aufgenommen.

3. *Osmotische Kommunikation:* Bei der osmotischen (verdichteten) Kommunikation sind die Kommunikationswege kurz und ergiebig. Die Teams arbeiten im selben Raum und durch diese Nähe fließen Informationen schnell zu allen Mitarbeitern. Dadurch werden die Kosten gering gehalten.[200]

Der Ablauf eines Crystal Clear Projekts ist üblicherweise individuell geregelt. Es sind keine spezifischen Werkzeuge oder Methoden vorgeschrieben. Lediglich die Kommunikation und die reflektierte Verbesserung des Prozesses stehen im Mittelpunkt. Das Team ist ständig darauf bedacht, die Prozessabläufe zu hinterfragen und zu verbessern. Trotz der wenigen Eigenschaften, die für ein Crystal Clear Projekt gefordert werden, sieht Cockburn folgende Anforderungen, die zwingend erfüllt werden müssen:[201]

- Das Produkt wird inkrementell, in regelmäßigen Schritten entwickelt und ca. alle drei Monate ausgeliefert.
- Meilensteine sind durch Entscheidungen und Lieferungen definiert und nicht durch Dokumente.
- Der Benutzer ist direkt beteiligt.
- Muster (Templates) werden in der Codierung und auch sonst verwendet.
- Die Funktionalität wird durch automatisierte Regressionstests geprüft.
- Jedes Release wird von mindestens zwei Benutzern geprüft.
- Zu Beginn und in der Mitte jedes Auslieferungszyklus findet ein Workshop statt, um die Details des Produkts und der Entwicklungsverfahren festzulegen. Lokale Regeln (local matters) wie beispielsweise die Dokumentation des Projekts sind nicht durch Crystal vorgegeben.
- Einige Werkzeuge und Hilfsmittel, wie eine Software zur Versionsverwaltung und eine elektronische Tafel zur Dokumentation der Ergebnisse aus den Besprechungen, sind erforderlich.[202]

Insgesamt handelt es sich bei Crystal eher um einen konzeptionellen Rahmen als um ein konkretes Vorgehensmodell.[203]

200 Vgl. Hanser, Eckhart: Agile Prozesse: Von XP über Scrum bis MAP, a. a. O., S. 49 f.

201 Vgl. Hanser, Eckhart: Agile Prozesse: Von XP über Scrum bis MAP, a. a. O., S. 53 f.

202 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 227.

Einsatzbereiche

Der Einsatz von Crystal ist in vielen unterschiedlichen Unternehmen möglich. Die Vielzahl an Alternativen bietet für nahezu jedes Unternehmen eine individuell abgestimmte Variante. Die Bandbreite von Crystal reicht von der Erstellung einer simplen Software bis zur komplexen Anwendung in der Raumfahrt. Letztere birgt dabei die Gefahr, dass bei Fehlern in der Programmierung Menschenleben in Gefahr sind. Projekte der Kategorie L müssen vor Beginn hinsichtlich rechtlicher Restriktionen geprüft werden. Ein Einsatz von Crystal ist dann nicht mehr zulässig, wenn die Einhaltung einer der vorgegebenen Eigenschaften nicht mehr gewährleistet ist.[204]

Vor- und Nachteile

Eine Differenzierung der Prozesse in Abhängigkeit von der Größe und dem Risiko eines Projekts ist sinnvoll, da ein Software Engineering, das nach dem Prinzip „ein Vorgehensmodell für alle Fälle“ gelehrt und angewendet wird, nicht praxistauglich ist. Eine Reihe von Merkmalen lassen sich nicht beliebig skalieren.

Der Fokus auf der Kommunikation und dem Wohlbefinden der Beteiligten sowie die Flexibilisierung des Prozesses sind Vorteile dieser Methodik.[205]

Die Crystal Methoden legen ihr Augenmerk auf die Kommunikation und nicht auf einen konkreten Prozess. Dadurch entstehen enorme Anforderungen an die Disziplin und die Fähigkeiten der Entwickler, da diese sich nicht an einem Prozess orientieren können.

Auch sind lediglich die Bereiche Crystal Clear bis Orange definiert. Weitere konkrete Beschreibungen stehen noch aus.

203 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 226.

204 Vgl. Cockburn, Alistair: Agile Software Development: The Cooperative Game, a. a. O., S. 338.

205 Vgl. Ludewig, Jochen; Lichter, Horst: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, a. a. O., S. 228.

6 Zukünftige Entwicklungen

Die bisherige Entwicklung von Vorgehensmodellen führte von den klassischen, starren Modellen wie dem Wasser- oder V-Modell bis hin zu den flexiblen agilen Vorgehensmodellen wie beispielsweise Scrum oder XP. Doch welche Formen können Vorgehensmodelle im Bereich der Software-Entwicklung in Zukunft annehmen?

Eine Möglichkeit bietet die Kombination von traditionellen Modellen mit agilen Ansätzen bzw. eine Mischung aus verschiedenen agilen Konzepten. Solche hybriden Vorgehensmodelle können vor dem Hintergrund einer wachsenden Bedeutung der Projektwirtschaft und neuer Organisationsformen temporärer Zusammenarbeit von Unternehmen eine große Rolle spielen.[206] Das Ziel von hybriden Vorgehensmodellen ist es, die Stärken aus klassischen und agilen Modellen zu kombinieren und deren Schwächen zu umgehen. Während beispielsweise bei phasenorientierten Vorgehensmodellen ein Festpreis erhoben werden kann, da zu Projektbeginn die Kundenanforderungen festgelegt werden, ist dies bei agilen Vorgehensmodellen nicht möglich. Hier kann der Kunde seine Anforderungen im Laufe des Projekts verändern. Daher ist es in diesem Kontext kaum möglich, einen Festpreis zu vereinbaren. Bei phasenorientierten Vorgehensmodellen wird die Software erst nach vollständiger Umsetzung beim Kunden getestet. Erfüllt das Produkt die Kundenanforderungen nicht, wird dies erst sehr spät festgestellt. Agile Vorgehensmodelle lösen solche Probleme durch regelmäßiges Kundenfeedback. Ein hybrides Modell könnte die genannten Vorteile folgendermaßen kombinieren: In einer sequenziellen Vorbereitungsphase wird mit dem Kunden eine Anforderungsanalyse mit vertraglichen Rahmenwerten durchgeführt. Die Umsetzung der Anforderungen erfolgt in Iterationen. Nach jeder Iteration wird das Produkt vom Kunden getestet. So können Abweichungen frühzeitig erkannt werden. Nach Umsetzung aller Anforderungen folgt die sequenzielle Finalisierungsphase, in der die Kundenabnahme und Auslieferung stattfindet. Durch die ausführliche Anforderungsanalyse vor der eigentlichen Umsetzung kann hier im Voraus ein Festpreis definiert werden.[207]

206 Vgl. Schelle, Heinz: Aktuelle Trends des Projektmanagements, in: Zukunft der Wissens- und Projektarbeit: Neue Organisationsformen in vernetzten Welten, Hrsg.: Weßels, Doris, Düsseldorf: Symposion Publishing 2014, S. 30.

207 Vgl. Sandhaus, Gregor; Knott, Philip; Berg, Björn: Hybride Softwareentwicklung. Das Beste aus klassischen und agilen Methoden in einem Modell vereint, Berlin: Springer Vieweg 2014, S. 53 f.

Wie groß der Trend zu hybriden Vorgehensmodellen tatsächlich ist, verdeutlicht eine Umfrage der „GULP-Knowledge Base“ mit 223 Freelancern und Projektanbietern – vornehmlich aus der IT und dem Engineering Projektmanagement. Abbildung 15 zeigt, dass 56% der Teilnehmer in Zukunft hybride Vorgehensmodelle einsetzen werden und 28% der Teilnehmer die agilen Methoden bevorzugen, während lediglich 5% klassische Modelle einsetzen werden.

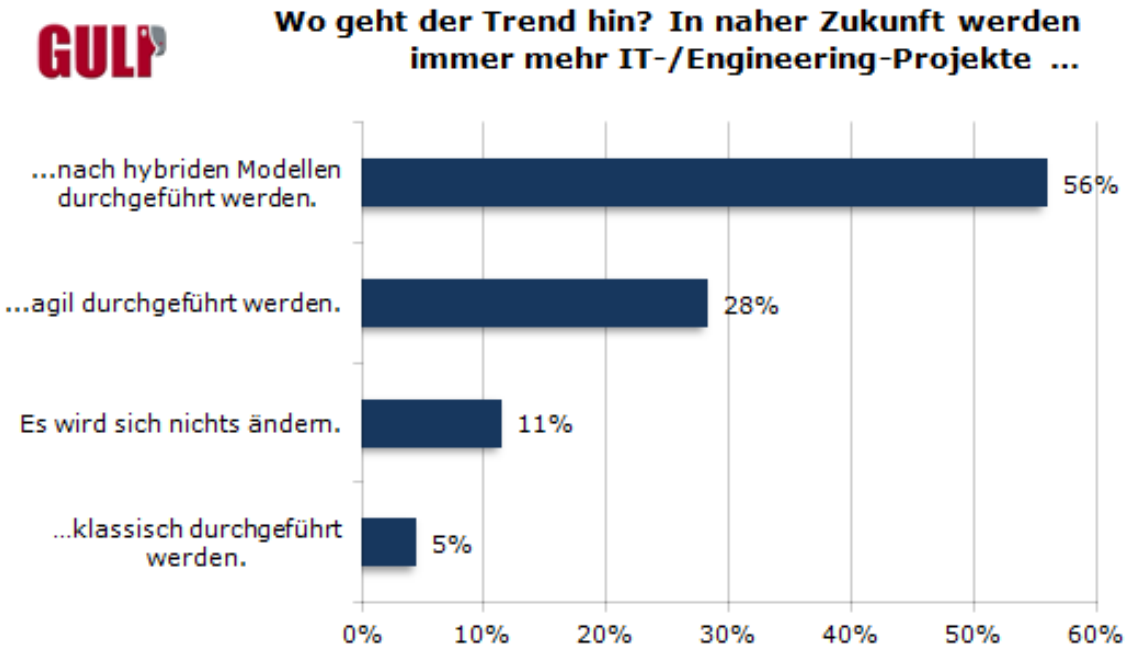

Abbildung 15: Trend zu hybriden Vorgehensmodellen[208]

Weiterhin lässt sich anmerken, dass heutige Vorgehensmodelle oftmals einen hohen Interpretationsspielraum aufweisen. Ein solcher Spielraum reduziert jedoch das Maß an Hilfestellung, das ein Vorgehensmodell bei einem Projekt bieten soll. Daher kann ein Vorgehensmodell mit einer präzise vorgegebenen Anwendung und einfacher Handhabung zum Erfolg eines Projekts beitragen. Gleichzeitig sollen Vorgehensmodelle einfach an konkrete Projekte anzupassen sein. Aufgrund dessen findet das Tailoring, also das Zuschneiden eines Modells auf die im Kontext eines Projekts relevanten Teile, immer größeres Interesse. Des Weiteren wäre die Entwicklung konkreterer Vorgehensmodelle vorteilhaft. Organisationsspezifische Modelle, die sich auf einzelne Geschäftsfelder eines Unternehmens beziehen, oder domänenspezifische Modelle für eine objektori-

208 Vgl. www.gulp.de (Hrsg.): Management von IT-/Engineering-Projekten: Die Zukunft ist hybrid. Ergebnis einer GULP Online-Umfrage unter Freelancern und Projektanbietern. Online im Internet: https://www.gulp.de/knowledge-base/markt-und-trends/management-von-it-engineering-projekten-die-zukunft-ist-hybrid.html, 27.08.2013.

entierte Entwicklung ermöglichen eine zielgerichtete spezifische Anwendung auf bestimmte Klassen von Projekten.[209]

Die bisherige Entwicklung von Vorgehensmodellen in der Softwaretechnik ist gekennzeichnet von unterschiedlichen Herangehensweisen und zum Teil uneinheitlicher Begriffsbildung. Daher soll diese Arbeit einen Beitrag dazu leisten, die verschiedenen Modelle hinsichtlich ihrer Eigenschaften einordnen zu können und auf diese Weise einen Überblick über die Vielzahl an Modellen zu vermitteln. Abschließend bleibt zu bemerken, dass die weitere Entwicklung von Vorgehensmodellen interessant zu beobachten ist. Gerade die Entwicklung hybrider Vorgehensmodelle kann bei einer Einordnung – wie sie in dieser Arbeit stattgefunden hat – in Zukunft zu Herausforderungen führen. Die Möglichkeiten, die eine Kombination von klassischen und agilen Methoden in Form von hybriden Modellen bietet, und Überlegungen hinsichtlich dessen, welche Kombinationen sinnvoll sind, würde eine ergänzende empirische Arbeit darstellen.

209 Vgl. Fritzsche, Martin; Keil, Patrick: Kategorisierung etablierter Vorgehensmodelle und ihre Verbreitung in der deutschen Software-Industrie, a. a. O., S. 43.

Literaturverzeichnis

1. **Abts, Dietmar; Mülder, Wilhelm:** Grundkurs Wirtschaftsinformatik. Eine kompakte und praxisorientierte Einführung. 8., überarb. und erw. Aufl. Wiesbaden: Springer Vieweg 2013.

2. **Balzert, Helmut:** Lehrbuch der Softwaretechnik. Software-Entwicklung, 2. Aufl., 1. Nachdr. Heidelberg: Spektrum Akad. Verlag 2001.

3. **Balzert, Helmut:** Lehrbuch der Softwaretechnik. Basiskonzepte und Requirements Engineering, 3. Aufl., Heidelberg: Spektrum Akad. Verlag 2009.

4. **Balzert, Helmut; Ebert, Christof:** Lehrbuch der Softwaretechnik. Softwaremanagement, 2. Aufl. Heidelberg: Spektrum Akad. Verl. 2008.

5. **Bartelt, Christian; Bauer, Otto; Beneken, Gerd; et al.:** V-Modell XT. Das deutsche Referenzmodell für Systementwicklungsprojekte. Version 2.0, Hrsg.: Verein zur Weiterentwicklung des V-Modell XT e.V., München: 2006.

6. **Bea, Franz Xaver; Göbel, Elisabeth:** Organisation, 3. neu bearb. Aufl. Stuttgart: Lucius & Lucius 2006.

7. **Benington, Herbert D.:** Production of Large Computer Programs. Reprinted in: Annals of the History of Computing, 5, Nr. 4, Oktober 1983, 1956, S. 350-361.

8. **Boehm, Barry W.:** Software Engineering, in: IEEE Transact. on Computers, C-25, Dezember 1976, S. 1216-1241.

9. **Boehm, Barry W.:** Guidelines for verifying and validating software requirements and design specifications, EURO IFIP, North-Holland 1979, S. 711-719.

10. **Boehm, Barry W.:** A Spiral Model of Software Development and Enhancement. in: IEEE Computer, Vol.21, Ausg. 5, Mai 1988, pp 61-72.

11. **Brandstäter, Jonathan:** Agile IT-Projekte erfolgreich gestalten. Wiesbaden: Springer Fachmedien 2013.

12. **Brandt-Pook, Hans; Kollmeier, Rainer:** Softwareentwicklung kompakt und verständlich. Wie Softwaresysteme entstehen, 2. Aufl., Wiesbaden: Springer Vieweg 2015.

13. **Bremer, Georg:** Genealogie von Entwicklungsschemata, in: Vorgehensmodelle für die betriebliche Anwendungsentwicklung, Hrsg.: Kneuper, Ralf; Müller-Luschnat, Günther; Oberweis, Andreas, Wiesbaden: Teubner 1998, S. 32-60.

14. **Budde, Reinhard; Kautz, Karlheinz; Kuhlenkamp, Karin; Züllighoven, Heinz:** Prototyping. An Approach toEvolutionary System Development, Berlin, Heidelberg: Springer 1992.

15. **Bunse, Christian; Von Knethen, Antje:** Vorgehensmodelle kompakt. 2. Aufl. Heidelberg: Spektrum Akad. Verlag 2008.

16. **Chroust, Gerhard:** Modelle der Software-Entwicklung, München: Oldenbourg 1992.

17. **Cockburn, Alistair:** Agile Software Development: The Cooperative Game, 2nd Edition, Boston: Addison-Wesley Professional 2006.

18. **Dogs, Carsten; Klimmer, Timo:** Agile Software-Entwicklung kompakt, Bonn: mitp-Verl. 2005.

19. **Dröschel, Wolfgang:** Einführung in das V-Modell, in: Das V-Modell 97. Der Standard für die Entwicklung von IT-Systemen mit Anleitung für den Praxiseinsatz, Hrsg.: Dröschel, Wolfgang, München: Oldenbourg 2000, S. 1-44.

20. **Ebert, Christof:**Systematisches Requirements Engineering. Anforderungen ermitteln, spezifizieren, analysieren und verwalten, 5. überarb. Aufl. Heidelberg: dpunkt Verl. 2014.
21. **Filß, Chritian:** Vergleichsmethoden für Vorgehensmodelle, Diplomarbeit, TU Dresden, Dresden, 2005.

22. **Fischer, Thomas; Biskup, Hubert; Müller-Luschnat, Günther:** Begriffliche Grundlagen für Vorgehensmodelle, in: Vorgehensmodelle für die betriebliche Anwendungsentwicklung, Hrsg.: Kneuper, Ralf; Müller-Luschnat, Günther; Oberweis, Andreas, Wiesbaden: Teubner 1998, S. 13-32.

23. **Floyd, Christiane:** A systematic look at Prototyping, in: Approaches to prototyping, Hrsg.: Budde, Reinhard; Kuhlenkamp, Karin; Mathiassen, Lars; Züllighoven, Heinz, Berlin, Heidelberg, New York, Tokyo: Springer 1984, S. 1-18.

24. **Friedrich, Jan; Kuhrmann, Marco; Sihling, Marc; Hammerschall, Ulrike:** Das V-Modell XT. Für Projektleiter und QS-Verantwortliche kompakt und übersichtlich, Berlin: Springer 2009.

25. **Fritzsche, Martin; Keil, Patrick:** Kategorisierung etablierter Vorgehensmodelle und ihre Verbreitung in der deutschen Software-Industrie, Institut für Informatik der Technischen Universität München, München 2007. Online verfügbar unter http://mediatum.ub.tum.de/doc/1094277/1094277.pdf.

26. **Gadatsch, Andreas:** Management von Geschäftsprozessen. Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker, 2. überarb. und erw. Aufl. Wiesbaden: Vieweg & Teubner Verlag 2002. Online verfügbar unter http://dx.doi.org/10.1007/978-3-322-91983-0.

27. **Gesellschaft für Informatik (Hrsg.):** Online im Internet: https://www.gi.de/.

28. **Gloger, Boris:** Scrum – Produkte zuverlässig und schnell entwickeln, 3. Aufl., München: Hanser 2008.

29. **Gnatz, Michael:** Vom Vorgehensmodell zum Projektplan. Saarbrücken: VDM Verl. Dr. Müller 2007. Online verfügbar unter http://deposit.d-nb.de/cgi-bin/dokserv?id=3036637&prov=M&dok_var=1&dok_ext=htm.

30. **Goll, Joachim:** Methoden und Architekturen der Softwaretechnik. Wiesbaden: Vieweg+Teubner Verlag (Studium) 2011. Online verfügbar unter http://dx.doi.org/10.1007/978-3-8348-8164-9.

31. **Goll, Joachim; Hommel, Daniel:** Mit Scrum zum gewünschten System, Wiesbaden: Springer Vieweg 2015. Online verfügbar unter http://dx.doi.org/10.1007/978-3-658-10721-5.

32. **Gulp.de (Hrsg.):** Management von IT-/Engineering-Projekten: Die Zukunft ist hybrid. Ergebnis einer GULP Online-Umfrage unter Freelancern und Projektanbietern. Online im Internet: https://www.gulp.de/knowledge-base/markt-und-trends/management-von-it-engineering-projekten-die-zukunft-ist-hybrid.html, 27.08.2013.

33. **Hansen, Hans Robert; Mendling, Jan; Neumann, Gustaf:** Wirtschaftsinformatik. Grundlagen und Anwendungen, 11. Völlig neu bearb. Aufl. Berlin: de Gruyter 2015.

34. **Hanser, Eckhart: Agile Prozesse:** Von XP über Scrum bis MAP, Berlin, Heidelberg: Springer 2010.

35. **Hesse, Wolfgang; Merbeth, Günter; Frölich, Rainer:** Software-Entwicklung, München: Oldenbourg 1992.

36. **Höhn, Reinhard; Höppner, Stephan; Rausch, Andreas:** Das V-Modell XT. Anwendungen, Werkzeuge, Standards, Berlin, Heidelberg: Springer 2008. Online verfügbar unter http://d-nb.info/989133036/34.

37. **Horn, Erika; Schubert, Wolfgang:** Objektorientierte Software-Konstruktion. Grundlagen – Modelle – Methoden – Beispiele. München: Carl Hanser Verlag 1993.

38. **Hruschka, Peter; Rupp, Chris; Starke, Gernot:**Agility kompakt. Tipps für erfolgreiche Systementwicklung, 2. Aufl., Heidelberg: Spektrum Akad. Verl. Springer 2009. Online verfügbar unter http://d-nb.info/997553073/34.

39. **Informatik Forum Simon (Hrsg.):** Online im Internet: http://www.infforum.de/.

40. **Kruchten, Philippe:** The rational unified process. An introduction, 2. ed., Boston: Addison-Wesley 2003.

41. **Leimbach, Timo:** Die Geschichte der Softwarebranche in Deutschland. Entwicklung und Anwendung von Informations- und Kommunikationstechnologie zwischen den 1950ern und heute, Stuttgart: Fraunhofer Verlag 2011.

42. **Loos, Peter; Fettke, Peter:** Aspekte des Wissensmanagements in der Software-Entwicklung am Beispiel von V-Modell und Extreme Programming, TU Chemnitz, Chemnitz, 2001.

43. **Ludewig, Jochen; Färberböck, Hannes; Lichter, Horst; Matheis, Hans:** Software-Entwicklung durch schrittweise Komplettierung, in: Requirementsengineering ´87, GMD-Studien 121, Hrsg.: Schmitz, Paul, St. Augustin: GMD 1987, S. 113-124.

44. **Ludewig, Jochen; Lichter, Horst:** Software Engineering. Grundlagen, Menschen, Prozesse, Techniken. Heidelberg: dpunkt 2007.

45. **Neutschel, Bernd; Vajna, Sandor:** Organisations- und Prozessintegration, in: Integrated Design Engineering. Ein interdisziplinäres Modell für die ganzheitliche

Produktentwicklung, Hrsg.: Vajna, Sandor, Berlin: Springer Vieweg 2014, S. 335-375.

46. **Noack, Jörg; Schienmann, Bruno:** Objektorientierte Vorgehensmodelle im Vergleich, in: Informatik-Spektrum, 22 3/1999, S. 166-180. DOI: 10.1007/s00287005 0135.

47. **Pichler, Roman:** Scrum – Agiles Projektmanagement erfolgreich einsetzen, Heidelberg: dpunkt Verlag 2008.

48. **Pomberger, Gustav; Pree, W.; Stritzinger, Alois:** Methoden und Werkzeuge für das Prototyping und ihre Integration, in: Informatik Forschung und Entwicklung, Vol. 7, 2/1992, S. 49-61.

49. **Pomberger, Gustav; Weinreich, Rainer:** Qualitative und quantitative Aspekte prototypingorientierter Software-Entwicklung – Ein Erfahrungsbericht, in: Informatik-Spektrum, 20/1997, S. 33-37.

50. **Pressman, Roger:** Software engineering. A practitioner´s approach. 5th ed., Boston: Mass: McGraw Hill 2001.

51. **Royce, Winston:** Managing the Development of Large Software Systems, in: IEEE WESCON Hrsg.: Institute of Electrical and Electronics Engineers, 26. Aufl., August 1970, S. 328-338.

52. **Rumpe, Bernhard:** Agile Modellierung mit UML, 2. Aufl., Berlin Heidelberg: Springer 2012.

53. **Sandhaus, Gregor; Knott, Philip; Berg, Björn:** Hybride Softwareentwicklung. Das Beste aus klassischen und agilen Methoden in einem Modell vereint, Berlin: Springer Vieweg 2014. Online verfügbar unter http://dx.doi.org/10.1007/978-3-642-55064-5.

54. **Schelle, Heinz:** Aktuelle Trends des Projektmanagements, in: Zukunft der Wissens- und Projektarbeit: Neue Organisationsformen in vernetzten Welten, Hrsg.: Weßels, Doris, Düsseldorf: Symposion Publishing 2014, S. 1-16.

55. **Schiffer, Bernd:** Scrum-Einführung bei einem Internet Service Provider – Leben und Werk eines ScrumMasters, in: Agile Projekte mit Scrum, XP und Kanban im Unternehmen durchführen, Hrsg.: Wolf, Henning, Heidelberg: dpunkt Verlag 2012, S. 29-47.

56. **Schwaber, Ken:** SCRUM Development Process, in: Business Object Design and Implementation, OOPSLA ´95 Workshop Proceedings 16 October 1995, Austin, Texas, Hrsg.: Sutherland, Jeff; Casanave, Cory; Miller, Joaquin; Patel, Philip; Hollowell, Glenn, London: Springer 1997, S.117-134.

57. **Schwickert, Axel C.:**Web Site Engineering – Modelltheoretische und methodische Erfahrungen aus der Praxis, in: Arbeitspapiere WI, Nr. 8/1997, Hrsg.: Lehrstuhl für Allg. BWL und Wirtschaftsinformatik, Johannes Gutenberg-Universität: Mainz 1997.

58. **Schwickert, Axel C.:**Web Site Engineering – Ein Komponentenmodell, in: Arbeitspapiere WI, Nr. 12/1998, Hrsg.: Lehrstuhl für Allg. BWL und Wirtschaftsinformatik, Johannes Gutenberg-Universität: Mainz 1998.

59. **Seibert, Siegfried:** Technisches Management. Innovationsmanagement, Projektmanagement, Qualitätsmanagement, Stuttgart: Teubner 1998.

60. **Sommerville, Ian:** Software Engineering, 9. aktual. Aufl. München: Pearson 2012.

61. **Stahlknecht, Peter; Hasenkamp, Ulrich:** Einführung in die Wirtschaftsinformatik, 11. Vollst. Überarb. Auflage. Berlin, Heidelberg: Springer 2005.

62. **Timinger, Holger:** Wiley Schnellkurs Projektmanagement, Hoboken: Wiley 2015. Online verfügbar unter http://gbv.eblib.com/patron/FullRecord.aspx?p=2048554.

63. **Versteegen, Gerhard:** Vorgehensmodelle, in: Software Management. Beherrschung des Lifecycles, Hrsg.: Versteegen, G., Berlin, Heidelberg: Springer 2002, S. 29-61. Online verfügbar unter http://dx.doi.org/10.1007/978-3-642-56367-6.

64. **Wolf, Henning; Rook, Stefan; Lippert, Martin:** eXtreme Programming. Eine Einführung mit Empfehlungen und Erfahrungen aus der Praxis, 2. Aufl., Heidelberg: dpunkt Verlag 2005. Online verfügbar unter http://gbv.eblib.com/patron/FullRecord.aspx?p=1175016.

65. **Zaenger, Roland:** Konzeption eines generischen Vorgehensmodells zur Entwicklung kollaborativer Applikationen und prototypische Implementierung am Beispiel einer J2EE-Plattform, Diplomarbeit, Universität Paderborn, Paderborn, 2005

Impressum

Reihe: **Arbeitspapiere Wirtschaftsinformatik** (ISSN 1613-6667)

Bezug: http://wiwi.uni-giessen.de/home/Schwickert/arbeitspapiere/

Herausgeber: Prof. Dr. Axel C. Schwickert
Prof. Dr. Bernhard Ostheimer

c/o Professur BWL – Wirtschaftsinformatik
Justus-Liebig-Universität Gießen
Fachbereich Wirtschaftswissenschaften
Licher Straße 70
D – 35394 Gießen
Telefon (0 64 1) 99-22611
Telefax (0 64 1) 99-22619
eMail: Axel.Schwickert@wirtschaft.uni-giessen.de
http://wi.uni-giessen.de

Ziele: Die Arbeitspapiere dieser Reihe sollen konsistente Überblicke zu den Grundlagen der Wirtschaftsinformatik geben und sich mit speziellen Themenbereichen tiefergehend befassen. Ziel ist die verständliche Vermittlung theoretischer Grundlagen und deren Transfer in praxisorientiertes Wissen.

Zielgruppen: Als Zielgruppen sehen wir Forschende, Lehrende und Lernende in der Disziplin Wirtschaftsinformatik sowie das IT-Management und Praktiker in Unternehmen.

Quellen: Die Arbeitspapiere entstehen aus Forschungsarbeiten, Abschluss-, Studien- und Projektarbeiten sowie Begleitmaterialien zu Lehr- und Vortragsveranstaltungen der Professur BWL – Wirtschaftsinformatik, Univ. Prof. Dr. Axel C. Schwickert, Justus-Liebig-Universität Gießen sowie der Professur für Wirtschaftsinformatik, insbes. medienorientierte Wirtschaftsinformatik, Fachbereich Wirtschaft, Hochschule Mainz.

Hinweise: Wir nehmen Ihre Anregungen und Kritik zu den Arbeitspapieren aufmerksam zur Kenntnis und werden uns auf Wunsch mit Ihnen in Verbindung setzen.

Falls Sie selbst ein Arbeitspapier in der Reihe veröffentlichen möchten, nehmen Sie bitte mit dem Herausgeber unter obiger Adresse Kontakt auf.

Informationen über die bisher erschienenen Arbeitspapiere dieser Reihe erhalten Sie unter der Adresse http://wi.uni-giessen.de.