T U M

I N S T I T U T F Ü R I N F O R M A T I K

Kategorisierung etablierter Vorgehensmodelle und ihre Verbreitung in der deutschen Software-Industrie

Martin Fritzsche, Patrick Keil

with special thanks to

Marco Kuhrmann

TUM-I0717
Juni 07

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-06-I0717-100/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Kategorisierung etablierter Vorgehensmodelle und ihre Verbreitung in der deutschen Software-Industrie

Abstract

Zahlreiche Studien, bspw. SUCCESS, haben in den letzten Jahren darauf hingewiesen, dass in vielen Unternehmen Vorgehensmodelle in der Software-Entwicklung nicht einheitlich und konsequent genug eingesetzt werden. Im Rahmen des vom BMBF geförderten Projektes IOSE-W² (Interorganisationale Softwareentwicklung unter dem Aspekt der Wandlungsfähigkeit und der Wiederverwendung; siehe www.iose-w.de) erbrachte eine Umfrage unter deutschen Unternehmen ein differenzierteres Bild: Vorgehensmodelle wie auch Reifegradmodelle sind weit verbreitet und die Unternehmen schätzen ihre Prozessdisziplin als hoch ein. Aber sowohl die Einheitlichkeit des Prozesses im Unternehmen (83% der Befragten setzen zwei oder mehr unterschiedliche Vorgehensmodelle ein) als auch die Zufriedenheit der Verantwortlichen mit den Prozessen sind sehr niedrig.

In diesem Beitrag werden die meist verbreiteten SE-Vorgehensmodelle wie z.B. Wasserfallmodell, Spiralmodell, V-Modell XT, RUP oder eXtreme Programming in der chronologischen Reihenfolge ihrer Entstehung vorgestellt, katalogisiert und bewertet. Dazu werden zunächst zentrale Begriffe definiert und eine Reihe von Kriterien aufgestellt, mit denen sich Vorgehensmodelle bewerten, vergleichen und hinsichtlich ihrer jeweiligen Stärken und Schwächen analysieren lassen.

Nach der ausführlichen Vorstellung und vergleichenden Bewertung der wichtigsten Vorgehensmodelle werden die auf Entwicklungsprozesse bezogenen Ergebnisse der bereits zitierten Studie vorgestellt, in der deutsche Unternehmen nach ihren Erfahrungen mit Vorgehensmodellen und Software-Prozessen befragt wurden.

Inhaltsverzeichnis

1 Einleitung

1.1 Ziel dieses Dokuments

Ein Vorgehensmodell ist eine abstrakte Darstellung eines Softwareprozesses. Der Prozess wird hierbei immer nur aus einer bestimmten Perspektive dargestellt (Janker 2005). Daher beinhaltet ein Vorgehensmodell immer nur einen Teil der Informationen über den Prozess. Dies hat allerdings den Vorteil, dass die den einzelnen Softwareprozessen zugrunde liegenden Prinzipien übersichtlich und verständlich dargestellt werden können.
In diesem Dokument werden die meist verbreiteten SE-Vorgehensmodelle identifiziert, katalogisiert und bewertet. Damit dient dieses Dokument als Grundlage und Ausgangspunkt für das Arbeitspaket 1.1.2, in dem die hier untersuchten Softwareentwicklungs-Vorgehensmodelle im Hinblick auf ihre Wandlungsfähigkeit beurteilt werden.

1.2 Aufbau

In diesem Dokument werden einige Vorgehensmodelle zusammen mit ihren Stärken und Schwächen dargestellt. In Kapitel 2 werden Kriterien und Eigenschaften zur Einordnung, Systematisierung und Katalogisierung sowie zur Bewertung der verschiedenen Modelle und Ansätze erarbeitet. In Kapitel 3 erfolgt eine überblicksartige Vorstellung der prominentesten Softwareentwicklungs-Vorgehensmodelle, wie z.B. des Wasserfallmodells, des Spiralmodells, des V-Modells, RUP, eXtreme Programming und anderer. Es wird versucht, diese Modelle in der chronologischen Reihenfolge ihrer Entstehung vorzustellen. Auf die einzelnen Vorgehensmodelle werden die in Kapitel 2 erarbeiteten Kriterien angewandt und damit eine Stärken-/Schwächenanalyse sowie ein Vergleich der wichtigsten Vorgehensmodelle ermöglicht.
Der letzte Abschnitt (Kapitel 4) zeigt die aktuellen Trends bei der Anwendung und Weiterentwicklung von Softwareentwicklungs-Vorgehensmodellen auf.

1.3 Begriffsklärung

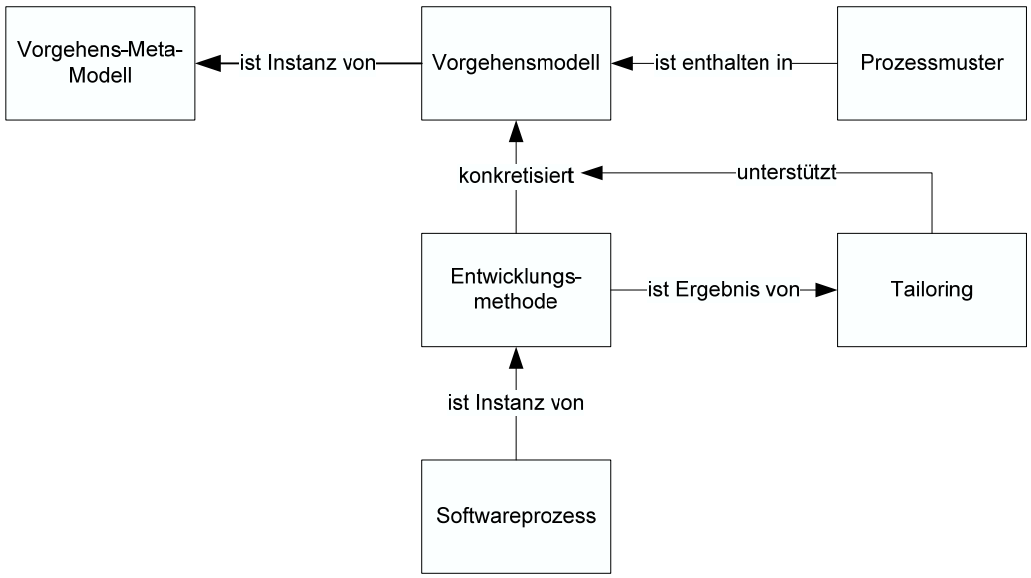


Abbildung 1: Überblick über die Begriffe

Entwicklungsmethode

Eine Entwicklungsmethode ist eine Methode, die eine Vorgehensweise zur Softwareentwicklung vorgibt. Sie dient als konkreter Leitfaden für den Entwickler und legt detailliert die einzelnen Schritte für Entwurf und Entwicklung, die einzuhaltenden Regeln sowie Inhalte von (Zwischen-) Ergebnissen fest. Neben der methodischen Vorgehensweise liefert

eine Entwicklungsmethode auch konkrete Vorschläge für Notationen zur Dokumentation der Entwicklungsergebnisse. Diese können formal bzw. semi-formal sein.
Eine Entwicklungsmethode ist in der Regel in ein generisches Vorgehensmodell eingebettet, das den organisatorischen Rahmen für den Entwicklungsprozess festlegt.

Prozessmuster

Die grundlegende Idee der Prozessmuster ist die Beschreibung eines Vorgehensmodells in Form modularer Einheiten entsprechend einem festgelegten Beschreibungsschema. Die Beschreibung eines Prozessmusters besteht aus einem initialen Kontext, der ein allgemeines, wiederkehrendes Problem in Entwicklungsprojekten beschreibt, einem Vorschlag einer bewährten und allgemeinen Lösung, sowie einem Ergebniskontext als Ergebnis der Anwendung des Prozessmusters. Prozessmuster geben Hilfestellung zu einzelnen Arbeitsschritten, die sich in der Praxis bewährt haben. Sie sind pragmatische Anleitungen für die Abwicklung einzelner Aktivitäten in einem Projekt. Beispielsweise könnte ein Prozessmuster beschreiben, wie man ausgehend von einem Designmodell konstruktiv Testfälle für die Testspezifikation erstellen kann. Ob und wann dieses Prozessmuster ausgeführt wird, ist aber nicht festgelegt, sondern wird bei Bedarf entschieden.

Softwareprozess

Ein konkreter Satz von Tätigkeiten, deren Ziel die Entwicklung oder die Weiterentwicklung von Software ist.
Ein Softwareprozess ist eine Instanz einer Entwicklungsmethode.

Tailoring

Beim so genannten Tailoring werden aus den Elementen des Vorgehensmodells diejenigen ausgewählt, die für das Projekt relevant sind. Das Zurechtschneiden des Vorgehensmodells („Tailoring“) bezüglich der anzuwendenden Prozessmuster beziehungsweise durchzuführenden Aktivitäten wird nicht nur zu Beginn des Projektes durchgeführt. Vielmehr soll dem Projektleiter und dem Projektteam die Möglichkeit gegeben werden, abhängig von der konkreten Projektsituation bestimmte Prozessmuster auszuwählen.

Vorgehensmodell

Ein Vorgehensmodell dient als zentraler Leitfaden und organisatorischer Rahmen für methodische Softwareentwicklung (oder Systementwicklung im allgemeinen), -wartung und -betrieb und kann, muss jedoch nicht, eine bestimmte Entwicklungsmethode vorgeben. Vorgehensmodelle, die sich nicht auf eine konkrete Entwicklungsmethode festlegen, werden auch als generisch bezeichnet.
Im Einzelnen legt ein Vorgehensmodell fest, welche Aktivitäten in welcher Reihenfolge von welchen Personen in ihnen zugewiesenen Rollen durchzuführen sind, um bestimmte Ergebnisse zu erarbeiten. Ziel ist eine übersichtlichere Gestaltung und damit verbesserte Beherrschbarkeit des Entwicklungsprozesses. Dazu unterteilt ein Vorgehensmodell den Entwicklungsprozess in überschaubare, zeitlich und inhaltlich begrenzte Schritte und ordnet diesen die durchzuführenden Aktivitäten zu.
Ein Vorgehensmodell kann auch als ein Baustein der konstruktiven Qualitätssicherung gesehen werden. Idee ist, dass durch eine geordnete und systematische Projektdurchführung die Projektrisiken minimiert und die Qualität des zu entwickelnden Systems positiv beeinflusst werden.
Die Begriffe Vorgehensmodell und Prozessmodell verstehen wir als synonym.

Vorgehens-Meta-Modell

Darunter verstehen wir ein Modell eines Vorgehensmodells. Im Vorgehens-Meta-Modell werden Beschreibungselemente wie Produkt, Aktivität, Rolle und Ablauf sowie strukturelle Beziehungen zwischen diesen Beschreibungselementen definiert. Konsistenzbedingun-

gen schränken die Anwendung der Beschreibungselemente bei der Beschreibung eines Vorgehensmodells ein. Das Vorgehens-Meta-Modell dient dem Prozessingenieur bei der Erstellung und Pflege von Vorgehensmodellen als Sprache, um eine einheitliche Beschreibung eines Vorgehensmodells zu erreichen.

2 Kriterien zur Bewertung und Einteilung von Vorgehensmodellen

Je nach Art des Projektes können unterschiedliche Vorgehensmodelle vorteilhaft sein. In diesem Kapitel wollen wir eine Reihe von Kriterien aufstellen, mit denen sich Vorgehensmodelle bewerten, vergleichen und hinsichtlich ihrer jeweiligen Stärken und Schwächen analysieren lassen.

Enge der Projektvorgaben

Darunter verstehen wir die Menge und die Verbindlichkeit der Vorgaben an das Projekt und an die zu entwickelnden Produkte.
Unter anderem wird hierbei betrachtet ob das Vorgehensmodell geeignet ist, wenn:

- genaue Zeit- und Kostenvorgaben eingehalten werden müssen
- ein großes und/oder verteiltes Projekt vorliegt
- ein kleines oder mittleres Projekt vorliegt
- Time to Market äußerst kritisch ist
- das System sicherheitskritisch ist

Formale Systementwicklung bietet beispielsweise Vorteile bei besonders sicherheitskritischen Projekten. Probleme treten jedoch aufgrund der aufwändigen Vorgehensweise bei zeitkritischen Projekten auf. Extreme Programming hingegen ist gut geeignet für Projekte, bei denen Time To Market kritisch ist. Hierbei können jedoch nicht in dem Maße, wie bei der formalen Systementwicklung, Garantien hinsichtlich geforderter Sicherheitsaspekte gegeben werden.

Umfang

Dieses Kriterium beleuchtet die Phasen- und Prozessabdeckung von Vorgehensmodellen. Die Phasenabdeckung betrifft den zeitlichen Verlauf des Systemlebenszyklus, während die Prozessabdeckung die Unterstützung von Querschnittsfunktionen wie Projektmanagement und Qualitätssicherung behandelt.
Wir ziehen für die Untersuchung der Phasenabdeckung wie Noack und Schienmann (1999) fünf Phasen heran:

- Voruntersuchung
- Systemanalyse
- Entwurf
- Erzeugung
- Einführung

Angelehnt an den Ansatz von Noack und Schienmann (1999) untersuchen wir hinsichtlich der Prozessabdeckung fünf Prozesse:

- Entwicklung
- Test
- Projektmanagement
- Qualitätssicherung
- Änderungs- und Konfigurationsmanagement

Im Falle von generischen Vorgehensmodellen sind einzelne Phasen und Teilprozesse nicht abgedeckt. Dies liegt daran, dass ihre Beschreibung sich auf einem zu hohen Abstraktionsniveau befindet. Konkrete Entwicklungsmethoden eines Projektes, die auf dem jeweiligen generischen Vorgehensmodell aufbauen, müssen diese Lücken füllen und damit alle Phasen und Teilprozesse abdecken.

Abstraktionsgrad

Angelehnt an Humphrey (1989) unterscheiden wir drei Niveaus von Vorgehensmodellen:

- **Universelle Vorgehensmodelle** beschreiben nur prinzipielle Schritte, geben also nur allgemeine Richtlinien, wie z.B. eine globale Ablaufstruktur.
- **Weltliche Vorgehensmodelle** beschreiben eine Abfolge von Schritten detailliert genug, dass sie direkt durchführbar sind.
- **Atomare Vorgehensmodelle** sind am detailliertesten. Sie geben ausführliche Beschreibungen von Aktivitäten und Entwicklungsprodukten.

Erfüllung von Anforderungen

Erstens beachten wir an dieser Stelle, inwieweit Vorgehensmodelle das Team unterstützen können, die tatsächlichen Anforderungen an das Produkt zu erfüllen. Man beachte, dass an dieser Stelle von den tatsächlichen Anforderungen und nicht den wahrgenommenen die Rede ist. Oft sind die tatsächlichen Anforderungen nicht bekannt oder werden erst spät im Projekt erkannt.
Zweitens führen wir auf, wie gut Vorgehensmodelle mit sich ändernden Anforderungen umgehen können.

Anpassbarkeit

Unter Anpassbarkeit verstehen wir die Fähigkeit eines Vorgehensmodells, sich an den Kontext eines Projektes anzupassen. Beachtung findet hierbei insbesondere die Möglichkeit des Tailorings sowie die Fähigkeit auf einen sich ändernden Projektkontext zu reagieren.
Chroust (1992, 45) unterscheidet in diesem Zusammenhang drei Grundtypen von Vorgehensmodellen:

- Das **Dach-Modell** enthält Beschreibungen zu sämtlichen notwendigen Aktivitäten und Entwicklungsprodukten. Die Anpassung erfolgt durch Entfernen von Teilen des Modells.
- Das **Kern-Modell** enthält nur Beschreibungen zu den wichtigsten Typen von Aktivitäten und Entwicklungsprodukten. Die Anpassung erfolgt durch Ergänzung der Vorgaben.
- Das **Modulare Bausteinmodell** besteht aus einer Reihe von lose gekoppelten Teilmodellen. Die Anpassung erfolgt durch Zusammenstellung von Teilmodellen.

Philosophie

An dieser Stelle betrachten wir Eigenschaften wie das zugrunde liegende Paradigma, das primäre Ziel und das antreibende Moment des Vorgehensmodells.

Prozesssteuerung

Die Prozesssteuerung ist entweder

- Aktivitätsorientiert oder
- Ergebnisorientiert.

Benutzerbeteiligung

Der Grad an Benutzerbeteiligung ist bei den unterschiedlichen Vorgehensmodellen sehr variabel. Ebenso ist der Zeitpunkt, an dem die Benutzer oder Kunden in das Projekt integriert werden, verschieden.

3 Vorgehensmodelle

Unter einem Vorgehensmodell verstehen wir einen standardisierten organisatorischen Rahmen für den idealen Ablauf eines Entwicklungsprojektes (Gnatz 2005). Vorgehensmodelle werden in Form zu erstellender Produkte, durchzuführender Aktivitäten und zu besetzender Rollen beschrieben. Ein Vorgehensmodell ist damit ein Modell beziehungsweise eine Abstraktion einer möglichen Vorgehensweise im konkreten Projekt. Vorgehensmodelle abstrahieren von projektspezifischen Vorgehensweisen mit dem Ziel einer breiteren Einsetzbarkeit der Modelle für unterschiedliche Arten von Projekten.
Der Nutzen der Anwendung von Vorgehensmodellen kann allgemein in Effizienzsteigerungen gesehen werden. Ein Vorgehensmodell kann als Lektüre verstanden werden, die zum Nachdenken über das eigene Vorgehen anregt. Die bloße Beschäftigung der Projektmitarbeiter mit der Materie kann bereits in einer verbesserten Effizienz resultieren. Wird ein ingenieurmäßiges Vorgehen angestrebt, muss ein Vorgehensmodell aber eher Regelungscharakter als anregenden Charakter haben. Zu diesem Zweck muss die Anwendung eines Vorgehensmodells im Projekt klar vorgegeben sein.
Die Definition des Begriffs Vorgehensmodell als Abstraktion einer idealen Vorgehensweise bedingt eine große Anzahl möglicher Ausprägungen konkreter Vorgehensmodelle. Tatsächlich existiert heute eine Vielzahl unterschiedlicher Vorgehensmodelle. Neben Normen, Standards und Konventionen verfügt jedes größere Unternehmen über ein oder mehrere eigene hausinterne Vorgehensmodelle. Mit dem Einsatz eines Vorgehensmodells werden im Allgemeinen die folgenden Ziele verfolgt (Gnatz, 2005):

- Ein Vorgehensmodell soll durch die Definition einheitlicher Begriffe und Ergebnisse zur Verbesserung der Kommunikation der Projektbeteiligten beitragen.
- Ein standardisiertes Vorgehen soll die Vollständigkeit und Qualität der zu liefernden Ergebnisse gewährleisten.
- Ein Vorgehensmodell zielt in mannigfaltiger Weise auf eine Eindämmung der Entwicklungskosten beziehungsweise der Kosten über den gesamten Lebenszyklus eines Systems ab.
- Ein einheitlich verwendetes Vorgehensmodell stellt die Nachvollziehbarkeit und Wiederholbarkeit des Vorgehens und Prozesses sicher.

3.1 Universelle Vorgehensmodelle

3.1.1 Das Wasserfallmodell

Die einzelnen Phasen des Wasserfallmodells entsprechen den fundamentalen Entwicklungs-Aktivitäten:

1. Analyse und Definition der Anforderungen
2. System- und Softwareentwurf
3. Implementierung und Komponententest
4. Integration und Systemtest
5. Betrieb und Wartung

Das Ergebnis einer jeden Phase ist eine Reihe von Artefakten, die in der nächsten Phase weiterverarbeitet werden. Am Ende einer Phase können deren Ergebnisse Qualitätssicherungsmaßnahmen unterzogen werden.
Mit der nächsten Phase soll erst begonnen werden, wenn die vorige abgeschlossen ist. In der Praxis überlappen die Phasen sich jedoch häufig. Während des Designs werden oft Probleme mit den Anforderungen festgestellt, während der Implementierung Probleme mit dem Design, usw. Daher werden oft Schleifen eingefügt, die es erlauben abgeschlossene Phasen erneut auszuführen. Um die Kosten, die durch die vielfache Erstellung und Ab-

nahme von Dokumenten entstehen, einzudämmen, werden nach einigen wenigen Iterationen bestimmte Phasen eingefroren und man fährt mit den übrigen Phasen fort. Dies führt aber beispielsweise dazu, dass schlechte Designs durch „Tricks“ in der Implementierung ausgeglichen werden. Bestimmte Fehler und Auslassungen werden erst in der letzten Phase entdeckt. Das System muss daher weiterentwickelt werden, was die erneute Ausführung vorausgegangener Phasen mit sich bringt.

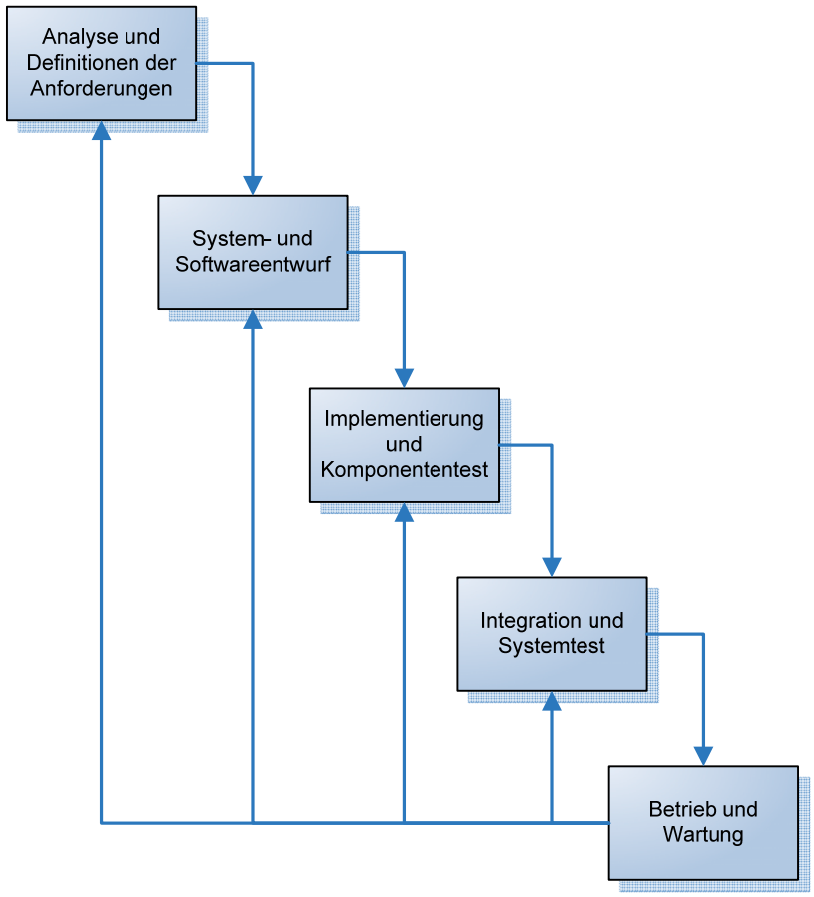


Abbildung 2: Wasserfallmodell (aus Sommerville 2001)

Vorteile

- In jeder Phase wird Dokumentation erstellt.
- Das Modell ist mit Vorgehensmodellen anderer Disziplinen kompatibel. Dies bietet Vorteile, wenn das Softwareentwicklungsprojekt Teil eines Projektes ist, das nicht nur die Entwicklung von Software umfasst (Sommerville 2001, 58).
- Verträge können die Spezifikation des Systems als Grundlage verwenden.
- Das Wasserfallmodell gibt einen klaren, aus Sicht des Projektmanagements beziehungsweise des Auftraggebers einfach zu verstehenden, linearen und gut kontrollierbaren Ablauf vor.

Nachteile

- Hauptnachteil ist die unflexible Aufteilung des Projektes in diskret sequenzialisierte, gegeneinander abgegrenzte Phasen. Schon früh müssen weit reichende Festlegungen bei geringem Kenntnisstand gemacht werden, was den Umgang mit sich ändernden Anforderungen schwierig gestaltet (Sommerville 2001, 58).

- Projekte folgen üblicherweise nicht dem sequenziellen Fluss, den das Modell vorsieht. Iteratives Vorgehen ist zwar möglich, jedoch nicht explizit angestrebt. Dies hat zur Folge, dass Änderungen Verwirrung hervorrufen (Pressman 2001, 30).
- Zu Beginn eines Projektes sind die Anforderungen oft noch unklar und können nicht vollständig und präzise formuliert werden (Models for Action Project 1998).
- Der Kunde bekommt erst sehr spät im Projekt eine lauffähige Version des Systems. Grobe Fehler werden unter Umständen erst während des Einsatzes entdeckt (Pressman 2001, 30).
- Ein einmal festgelegter Entwicklungsablauf wird starr eingehalten, ohne dass beispielsweise Projekt- und Entwicklungsrisiken berücksichtigt werden.
- Das Risiko der erst spät erkannten Mängel des Entwicklungsgegenstands kann zu gravierenden Änderungsforderungen führen. Ein fester Auslieferungstermin kann womöglich nicht eingehalten werden (Bunse und von Knethen 2002, 5f).
- Durch die Sequenzialisierung des Prozesses kann es vorkommen, dass Projektmitglieder darauf warten müssen, dass andere ihre Aufgaben beenden bevor sie mit ihrer Arbeit beginnen können (Bradac et al. 1994).
- Hohe Erfahrung mit den Entwicklungstechniken und der Abschätzung von Projektkosten wird benötigt. Anders als bei iterativen Ansätzen kann hier nicht von den gewonnenen Erfahrungen früherer Iterationen profitiert werden (Bunse und von Knethen 2002, 6).
- Wenn Probleme im Entwurf durch Tricks in der Implementierung umgangen werden, kann dies zu einem schlecht strukturierten System führen (Sommerville 2001, 58).
- Es besteht die Gefahr, dass die Dokumentation wichtiger wird als das eigentliche System (Balzert 1998, 101).

Enge der Projektvorgaben

Die Methode bereitet Probleme bei Projekten, die genaue Vorgaben bezüglich Auslieferungstermin und Budget machen, da diese nicht garantiert eingehalten werden können.
Das Projektteam muss Erfahrung mit den Entwicklungstechniken und der Abschätzung von Projektkosten haben. Anders als bei iterativen Ansätzen kann hier nicht von den gewonnenen Erfahrungen früherer Iterationen profitiert werden (Bunse und von Knethen 2002, 6).

Umfang

Phasenabdeckung	
Voruntersuchung	nicht abgedeckt
Systemanalyse	abgedeckt
Entwurf	abgedeckt
Erzeugung	abgedeckt
Einführung	abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	nicht abgedeckt
Qualitätssicherung	nicht abgedeckt
Änderungs- und Konfigurationsmanagement	nicht abgedeckt

Abstraktionsgrad

Das Wasserfallmodell ist ein universelles Vorgehensmodell.

Erfüllung von Anforderungen

Aufgrund der sequenziellen Ausführung können sich ändernde Anforderungen nur schwer berücksichtigt werden. Die Anforderungen werden zu Beginn des Projektes vollständig spezifiziert. Falls zu Beginn die tatsächlichen Anforderungen nicht bekannt oder noch unklar sind können sie nicht vom Produkt erfüllt werden.
Das Wasserfallmodell ist nur bei Vorliegen bekannter, stabiler Anforderungen sinnvoll einsetzbar.

Anpassbarkeit

Das Wasserfallmodell gibt einen klaren, leicht zu verstehenden und gut kontrollierbaren Ablauf vor. Ein einmal festgelegter Entwicklungsablauf wird starr eingehalten. „Dies birgt den Nachteil, dass lang andauernde Projekte nicht angemessen auf ein sich veränderndes Projektumfeld reagieren können. Die aktive Beobachtung des Projektumfelds tritt in den Hintergrund.“ (Gnatz 2005, 24)
Ein Zuschneiden des Modells auf spezielle Projekttypen ist nicht möglich.

Philosophie

Das Wasserfallmodell ist ein sequenzielles Modell. Sein primäres Ziel ist es, den Managementaufwand zu minimieren. Antreibendes Moment sind Dokumente.

Prozesssteuerung

Das Wasserfallmodell ist ergebnisorientiert.

Benutzerbeteiligung

Benutzer werden nur zu Beginn des Projektes und bei Auslieferung des Produktes beteiligt. Daher können sie keinen Einfluss auf das laufende Projekt ausüben. Fehlentwicklungen werden daher unter Umständen erst während des Einsatzes entdeckt.

3.1.2 Evolutionäre Entwicklung

Oftmals ist eine vollständige Ermittlung der Anforderungen zu Beginn des Projektes, wie beim Wasserfallmodell gefordert, nicht möglich. Anforderungen können auch erst im Laufe des Projektes auftauchen oder sich ändern.
Getrieben von dieser Erkenntnis entstand die Idee der evolutionären Entwicklung. Das System wird dabei stufenweise in einer Vielzahl von Iterationen entwickelt. So wird beginnend mit einem Kernsystem eine Reihe von Versionen des Systems realisiert und ausgeliefert. Auf diese Weise können Auftraggeber und Anwender Erfahrungen mit dem System sammeln. Die Weiterentwicklung orientiert sich an ihrem Feedback. Auf diese Weise werden die Anforderungen an das System im Laufe des Projektes ermittelt.
Nach Sommerville (2001, 60) ist der evolutionäre Ansatz für kleine und mittlere Systeme (bis zu 500.000 Zeilen Programmcode) der beste. Seine Nachteile sind bei großen, komplexen und langlebigen Systemen, bei denen eine Vielzahl an Teams mitarbeitet, besonders gravierend. In diesem Fall empfiehlt Sommerville (2001, 60) einen gemischten Ansatz, „der die besten Eigenschaften des Wasserfall-Modells und des evolutionären Entwicklungsmodells vereinigt.“

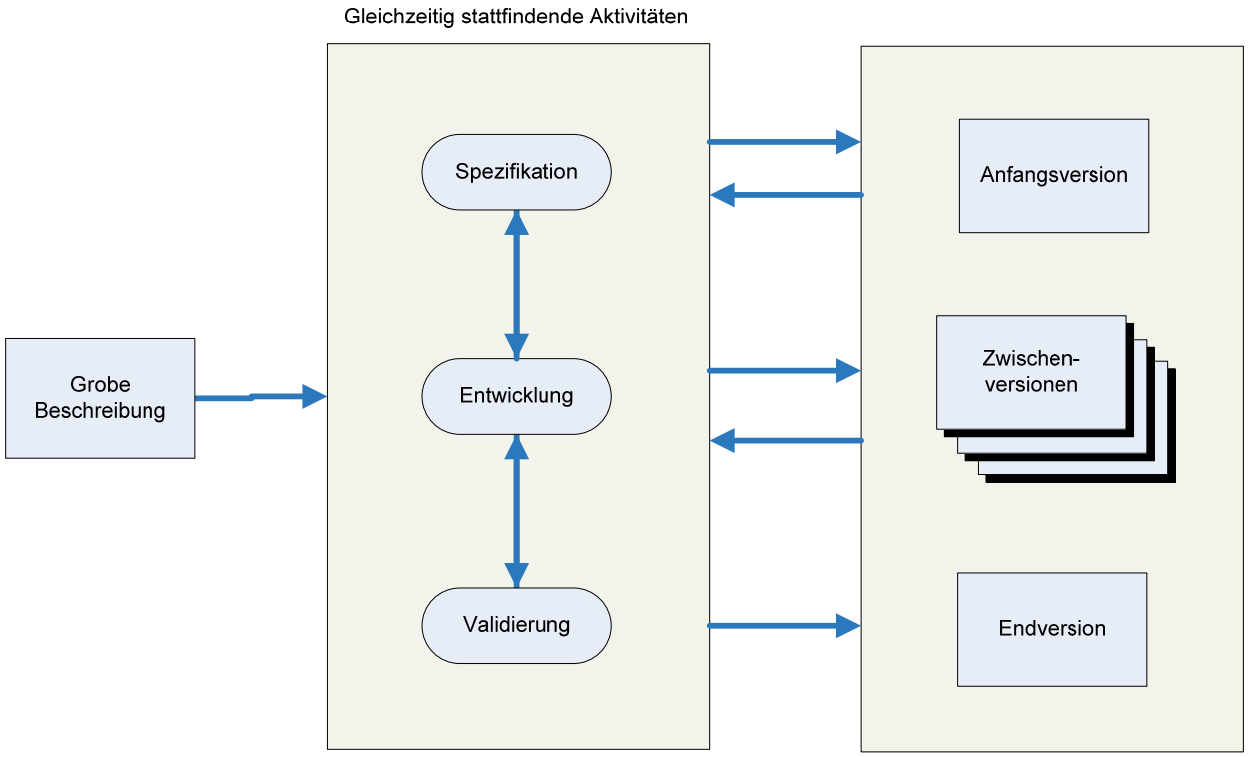


Abbildung 3: Evolutionäre Entwicklung (aus Sommerville 2001)

Vorteile

- Durch die inkrementelle Entwicklung der Spezifikation gewinnen Benutzer und Entwicklungsteam ein besseres Verständnis des Problems (Sommerville 2001, 59).
- Die Erfolgsquote, was die Erfüllung der Bedürfnisse des Kunden anbetrifft, ist höher als beim Wasserfallmodell (Sommerville 2001, 59).
- Der Ansatz kommt bis zu einem gewissen Punkt auch mit Anforderungen, die sich während der Projektlaufzeit ändern oder erst spät im Projekt auftauchen, zurecht (Bunse und von Knethen 2002, 14).
- Wenn Time to Market sehr kritisch ist, kann eine komplette Version oft nicht schnell genug geliefert werden. Evolutionäre Ansätze erlauben es jedoch, rechtzeitig zumindest eine eingeschränkte Version zu realisieren (Bunse und von Knethen 2002, 15).
- Das Team kann über die Inkremente hinweg lernen. Daher kann der Ansatz auch eingesetzt werden, wenn zu Beginn des Projektes die Erfahrung des Teams mit den verwendeten Entwicklungstechniken gering ist (Bunse und von Knethen 2002, 15). Nicht zuletzt wird dadurch das Risiko gesenkt.
- Integrationstests werden vereinfacht, da nicht alles gleichzeitig integriert und getestet wird (Bunse und von Knethen 2002, 14f).
- Die Entwicklung eines Systems wird im Vergleich zum Wasserfallmodell in kleinere Einheiten überschaubarer Größe aufgeteilt. Dadurch ist es möglich, den Projektablauf an die im Projektverlauf gewonnenen Erfahrungen anzupassen beziehungsweise immer wieder neu zu definieren.

Nachteile

- Es besteht die Gefahr, dass frühe Versionen nicht flexibel genug sind, um sich an ungeplante Evolutionspfade anzupassen (Balzert 1998, 122).

- Der Fortschritt ist schwerer zu erkennen (Sommerville 2001, 59), d.h. aus Sicht des Managements geht bei der evolutionären Entwicklung die langfristige Planbarkeit eines Projektes verloren, da es bei nicht festgelegter Anzahl von Iterationen auch nicht möglich ist, das Projektende abzusehen.
- Aus technischer Sicht besteht die Gefahr, dass aufgrund neu hinzugekommener Anforderungen die Architektur des Systems überarbeitet werden muss.
- Das System ist aufgrund stetiger Veränderungen oft schlecht strukturiert (Sommerville 2001, 59).
- Die Benutzer müssen während des gesamten Projektes beteiligt werden (Models for Action Project 1998).
- Es entsteht ein erhöhter Aufwand bzgl. Kommunikation und Koordination. Dies stellt insbesondere dann ein Problem dar, wenn das System verteilt entwickelt wird (Models for Action Project 1998).
- Anforderungen müssen sich in Teilmengen aufteilen lassen, die möglichst wenig Abhängigkeiten untereinander besitzen (Bunse und von Knethen 2002, 14).
- Das Management beobachtet den Projektfortschritt normalerweise über die erstellten Dokumente. Da inkrementell entwickelte Systeme sich jedoch schnell ändern, ist die Erstellung umfangreicher Dokumentation oft nicht kosteneffizient (Sommerville 1989, 393).
- Normalerweise basieren Verträge auf einer Spezifikation des Systems. Diese existiert bei der iterativen Entwicklung zu Beginn jedoch nicht (Sommerville 1989, 394).
- Verifikation und Validation sind normalerweise darauf ausgerichtet, zu zeigen, dass das System seiner Spezifikation entspricht. Iterative Ansätze versuchen jedoch Dokumentation zu minimieren und überlappen Spezifikation und Entwicklung (Sommerville 1989, 394).
- Bei sicherheitskritischen Systemen ist es von Vorteil, wenn man zu Beginn alle Anforderungen im Hinblick auf Interaktionen, die die Sicherheit gefährden, untersuchen kann. Bei iterativer Entwicklung kennt man zu Beginn jedoch unter Umständen noch nicht alle Anforderungen. Eine solche Analyse ist daher nur erschwert möglich (Sommerville 1989, 394).

Enge der Projektvorgaben

Evolutionäre Entwicklung verursacht einen erhöhten Aufwand hinsichtlich Kommunikation und Koordination. Bei mehr Mitarbeitern fällt dabei ein überproportionaler Mehraufwand an. Bei besonders großen und verteilten Projekten ergeben sich daher Probleme.
Bei sicherheitskritischen Systemen ist es von Vorteil, wenn man zu Beginn alle Anforderungen im Hinblick auf Interaktionen, die die Sicherheit gefährden, untersuchen kann. Da dies bei der Evolutionären Entwicklung nicht möglich ist, bringt das Modell in dieser Hinsicht Nachteile mit sich.
Evolutionäre Ansätze bieten Vorteile wenn Time to Market kritisch ist, da sie es erlauben zumindest eine eingeschränkte Version (Demonstrator, Prototyp) rechtzeitig auszuliefern.
Der Ansatz kann auch eingesetzt werden wenn das Projektteam wenig Erfahrung mit den verwendeten Entwicklungstechniken besitzt, da es im Laufe der Iterationen hinzulernen kann.

Umfang

Phasenabdeckung	
Voruntersuchung	nicht abgedeckt
Systemanalyse	abgedeckt
Entwurf	abgedeckt
Erzeugung	abgedeckt
Einführung	abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	nicht abgedeckt
Qualitätssicherung	nicht abgedeckt
Änderungs- und Konfigurationsmanagement	nicht abgedeckt

Abstraktionsgrad

Evolutionäre Entwicklung ist ein universelles Vorgehensmodell.

Erfüllung von Anforderungen

Durch die iterative Entwicklung der Spezifikation ist es den Benutzern und Entwicklern besser möglich, die Anforderungen zu verstehen und die tatsächlichen Anforderungen zu erkennen. Die Bedürfnisse der Benutzer werden daher mit einer höheren Wahrscheinlichkeit als beim Wasserfallmodell erfüllt.
Zudem kann der Ansatz auf neu auftauchende oder sich ändernde Anforderungen reagieren. Die Richtung der Entwicklung lässt sich in dem Fall korrigieren. Somit ist das Modell auch bei unstabilen Anforderungen problemlos einsetzbar.

Anpassbarkeit

Die Entwicklung eines Systems ist in kleine Einheiten überschaubarer Größe aufgeteilt. Dies ermöglicht es, dass die Entwicklung an einen sich ändernden Kontext des Projektes angepasst werden kann.
Ein Zuschneiden des Modells auf spezielle Projekttypen ist nicht möglich.

Philosophie

Evolutionäre Entwicklung ist ein iteratives Modell. Das primäre Ziel ist es, die Entwicklungszeit zu minimieren. Das antreibende Moment ist der Code.

Prozesssteuerung

Die Evolutionäre Entwicklung ist aktivitätsorientiert.

Benutzerbeteiligung

Die Benutzer werden während des gesamten Projektes immer wieder am Projekt beteiligt.

3.1.2.1 Explorative Entwicklung

Manchmal ist es sehr schwierig, oder sogar unmöglich, die Anforderungen zu Beginn des Projektes zu identifizieren. In diesem Fall bietet sich die explorative Entwicklung an.
Das Ziel ist es, durch enge Zusammenarbeit mit den Kunden deren Anforderungen herauszufinden und ein möglichst vollständiges System zu entwickeln. Dabei beginnt die Entwicklung mit den Teilen des Systems, die schon spezifiziert werden können. Das System wird allmählich im Laufe mehrerer Iterationen den Vorschlägen der Kunden entsprechend erweitert.

Ein typisches Merkmal der explorativen Entwicklung ist das Fehlen von präzisen Spezifikationen. Validation geschieht nur durch Betrachtung der Eignung des Endergebnisses ohne Überprüfung der Einhaltung von vorher aufgestellten Anforderungen.

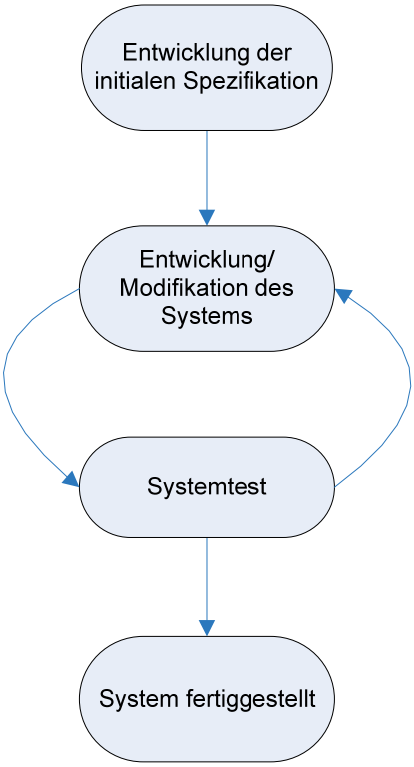


Abbildung 4: Explorative Entwicklung

Vorteile

- Der Ansatz ist auch dann anwendbar, wenn Anforderungen nicht oder nur sehr schwer zu Beginn des Projektes identifiziert werden können (Models for Action Project 1998). Zum Beispiel können durch die einfache Generierung von UI-Prototypen Anforderungen und Zielvorstellungen in frühen Phasen bereits konkret diskutiert werden.

Nachteile

- Der Ansatz ist nur geeignet für High-Level Sprachen, die eine schnelle Entwicklung erlauben, wie z.B. LISP (Models for Action Project 1998).
- Die Kosteneffizienz ist schwer abzuschätzen und zu messen (Models for Action Project 1998).
- Das Design des Systems ist oft von minderer Qualität (Models for Action Project 1998).

Enge der Projektvorgaben

Wie auch die evolutionäre Entwicklung verursacht die explorative Entwicklung einen erhöhten Aufwand an Kommunikation und Koordination. Daher ist sie bei besonders großen und verteilten Projekten, bei denen die Kommunikation bereits durch die Projektkomplexität erschwert wird, weniger günstig.

Für sicherheitskritische Systeme ist sie ungeeignet, da diese eine spezifische Betrachtung und Planung von Sicherheitsaspekten benötigen.
Analog zur evolutionären Entwicklung bietet das Modell Vorteile bei zeitkritischen Projekten, sowie dann, wenn das Projektteam nur wenig Erfahrung mit den verwendeten Entwicklungstechniken besitzt.
Explorative Entwicklung ist nur geeignet für High-Level Sprachen, die eine schnelle Entwicklung erlauben (Models for Action Project 1998).

Umfang

Phasenabdeckung	
Voruntersuchung	nicht abgedeckt
Systemanalyse	Abgedeckt
Entwurf	Abgedeckt
Erzeugung	Abgedeckt
Einführung	nicht abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	nicht abgedeckt
Qualitätssicherung	nicht abgedeckt
Änderungs- und Konfigurationsmanagement	nicht abgedeckt

Abstraktionsgrad

Explorative Entwicklung ist ein universelles Vorgehensmodell.

Erfüllung von Anforderungen

Explorative Entwicklung ist speziell für Projekte ausgelegt bei denen die Anforderungen an das System zu Beginn des Projektes kaum bekannt sind. Sich ändernde und im Laufe des Projektes auftauchende Anforderungen sind der Normalfall bei explorativer Entwicklung. Das Modell ist speziell für instabile Anforderungen gedacht.

Anpassbarkeit

Die iterative Entwicklung erlaubt es, auf einen sich ändernden Kontext des Projektes zu reagieren.
Ein Zuschneiden des Modells auf spezielle Projekttypen ist nicht möglich.

Philosophie

Explorative Entwicklung ist ein iteratives Modell. Das primäre Ziel ist es, unklare Anforderungen zu verstehen. Das antreibende Moment ist der Code.

Prozesssteuerung

Die Explorative Entwicklung ist aktivitätsorientiert.

Benutzerbeteiligung

Das Modell sieht eine enge Zusammenarbeit mit dem Kunden während des gesamten Projektes vor.

3.1.2.2 Inkrementelle Entwicklung

Die inkrementelle Entwicklung ist ein Ansatz, der zwischen dem Wasserfallmodell und der evolutionären Entwicklung liegt. Sie versucht die langfristige Planbarkeit des Wasserfallmodells mit dem durch frühzeitige Auslieferungen gewonnenen Feedback zu kombinieren.

Bei der inkrementellen Entwicklung wird zunächst eine Grobarchitektur aufgestellt. Diese umfasst eine skizzenhafte Darstellung aller Anforderungen, die das System erfüllen soll. Jeder dieser Anforderungen wird eine bestimmte Priorität zugewiesen. Daraufhin wird eine Anzahl von Iterationen geplant. Jede Iteration implementiert einen Teil der Anforderungen. Die Anforderungen mit der höchsten Priorität werden dabei als erstes realisiert. Sobald die Inkremente festgelegt sind, definiert man detailliert die Anforderungen, die im ersten Inkrement implementiert werden sollen. Die Anforderungsanalyse wird während den Iterationen fortgesetzt, mit der Einschränkung, dass die Anforderungen des laufenden Inkrements nicht geändert werden dürfen. Sobald ein Inkrement fertig gestellt und ausgeliefert wurde, kann es von den Anwendern eingesetzt werden. Diese können so mit dem System experimentieren. Auf diese Weise erhalten sie Kenntnisse, die ihnen bei der weiteren Identifikation von Anforderungen helfen. Neue Inkremente werden in das System integriert, so dass dessen Funktionalität schrittweise erweitert wird.

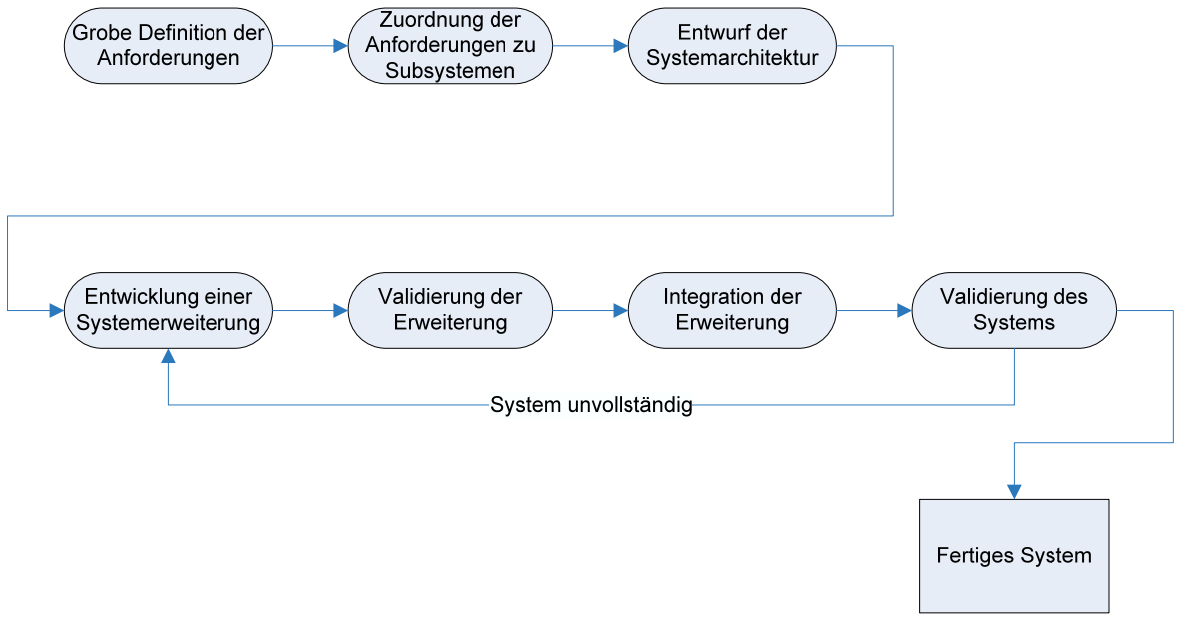


Abbildung 5: Inkrementelle Entwicklung (aus Sommerville 2001)

Vorteile

- Die Kunden erhalten schon frühzeitig Teile des Systems und können damit schon früh Wert schöpfen (Sommerville 2001, 64).
- Die Kunden können die ersten Inkremente als Prototypen benutzen, mit denen sie Erfahrungen sammeln können, die ihnen bei der Aufstellung von Anforderungen helfen (Sommerville 2001, 65).
- Das Risiko des vollständigen Scheiterns des Projektes wird vermindert. Selbst wenn einige Inkremente scheitern, können die anderen den Kunden immer noch von Nutzen sein (Sommerville 2001, 65).
- Da die Anforderungen mit der höchsten Priorität zuerst umgesetzt werden, werden sie im Verlaufe des Projektes auch am meisten geprüft. Daher finden sich in den wichtigsten Teilen des Systems die wenigsten Fehler (Sommerville 2001, 65).
- Inkrementelle Entwicklung erlaubt es, zu Beginn des Projektes mit weniger Entwicklern zu arbeiten. Wenn das System auf Zustimmung stößt, können bedarfsweise für spätere Inkremente zusätzliche Mitarbeiter eingestellt werden (Pressman 2001, 36).

- Technische Risiken können gemindert werden. Wenn man z.B. auf eine bestimmte Hardwarekomponente warten muss, deren Fertigstellung ungewiss ist, können frühe Inkremente so geplant werden, dass man sich zunächst Bereichen widmet, die unabhängig von der Hardware sind. So lässt sich der Zeitpunkt, an dem man mit den Teilen beginnt, die die Hardware benötigen, flexibel gestalten (Pressman 2001, 36).
- Falls die Architektur im Laufe des Projektes relativ stabil bleibt, bietet die inkrementelle Entwicklung Vorteile gegenüber der evolutionären Entwicklung. Dies liegt darin begründet, dass die Gesamtarchitektur des Systems zu Beginn wenigstens in groben Zügen festgelegt wird.

Nachteile

- Da die Inkremente eher klein sein sollen, kann es schwierig sein die Anforderungen der Kunden Inkrementen angemessener Größe zuzuordnen (Sommerville 2001, 65). Insbesondere bei Änderungen am System können transitiv weitere Änderungen entstehen, die geplante Volumina von Inkrementen übersteigen.
- Die meisten Systeme beinhalten Funktionen, die von vielen Bestandteilen des Systems benutzt werden. Da die Anforderungen immer erst dann detailliert spezifiziert werden, wenn sie im nächsten Inkrement realisiert werden sollen, ist es schwierig diese gemeinsam benutzten Funktionen zu identifizieren (Sommerville 2001, 65).

Enge der Projektvorgaben

Bei sicherheitskritischen Systemen ist es von Vorteil, wenn man zu Beginn alle Anforderungen im Hinblick auf Interaktionen, die die Sicherheit gefährden, untersuchen kann. Da dies bei der inkrementellen Entwicklung nur eingeschränkt möglich ist, bringt das Modell in dieser Hinsicht Nachteile mit sich.
Die inkrementelle Entwicklung bietet Vorteile wenn Time to Market kritisch ist, da sie es erlaubt, zumindest eine eingeschränkte Version rechtzeitig auszuliefern. Der Ansatz kann auch eingesetzt werden wenn das Projektteam wenig Erfahrung mit den verwendeten Entwicklungstechniken besitzt, da es im Laufe der Iterationen hinzulernen kann.

Umfang

Phasenabdeckung	
Voruntersuchung	Nicht abgedeckt
Systemanalyse	Abgedeckt
Entwurf	Abgedeckt
Erzeugung	Abgedeckt
Einführung	Abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	nicht abgedeckt
Qualitätssicherung	nicht abgedeckt
Änderungs- und Konfigurationsmanagement	nicht abgedeckt

Abstraktionsgrad

Inkrementelle Entwicklung ist ein universelles Vorgehensmodell.

Erfüllung von Anforderungen

Durch die inkrementelle Entwicklung der Spezifikation ist es den Benutzern und Entwicklern besser möglich die tatsächlichen Anforderungen zu erkennen. Unklare Anforderungen

werden durch frühes Feedback aus den ersten Inkrementen klarer; Lücken können erkannt und geschlossen werden. Die Bedürfnisse der Benutzer werden daher mit einer höheren Wahrscheinlichkeit als beim Wasserfallmodell erfüllt.
Zudem kann der Ansatz auf neu auftauchende oder sich ändernde Anforderungen reagieren. Die Richtung der Entwicklung lässt sich in dem Fall korrigieren. Somit ist das Modell auch bei instabilen Anforderungen problemlos einsetzbar.

Anpassbarkeit

Die Entwicklung eines Systems ist in kleine Einheiten überschaubarer Größe aufgeteilt. Dies ermöglicht es, dass die Entwicklung an einen sich ändernden Kontext des Projektes angepasst werden kann.
Ein Zuschneiden des Modells auf spezielle Projekttypen ist nicht möglich.

Philosophie

Inkrementelle Entwicklung ist ein iteratives Modell. Das primäre Ziel ist es, die Entwicklungszeit und das Risiko zu minimieren. Das antreibende Moment ist der Code.

Prozesssteuerung

Die Inkrementelle Entwicklung ist aktivitätsorientiert.

Benutzerbeteiligung

Die Benutzer werden während des gesamten Projektes immer wieder am Projekt beteiligt.

3.1.2.3 Spiralförmige Entwicklung

Das Spiralmodell geht auf Boehm (Boehm, 1988) zurück. Der Software Prozess wird als Spirale dargestellt. Jede Windung entspricht einer Phase des Softwareprozesses. So kann beispielsweise in der ersten Windung die Machbarkeit des Systems untersucht werden, in der nächsten die Systemanforderungen aufgestellt werden, in der folgenden der Systementwurf, etc.
Jede Windung ist in vier Sektoren unterteilt, die im Projekt iterativ zu durchlaufen sind:

1. **Ziele aufstellen**: Hierbei werden Ziele für die aktuelle Phase definiert, die Randbedingungen des Prozesses und des Produktes identifiziert und ein detaillierter Managementplan aufgestellt. Risiken werden identifiziert und alternative Möglichkeiten bezüglich der Realisierung des Systems abgewogen – beispielsweise unterschiedliche Architekturentwürfe, Wiederverwendung oder der Einsatz von Fertigprodukten.
2. **Risiken einschätzen und verringern**: Es findet eine detaillierte Analyse jedes Projektrisikos statt. Maßnahmen zur Risikominderung werden eingeleitet.
3. **Entwicklung und Validierung**: Für die Durchführung dieser Phase wird ein im Hinblick auf die identifizierten Risiken geeignetes Modell für die Entwicklung gewählt (Anwendung des Wasserfallmodells, des inkrementellen Modells, des evolutionären Modells oder auch eine Kombination dieser Modelle). Damit enthält das Spiralmodell „andere Vorgehensmodelle als Spezialfälle und kann somit als eine Art Meta-Vorgehensmodell verstanden werden. Dabei formuliert Boehm bereits Kriterien für die Auswahl eines oder die Kombination mehrerer dieser Modelle abhängig von bestimmten Projektmerkmalen“ (Gnatz 2005, 24).
4. **Planung**: Das Projekt wird einem Review unterzogen und es wird entschieden, ob noch eine weitere Schleife durchlaufen werden soll. Falls das Projekt fortgesetzt wird, wird an dieser Stelle die nächste Phase des Projektes geplant. Kosten- und Terminplan werden angepasst. Zusätzlich passt der Projektmanager die geplante Anzahl verbleibender Iteration an.

Konfigurationsmanagement und Qualitätssicherung werden kontinuierlich während des gesamten Projektes durchgeführt. Der Hauptunterschied zu anderen Modellen liegt in der expliziten Betrachtung von Risiken.

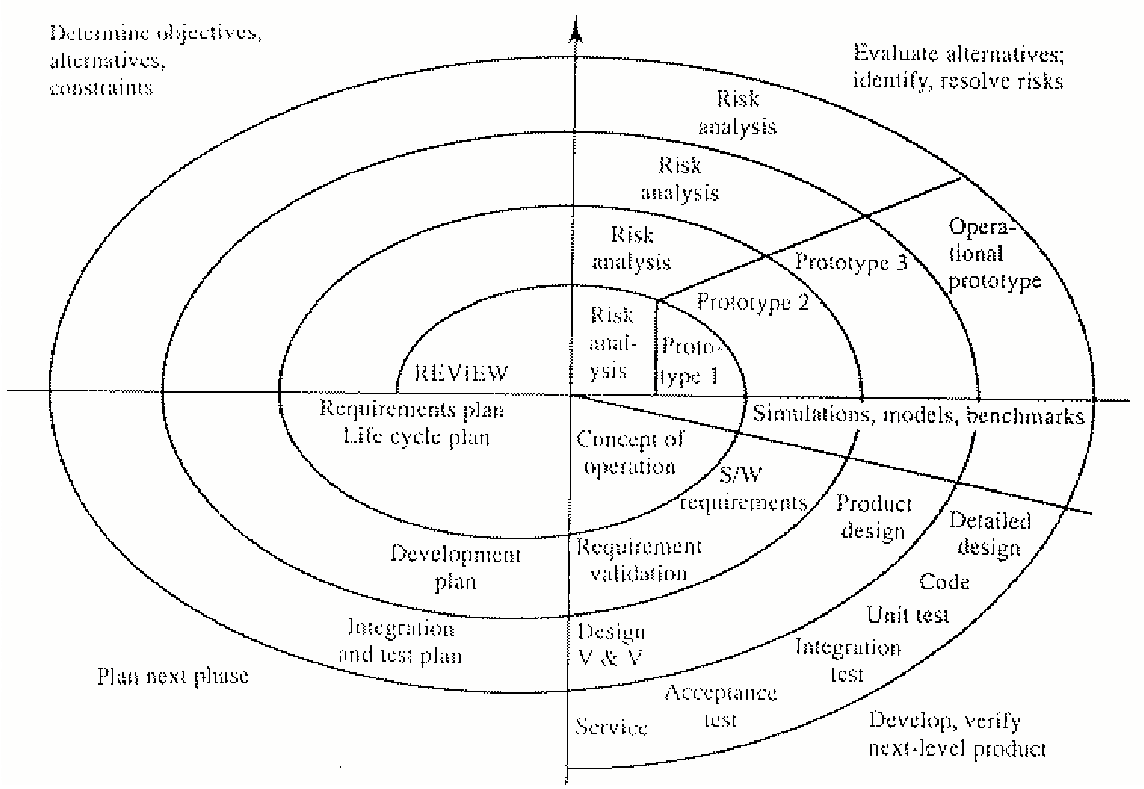


Abbildung 6: Spiralförmige Entwicklung nach Boehm (1988)

Vorteile

- Risiken werden explizit betrachtet (Sommerville 2001, 66).
- Prototypen werden zur Verminderung des Risikos eingesetzt (Pressman 2001, 38).
- Auf neue Erkenntnisse innerhalb des Projekts kann dynamisch reagiert werden.
- Anders als klassische Prozessmodelle, die enden wenn die Software ausgeliefert wird, kann das Spiralmodell auf den gesamten Lebenszyklus der Software angewandt werden (Pressman 2001, 37).
- Andere Vorgehensmodelle werden als Spezialfälle integriert (Balzert 1998, 132; Gnatz 2005, 24).
- Fehler und ungeeignete Alternativen werden frühzeitig eliminiert (Balzert 1998, 132).
- Der Ansatz ist sehr flexibel (Balzert 1998, 133).
- Durch die Betrachtung von Alternativen wird Wiederverwendung unterstützt (Balzert 1998, 133).

Nachteile

- Es ist womöglich schwer, den Kunden davon zu überzeugen dass der evolutionäre Ansatz kontrollierbar ist (Pressman 2001, 38).
- Benötigt Expertenwissen im Bereich Risikobewertung (Pressman 2001, 38). Die „richtige“ Anwendung des Modells hängt entscheidend von der Erfahrung der Projektverantwortlichen ab.

- Es besteht ein hoher Managementaufwand, da oft Entscheidungen über den weiteren Prozessablauf getroffen werden müssen (Balzert 1998, 133).
- Für kleine und mittlere Projekte ist der Ansatz weniger gut geeignet (Balzert 1998, 133).

Enge der Projektvorgaben

Für kleine und mittlere Projekte ist der Ansatz weniger gut geeignet (Balzert 1998, 133).
Der Ansatz bietet Vorteile wenn Time to Market kritisch ist, da er es erlaubt zumindest eine eingeschränkte Version rechtzeitig auszuliefern.
Der Ansatz kann auch eingesetzt werden wenn das Projektteam wenig Erfahrung mit den verwendeten Entwicklungstechniken besitzt, da es im Laufe der Iterationen hinzulernen kann. Im Bereich der Risikobewertung wird jedoch Expertenwissen benötigt. Die korrekte Anwendung des Modells hängt entscheidend von der Erfahrung der Projektverantwortlichen ab.

Umfang

Phasenabdeckung	
Voruntersuchung	abgedeckt
Systemanalyse	abgedeckt
Entwurf	abgedeckt
Erzeugung	abgedeckt
Einführung	abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	abgedeckt
Qualitätssicherung	abgedeckt
Änderungs- und Konfigurationsmanagement	nicht abgedeckt

Abstraktionsgrad

Spiralförmige Entwicklung ist ein universelles Vorgehensmodell.

Erfüllung von Anforderungen

Durch die inkrementelle Entwicklung der Spezifikation ist es den Benutzern und Entwicklern besser möglich die tatsächlichen Anforderungen zu erkennen. Unklare Anforderungen werden durch frühes Feedback in Form von ersten Inkrementen klarer, Lücken können erkannt werden. Die Bedürfnisse der Benutzer können dabei eher als beim Wasserfallmodell erfüllt werden.
Zudem kann der Ansatz auf neu auftauchende oder sich ändernde Anforderungen reagieren. Die Richtung der Entwicklung lässt sich in dem Fall korrigieren. Somit ist das Modell auch bei unstabilen Anforderungen problemlos einsetzbar.

Anpassbarkeit

Das Modell ist sehr flexibel. Es enthält andere Vorgehensmodelle als Spezialfälle. Die Methode kann daher auf bestimmte Projekttypen zugeschnitten werden.
Die Entwicklung eines Systems ist in kleine Einheiten überschaubarer Größe aufgeteilt. Dies ermöglicht es, dass die Entwicklung an einen sich ändernden Kontext des Projektes angepasst werden kann.

Philosophie

Spiralförmige Entwicklung ist ein iteratives Modell. Das primäre Ziel ist es, das Risiko zu minimieren. Das antreibende Moment ist die Betrachtung von Risiken.

Prozesssteuerung

Die Spiralförmige Entwicklung ist aktivitätsorientiert.

Benutzerbeteiligung

Die Benutzer werden während des gesamten Projektes immer wieder am Projekt beteiligt.

3.1.3 Prototypische Vorgehensmodelle

„Ein Prototyp ist die Anfangsversion eines Softwaresystems, die dazu verwendet wird, Konzepte zu demonstrieren, Entwurfsmöglichkeiten auszuprobieren und – grundsätzlich – Erkenntnisse über das Problem und seine möglichen Lösungen zu gewinnen“ (Sommerville 2001, 181).

Prototypen dienen unterschiedlichen Zwecken. Analyseprototypen unterstützen die korrekte und vollständige Ermittlung der Anforderungen. Entwurfsprototypen hingegen sichern, dass die gewählte Systemarchitektur die geforderten nicht-funktionalen Anforderungen erfüllt. Sie werden allgemein dazu eingesetzt, um unklare oder schwierige Entwurfsaspekte zu überprüfen.

Es werden zwei grundlegende Typen von Prototypischen Vorgehensmodellen unterschieden:

1. Evolutionary Prototyping und
2. Throw-Away Prototyping

Bei Evolutionary Prototyping entwickelt man möglichst rasch einen ersten Prototyp. Anhand diesem holt man Feedback von der Kundenseite ein und verfeinert ihn in zahlreichen weiteren Schritten bis eine zufrieden stellende Lösung erreicht ist. Der Endbenutzer wird in die Entwicklung und Auswertung aller Schritte eingebunden. Spezifikation, Entwurf und Implementierung werden überlappt ausgeführt. Eine detaillierte Systemspezifikation wird nicht erstellt.

Das Ziel von Throw-Away Prototyping ist es, die Anforderungen des Kunden besser zu verstehen und somit eine bessere Spezifikation der Anforderungen zu entwickeln. Prototypen werden eingesetzt, um mit den schlecht verstandenen Kundenanforderungen zu experimentieren. Im Unterschied zum Evolutionary Prototyping entsteht aus dem Prototyp kein fertiges System.

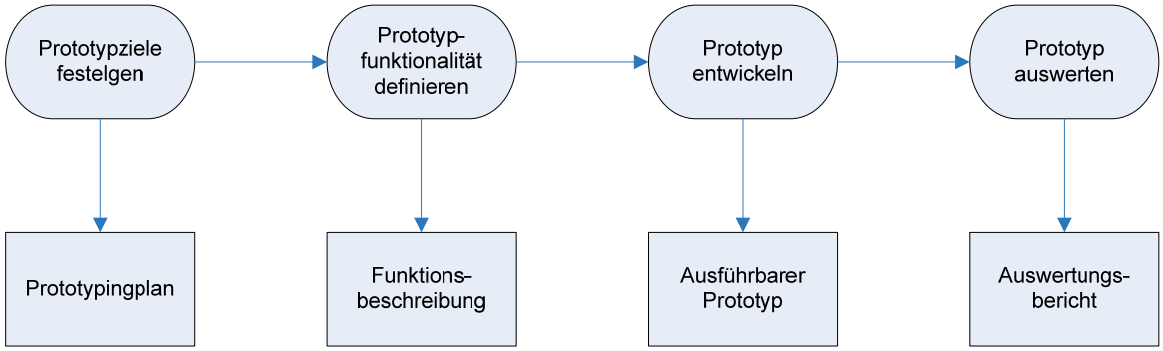


Abbildung 7: Prototypische Vorgehensmodelle (aus Sommerville 2001)

Vorteile

- Das Entwicklungsrisiko wird durch den frühen Einsatz von Prototypen gesenkt (Balzert 1998, 119).
- Prototypen können sinnvoll in andere Vorgehensmodelle integriert werden (Balzert 1998, 119).

- Eine starke Rückkopplung mit dem Endbenutzer und dem Auftraggeber wird gefördert (Balzert 1998, 119).
- Analyseprototypen unterstützen eine korrekte und vollständige Ermittlung der Anforderungen (Bunse und von Knethen 2002, 9).
- Entwurfsprototypen sichern, dass die gewählte Systemarchitektur geforderte nichtfunktionale Anforderungen erfüllt (Bunse und von Knethen 2002, 9).
- Der Ansatz kommt auch mit Anforderungen zurecht, die zu Beginn des Projektes noch unklar sind (Bunse und von Knethen 2002, 9).
- Die schnelle Erstellung eines ersten funktionsfähigen Systems erleichtert die Kommunikation zwischen Entwicklung, Management und Kunde (Sommerville 2001, 182).
- Durch den Einsatz von Prototypen können die Kosten der gesamten Entwicklung gesenkt werden (Sommerville 2001, 182).
- Der Ansatz bringt Systeme hervor, die eine bessere Wartbarkeit aufweisen (Gordon und Bieman 1995).
- Laut einer Studie von Gordon und Bieman (1995) führt der Ansatz zu besseren Entwürfen[1].

Nachteile

- Für Prototypische Vorgehensmodelle ist die übliche Art von Verträgen nicht geeignet (Balzert 1998, 119).
- Prototypen werden oft als Ersatz für die fehlende Dokumentation angesehen (Balzert 1998, 119).
- Prototypen können bei den Kunden falsche Erwartungen wecken (Models for Action Project 1998).
- Falls ineffizienter Programmcode des Prototyps weiterverwendet wird, kann dadurch die Performanz und Stabilität des Systems beeinträchtigt werden (Gordon und Bieman, 1995).
- Die Art und Weise wie der Prototyp benutzt wird, ist nicht unbedingt dieselbe wie die des Endsystems. Daher ist der Tester des Prototyps nicht unbedingt ein typischer Endbenutzer (Sommerville 1989, 412). Beispielsweise werden die Tester die langsamen Funktionen meiden, falls der Prototyp sehr langsam ist, was zu einer verfälschten Art der Benutzung führt.

Enge der Projektvorgaben

Evolutionary Prototyping bietet Vorteile wenn Time to Market kritisch ist, da der Ansatz erlaubt zumindest eine eingeschränkte Version rechtzeitig auszuliefern.

Umfang

Phasenabdeckung	
Voruntersuchung	nicht abgedeckt
Systemanalyse	abgedeckt
Entwurf	abgedeckt
Erzeugung	abgedeckt
Einführung	abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt

[1] Das Models for Action Project (1998) konstatiert jedoch, dass Prototypische Vorgehensmodelle oft schlechte Entwürfe hervorbringen, da keine globale Betrachtung der Integration aller Komponenten stattfindet.

Test	abgedeckt
Projektmanagement	nicht abgedeckt
Qualitätssicherung	nicht abgedeckt
Änderungs- und Konfigurationsmanagement	nicht abgedeckt

Abstraktionsgrad

Evolutionary Prototyping und Throw-Away Prototyping sind universelle Vorgehensmodelle.

Erfüllung von Anforderungen

Durch Prototypen ist es den Benutzern und Entwicklern besser möglich die Anforderungen zu verstehen und die tatsächlichen Anforderungen zu erkennen.
Das Modell bietet Vorteile im Umgang mit unstabilen Anforderungen.

Anpassbarkeit

Bei Evolutionary Prototyping ist die Entwicklung in kleine Einheiten überschaubarer Größe aufgeteilt. Dies ermöglicht es, dass die Entwicklung an einen sich ändernden Kontext des Projektes angepasst werden kann.
Ein Zuschneiden des Modells auf spezielle Projekttypen ist nicht möglich. Es ist jedoch möglich Prototypische Ansätze sinnvoll in andere Vorgehensmodelle zu integrieren.

Philosophie

Evolutionary Prototyping ist ein iteratives Modell. Das primäre Ziel ist es, das Risiko zu minimieren. Das antreibende Moment ist der Code.

Prozesssteuerung

Prototypische Vorgehensmodelle sind produktorientiert.

Benutzerbeteiligung

Das Modell sieht eine enge Zusammenarbeit mit dem Kunden während des gesamten Projektes vor.

3.1.4 Nebenläufige Vorgehensmodelle

Nebenläufige Vorgehensmodelle setzen es sich zum Ziel, möglichst viele Abläufe parallel abzuarbeiten. Dafür werden alle Projektbeteiligten in einem Team integriert und somit ihre gesamten Erfahrungen frühzeitig zusammengebracht. Ziel hierbei ist, alle relevanten Punkte gleich von Beginn an zu berücksichtigen, damit später kein Zusatzaufwand durch Überarbeitungen entsteht.
Man versucht, Zeit einzusparen, indem man Aktivitäten parallel ausführt, Wartezeiten zwischen arbeitsorganisatorisch verbundenen Aktivitäten vermindert und das Motto „right the first time“ dem „trial and error“ vorzieht. Im Falle der reinen Softwareentwicklung führt man die Phasen des Wasserfallmodells jeweils um eine Phase versetzt parallel aus.
Im Gegensatz zu evolutionären Modellen liefert man keine Zwischenversionen sondern gleich das vollständige System aus. (Balzert 1998, 126ff)

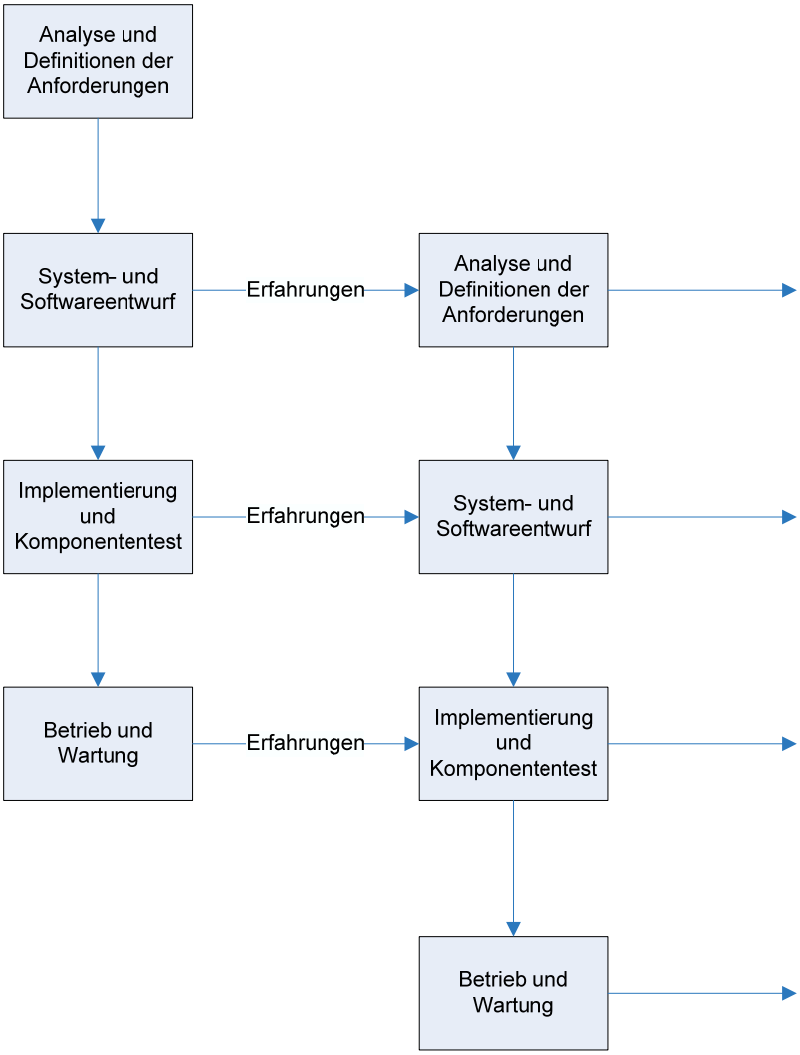


Abbildung 8: Nebenläufige Vorgehensmodelle

Vorteile

- Die Parallelisierung verkürzt Produkteinführungszeiten (Hammer und Champy 2001).
- Der Ansatz fördert die funktionsübergreifende Kollaboration und führt das gesammelte Expertenwissen frühzeitig zusammen (Hildenbrand et al 2006).
- Durch die Integration aller Projektbeteiligten werden Probleme frühzeitig erkannt (Balzer 1998, 239).
- Der Ansatz bietet eine optimale Zeitausnutzung (Balzert 1998, 129).

Nachteile

- Paralleles Ausführen von logisch nachgelagerten Aktivitäten birgt zusätzliche Risiken (Balzert 1998, 127).
- Mit der nebenläufigen Entwicklung ist in jedem Fall ein sehr hoher Aufwand verbunden, um in der Planung Engpässe zu antizipieren (Balzert 1998, 129).
- Paralleles Arbeiten erfordert präzises Management hinsichtlich Aufgaben und Ressourcen, insbesondere bei geteilten Ressourcen auf denen Bearbeitungsrechte und ggf. auch Zeitfenster festgelegt werden müssen. Das Risiko von Konflikten steigt mit dem Grad des Parallelisierens; schlimmstenfalls müssen viele parallel laufende

Prozesse mitsamt ihrer Ergebnisse verworfen werden (z.B. wenn sich eine Kernanforderung an das Produkt ändert).

- Es steht zu bezweifeln, ob man bei Softwareentwicklung das Ziel „right the first time“ erreichen kann (Balzert 1998, 129).
- Es besteht das Risiko, dass grundlegende und kritische Entscheidungen zu spät getroffen werden, so dass Iterationen nötig werden (Balzert 1998, 129)

Enge der Projektvorgaben

Nebenläufige Entwicklung verursacht einen erhöhten Aufwand an Kommunikation und Koordination. Bei mehr Mitarbeitern fällt dabei ein überproportionaler Mehraufwand an. Bei besonders großen und verteilten Projekten ergeben sich daher Probleme bei diesem Modell.

Umfang

Phasenabdeckung	
Voruntersuchung	nicht abgedeckt
Systemanalyse	abgedeckt
Entwurf	abgedeckt
Erzeugung	abgedeckt
Einführung	abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	nicht abgedeckt
Qualitätssicherung	nicht abgedeckt
Änderungs- und Konfigurationsmanagement	nicht abgedeckt

Abstraktionsgrad

Das Nebenläufige Vorgehensmodell ist ein universelles Vorgehensmodell.

Erfüllung von Anforderungen

Sich ändernde Anforderungen sind aufgrund des Prinzips „right the first time“ problematisch. Bei hoher Parallelbearbeitung kann es zu kaskadenartigen Effekten kommen, in denen ganz Teilprojekte still stehen oder scheitern.

Anpassbarkeit

Auf einen sich ändernden Kontext des Projektes kann nur schwer reagiert werden.
Ein Zuschneiden des Modells auf spezielle Projekttypen ist nicht möglich.

Philosophie

Das Nebenläufige Vorgehensmodell ist, wie der Name schon sagt, nebenläufig. Das primäre Ziel ist es, die Entwicklungszeit zu minimieren. Das antreibende Moment ist die Zeit.

Prozesssteuerung

Das Nebenläufige Vorgehensmodell ist aktivitätsorientiert.

Benutzerbeteiligung

Das Modell sieht eine enge Zusammenarbeit mit dem Kunden während des gesamten Projektes vor.

3.1.5 Komponentenbasierte Entwicklung

Die zentrale Idee der Komponentenbasierten Entwicklung ist Wiederverwendung. Man konzentriert sich darauf, bereits entwickelte und für das System wieder verwendbare Teile zu integrieren, anstatt sie von Grund auf neu zu entwickeln.
Anforderungsspezifikation und Validierung laufen ähnlich wie bei anderen Prozessen ab. Die Unterschiede liegen in den anderen Phasen:

1. **Analyse der Komponenten**: Ausgehend von der Anforderungsspezifikation werden bestehende Komponenten gesucht, die diese umsetzen. Oft findet man dabei keine exakte Umsetzung, sondern nur Komponenten die einen Teil der geforderten Funktionalität realisieren.
2. **Anpassung der Anforderungen**: In dieser Phase werden die Anforderungen angepasst, so dass sie den vorhandenen Komponenten entsprechen. Wenn eine Anpassung unmöglich ist, führt man die Komponentenanalyse erneut aus, um alternative Lösungen zu finden.
3. **Systementwurf mit Wiederverwendung**: In dieser Phase wird der Systementwurf erstellt. Es wird dabei der Wiederverwendung von Komponenten Rechnung getragen.
4. **Entwicklung und Integration**: Die Funktionalität, die nicht durch bestehende Komponenten bereitgestellt wird, wird entwickelt. Die Integration wird vorgenommen.

Obwohl das Prinzip der Wiederverwendung eine weite Verbreitung gefunden hat und in vielen Prozessen vorkommt, orientieren nur wenige Organisationen explizit daran (Sommerville 2001, 57).

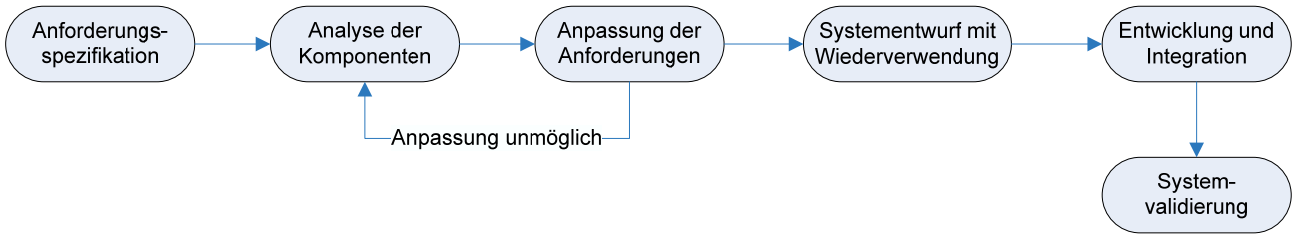


Abbildung 9: Komponentenbasierte Entwicklung (aus Sommerville 2001)

Vorteile

- Ein offensichtlicher Vorteil dieses Ansatzes liegt darin, dass weniger Software entwickelt werden muss, wodurch Zeit und Kosten gespart werden (Sommerville 2001, 63).
- Eine Organisation kann sich auf die eigenen Stärken konzentrieren und Komponenten, die Standardfunktionalität bereitstellen und/oder nicht der Kernkompetenz entsprechen, zukaufen (Balzert 1998, 126).
- Komponentenbasierte Entwicklung lässt sich auch mit anderen Vorgehensmodellen kombinieren.

Nachteile

- Der Hauptnachteil liegt darin, dass Kompromisse bei den Anforderungen gemacht werden müssen, was dazu führen kann, dass die Bedürfnisse des Kunden nicht abgedeckt werden (Sommerville 2001, 63).
- Die Weiterentwicklung wird dadurch erschwert, dass man keine Kontrolle über neue Versionen der wiederverwendeten Komponenten hat (Sommerville 2001, 63).

- Der Ansatz ist nur voll nutzbar, wenn objekt-/komponentenorientierte Methoden eingesetzt werden (Balzert 1998, 126).
- Eine geeignete Infrastruktur (Wiederverwendungsarchiv) muss vorhanden sein (Balzert 1998, 126).
- Bestimmte technische Probleme, wie z.B. Inkompatibilitäten von Klassenbibliotheken, müssen überwunden werden (Balzert 1998, 126).

Enge der Projektvorgaben

Das Modell ist nur voll nutzbar, wenn objekt- bzw. komponentenorientierte Methoden eingesetzt werden.
Der Ansatz ist besonders für zeit- und kostenkritische Projekte geeignet.

Umfang

Phasenabdeckung	
Voruntersuchung	nicht abgedeckt
Systemanalyse	abgedeckt
Entwurf	abgedeckt
Erzeugung	abgedeckt
Einführung	abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	nicht abgedeckt
Qualitätssicherung	nicht abgedeckt
Änderungs- und Konfigurationsmanagement	nicht abgedeckt

Abstraktionsgrad

Komponentenbasierte Entwicklung ist ein universelles Vorgehensmodell.

Erfüllung von Anforderungen

Der Ansatz erfordert, dass Kompromisse bei den Anforderungen gemacht werden. Dies kann dazu führen, dass die tatsächlichen Anforderungen nicht erfüllt werden.
Aufgrund der sequenziellen Ausführung können sich ändernde Anforderungen nur schwer berücksichtigt werden. Die Anforderungen werden zu Beginn des Projektes vollständig spezifiziert. Falls zu Beginn die tatsächlichen Anforderungen nicht bekannt oder noch unklar sind können sie nicht vom Produkt erfüllt werden. Der Grad der Erfüllung/Erreichung der Projektziele ist weiterhin von der Verfügbarkeit benötigter Komponenten (zum Zeitpunkt der Erstellung/Betrieb/Pflege/Weiterentwicklung) abhängig.

Anpassbarkeit

Auf einen sich ändernden Kontext des Projektes kann nur schwer reagiert werden.
Ein Zuschneiden des Modells auf spezielle Projekttypen ist nicht möglich. Es ist jedoch möglich Komponentenbasierte Entwicklung sinnvoll in andere Vorgehensmodelle zu integrieren.

Philosophie

Komponentenbasierte Entwicklung ist ein sequenzielles Modell. Das primäre Ziel ist es, die Entwicklungszeit und Kosten zu minimieren. Das antreibende Moment sind wieder verwendbare Komponenten.

Prozesssteuerung

Komponentenbasierte Entwicklung ist aktivitätsorientiert.

Benutzerbeteiligung

Benutzer werden nur zu Beginn des Projektes und bei Auslieferung des Produktes beteiligt. Daher können sie keinen Einfluss auf das laufende Projekt ausüben. Fehlentwicklungen werden daher unter Umständen erst während des Einsatzes entdeckt.

3.1.6 Formale Systementwicklung

Bei diesem Ansatz wird die Spezifikation der Anforderungen so weit verfeinert bis eine detaillierte, formale Spezifikation vorliegt, die z.B. in einer mathematischen Notation repräsentiert wird. Die formale Spezifikation wird dann in einer Reihe von Umwandlungen in ein äquivalentes Programm umgesetzt. Durch Verifikationen wird gewährleistet, dass diese Umwandlungen die Korrektheit erhalten und somit das Programm seiner Spezifikation entspricht.
Diese Gewährleistung macht den Ansatz für sicherheitskritische Systeme interessant. Außerhalb dieser Systemdomäne wird er jedoch kaum verwendet (Sommerville 1989, 66).

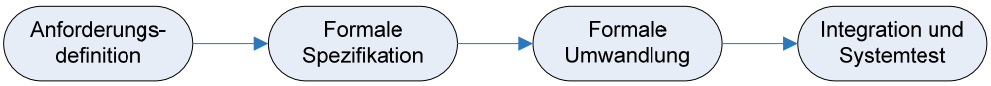


Abbildung 10: Formale Systementwicklung (aus Sommerville 2001)

Vorteile

- Formale Systementwicklung ist besonders gut für die Entwicklung von Systemen geeignet, bei denen Sicherheit und Verlässlichkeit kritische Faktoren sind (Sommerville 1989, 66).
- Die Methode gewährleistet, dass das Programm der Spezifikation entspricht.
- Verträge können die Spezifikation des Systems als Grundlage verwenden.

Nachteile

- Der Zeit- und Kostenaufwand ist sehr hoch (Pressman 2001, 44).
- Umfangreiches Training ist nötig, da nur wenige Software-Entwickler über das entsprechende Wissen verfügen (Pressman 2001, 44).
- Die Kommunikation mit dem Kunden gestaltet sich schwierig, da dieser mit formalen Modellen in der Regel nicht viel anfangen kann (Pressman 2001, 44).

Enge der Projektvorgaben

Formale Systementwicklung ist besonders gut für die Entwicklung von Systemen geeignet, bei denen Sicherheit und Verlässlichkeit kritische Faktoren sind.
Für Projekte, die enge Vorgaben hinsichtlich Zeit und Kosten haben, ist der Ansatz eher ungeeignet.
Das Modell ist eher für kleine und mittlere Projekte geeignet, da eine formale Spezifikation großer Systeme zu komplex wird.
Formale Systementwicklung setzt sehr spezifisches Expertenwissen voraus.

Umfang

Phasenabdeckung	
Voruntersuchung	nicht abgedeckt
Systemanalyse	abgedeckt
Entwurf	abgedeckt
Erzeugung	abgedeckt
Einführung	nicht abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	nicht abgedeckt
Qualitätssicherung	abgedeckt
Änderungs- und Konfigurationsmanagement	nicht abgedeckt

Abstraktionsgrad

Formale Systementwicklung ist ein universelles Vorgehensmodell.

Erfüllung von Anforderungen

Aufgrund der sequenziellen Ausführung können sich ändernde Anforderungen nur schwer berücksichtigt werden. Die Anforderungen werden zu Beginn des Projektes vollständig spezifiziert. Falls zu Beginn die tatsächlichen Anforderungen nicht bekannt oder noch unklar sind können sie nicht vom Produkt erfüllt werden.
Formale Systementwicklung ist nur bei Vorliegen stabiler Anforderungen problemlos einsetzbar.

Anpassbarkeit

Auf einen sich ändernden Kontext des Projektes kann nur schwer reagiert werden.
Ein Zuschneiden des Modells auf spezielle Projekttypen ist nicht möglich.

Philosophie

Formale Systementwicklung ist ein sequenzielles Modell. Das primäre Ziel ist es, die Korrektheit zu gewährleisten. Das antreibende Moment sind formale Transformationen.

Prozesssteuerung

Formale Systementwicklung ist ergebnisorientiert.

Benutzerbeteiligung

Benutzer werden nur zu Beginn des Projektes und bei Auslieferung des Produktes beteiligt. Daher können sie keinen Einfluss auf das laufende Projekt ausüben. Fehlentwicklungen werden daher unter Umständen erst während des Einsatzes entdeckt.
Die Kommunikation mit dem Kunden gestaltet sich schwierig, da dieser mit formalen Modellen in der Regel nicht viel anfangen kann.

3.2 Weltliche Vorgehensmodelle

3.2.1 Extreme Programming

Extreme Programming (XP) ist eine agile Softwareentwicklungsmethode, die iterativ vorgeht. Die Methode versucht organisatorischen Ballast soweit wie möglich zu vermeiden. Der Fokus liegt auf der Erstellung des Programmcodes.
Das Ziel von XP ist die effiziente Entwicklung qualitativ hochwertiger Software unter Einhaltung des Zeit- und Kostenbudgets. Um dieses Ziel zu erreichen hält XP eine Reihe von Werten, Prinzipien und Entwicklungspraktiken bereit. Die vier zentralen Werte sind:

- Einfachheit
- Kommunikation
- Feedback
- Eigenverantwortung und Courage

XP nutzt bewährte Entwicklungspraktiken:

1. Planspiel
2. Metaphern
3. Pair Programming

4. Testen
5. Refactoring
6. Gemeinsame Verantwortung für den Code
7. Kleine Releases
8. Kontinuierliche Integration
9. 40 Stunden Woche
10. Ständig verfügbarer Kunde
11. Programmierstandards
12. Einfaches Design

Mit den zwölf Entwicklungspraktiken setzt XP die folgenden „agilen Prinzipien“ um:

1. Inkrementelle Entwicklung wird durch kleine, häufige Releases unterstützt. Die Anforderungsspezifikation, die auf Kundenszenarios (so genannten User Stories) basiert, kann als Grundlage für die Planung der Iterationen dienen.
2. Der Kunde ist ständig vor Ort und nimmt an der Entwicklung teil. Außerdem ist er dafür verantwortlich, Funktionstests zu definieren.
3. Pair Programming und die gemeinsame Verantwortung für den Code legen den Fokus auf Personen und nicht auf Prozesse.
4. Änderungen werden durch regelmäßige Releases, den Test-First Ansatz und kontinuierliche Integration unterstützt.
5. Einfachheit wird durch ständiges Refactoring erreicht, um die Qualität des Codes zu verbessern und den Einsatz von einfachen Designs zu ermöglichen.

Bei XP nimmt sich der Kunde der Spezifikation und der Priorisierung von Anforderungen an. Er wird dabei durch das Entwicklungsteam unterstützt. Niedergeschrieben werden die Anforderungen in so genannten Story Cards. Der Kunde gibt vor, welche Stories im nächsten Release verwirklicht werden sollen. Diese werden vom Entwicklungsteam in einzelne Tasks aufgebrochen und realisiert. Noch nicht implementierte Stories können problemlos geändert werden. Beziehen sich Änderungswünsche auf Teile des Systems die schon ausgeliefert wurden, werden einfach neue Stories entwickelt, die diese Änderungen beschreiben.

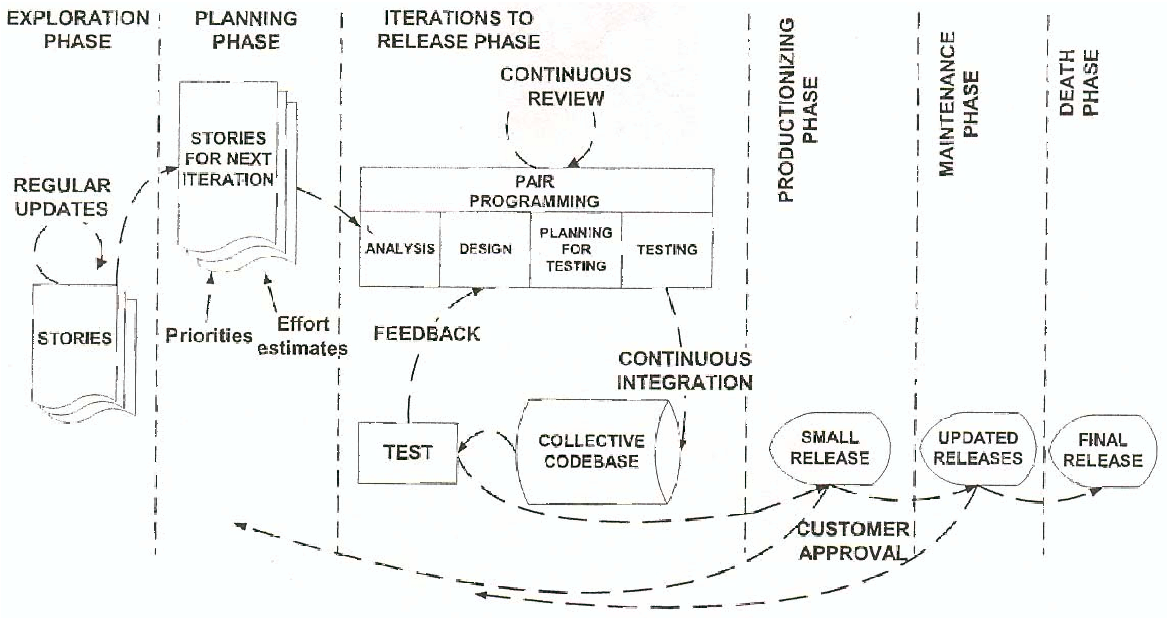


Abbildung 11: eXtreme Programming

Vorteile

- Das Team wird motiviert (Rumpe 2003, 29).
- Der Aufwand wird reduziert (Rumpe 2003, 30).
- XP ermöglicht die schnelle Reaktion auf sich ändernde Anforderungen oder Änderungen im Projektumfeld (Rumpe 2003, 30f).
- Qualität und Effizienz werden durch automatisierte Tests, Pair Programming und die Forderung nach Einfachheit gefördert (Rumpe 2003, 33).
- Die Betonung von Teamwork und Kommunikation erhöht die Effizienz eines Software Teams (Humphrey 2001).
- Aufwandsabschätzungen helfen bei der Einhaltung des Zeitplans und bei der Verpflichtung der Entwickler auf den Plan (Humphrey 2001).
- XP fördert die kontinuierliche Messung des Fortschrittes (Humphrey 2001).
- Häufige Reviews erhöhen die Effizienz (Humphrey 2001).
- XP minimiert das Risiko des völligen Scheiterns des Projekts (Spinellis 2001).
- Risiken werden abgeschätzt und überwacht.
- XP hält eine Reihe von Praktiken zur Förderung der Qualität der Implementierung bereit.
- Weiterentwicklung ist zentraler Bestandteil von XP.

Nachteile

- Oft sind Kunden nicht in der Lage oder nicht willens, sich in dem von XP geforderten Maße in das Projekt einzubringen.
- Von den Teammitgliedern wird unbedingte Teamfähigkeit und Kommunikationsfähigkeit verlangt.
- Für XP ist die übliche Art von Verträgen nicht geeignet.
- Da XP vielfach Änderungen am System verlangt, ist es auf eine abgeflachte Kostenkurve angewiesen. Sollte aus technologischen Gründen diese Kurve aber exponentiell sein, Änderungen also sehr teuer werden, wenn sie spät erfolgen, dann ist XP nicht einsetzbar (Beck 2003, 157).
- XP ist auf eine Umgebung angewiesen, in der Feedback schnell erfolgen kann (Beck 2003, 157).
- XP ist nur für Projekte mit höchstens zwölf Mitarbeitern gedacht. Falls jedoch eine größere Mitarbeiterzahl unumgänglich ist, sollte man laut Beck (2004, 112) das Problem in viele kleine Teile zerlegen, die von autonomen Teams bearbeitet werden können.
- Der Fokus auf den Code anstatt auf das Design kann bei großen Systemen zum Problem werden (Humphrey 2001).
- Die mangelnde Design-Dokumentation macht Wiederverwendung schwer (Humphrey 2001).
- Es wird nicht immer lesbarer Code produziert. XP ist aber abhängig davon, dass der Code lesbar ist (Humphrey 2001).
- XP benötigt ein angepasstes Management (Humphrey 2001).
- XP hält keine Anleitung für die Einführung der Methode bereit (Humphrey 2001).
- Pair Programming ist weniger strukturiert als Peer Reviews und daher weniger effektiv (Paulk 2001).
- Die Methode ist nur im objektorientierten Umfeld voll nutzbar (Spinellis 2001).
- Es wird kaum Dokumentation erstellt.
- Es besteht die „Gefahr des ‚ewigen Refactorings', wenn die Wünsche des Kunden keine klare Richtung finden" (Dogs und Klimmer 2005, 194).

Enge der Projektvorgaben

XP ist gut für Projekte geeignet, bei denen Time to Market kritisch ist und die Anforderungen sich häufig ändern, zu Beginn unbekannt oder unklar bzw. instabil sind.
XP benötigt ein Projektumfeld, in dem Feedback schnell erfolgen kann.
Das Projektteam muss klein sein. XP sieht Teams von höchstens zwölf Mitarbeitern vor.
Voll nutzbar ist der Ansatz nur im objektorientierten Umfeld.
XP funktioniert nur bei Projekten die eine abgeflachte Kostenkurve aufweisen. Ist aus technologischen Gründen diese Kurve exponentiell, so ist XP nicht einsetzbar.

Umfang

Phasenabdeckung	
Voruntersuchung	nicht abgedeckt
Systemanalyse	abgedeckt
Entwurf	abgedeckt
Erzeugung	abgedeckt
Einführung	abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	abgedeckt
Qualitätssicherung	abgedeckt
Änderungs- und Konfigurationsmanagement	abgedeckt

Abstraktionsgrad

XP ist ein weltliches Vorgehensmodell.

Erfüllung von Anforderungen

Der Ansatz ist sehr gut geeignet um die tatsächlichen Anforderungen herauszufinden (zum Beispiel im Rahmen einer Prototyperstellung). Er ist für Projekte mit instabilen Anforderungen gedacht.

Anpassbarkeit

XP ermöglicht die schnelle Reaktion auf einen sich ändernden Kontext des Projektes.
Ein Zuschneiden des Modells auf spezielle Projekttypen ist nicht möglich.

Philosophie

XP ist ein iteratives Modell. Das primäre Ziel ist es, den Kunden zufrieden zu stellen. Das antreibende Moment ist der Code.

Prozesssteuerung

XP ist aktivitätsorientiert.

Benutzerbeteiligung

Das Modell sieht eine intensive Zusammenarbeit mit dem Kunden während des gesamten Projektes vor.

3.2.2 Rational Unified Process

Im Wesentlichen entspricht die Prozessarchitektur dem Spiralmodell. Der Rational Unified Process (RUP) vereint Iterationen und gute Spezifikations- und Entwurfspraktiken mit Elementen des Wasserfallmodells, der evolutionären Entwicklung und der komponentenbasierten Entwicklung. Er deckt alle wichtigen Phasen des Softwarelebenszyklus ab. Er hält eine Reihe von Best Practices und Road Maps für bestimmte Arten von Entwicklungsprojekten (komponentenbasierte Lösungen, E-Business-Lösungen, benutzerfreundliche Lö-

sungen, kleine Softwareprojekte und agile Entwicklungsmethoden) bereit. Zudem weist er ein Rollenmodell mit 30 definierten Rollen, die in sechs Kategorien unterteilt sind, auf. RUP bietet zusätzlich eine durchgängige Werkzeugunterstützung. Es finden sich darunter auch Werkzeuge zur Modelladaption, d.h. zur Prozessmodellierung, zur Verwaltung und Erstellung von Inhalten und zum Tailoring des RUP[2].
RUP sieht vier Phasen vor, die im Gegensatz zum Wasserfallmodell jedoch nicht Aktivitäten in der Entwicklung widerspiegeln. Sie sind eher mit geschäftlichen als mit technischen Aspekten verknüpft.

1. **Inception**: In dieser Phase wird ein Geschäftsmodell aufgestellt. Sämtliche Entitäten die mit dem System interagieren, seien dies Personen oder auch andere Systeme, werden identifiziert. Die Interaktionen werden ebenfalls definiert. Die Rentabilität des Projektes wird abgeschätzt.
2. **Elaboration**: In dieser Phase soll ein Verständnis der Problemdomäne gewonnen, die Architektur des Systems entworfen, der Projektplan aufgestellt und Projektrisiken identifiziert werden.
3. **Construction**: Diese Phase behandelt Systementwurf, Implementierung und Tests. Die Teile des Systems werden parallel entwickelt und dann zusammengeführt.
4. **Transition**: Ziel dieser Phase ist es, das System auszuliefern und es in seiner Zielumgebung zum Einsatz zu bringen.

Iterationen finden auf zwei Ebenen statt. Zum einen kann jede Phase iterativ ausgeführt werden und zum anderen können die vier Phasen als Ganzes Inkremente aufweisen. Konkrete Aufgaben im Kontext eines Projekts werden vom RUP als Workflows modelliert. Folgende Workflows sind bereits standardmäßig definiert:

- Unternehmensmodellierung
- Anforderungen
- Analyse und Design
- Implementierung
- Test
- Installation und Einsatz
- Konfigurationsmanagement
- Projektmanagement
- Umgebung

Definiert werden die einzelnen Workflows durch die Elemente Aktivität, Artefakt und Rolle. Alle Workflows können gleichzeitig ausgeführt werden. Bestimmte Workflows werden in bestimmten Phasen jedoch verstärkt durchgeführt, während andere kaum Beachtung finden.
Ähnlich wie XP definiert auch der RUP einige fundamentale Prinzipien, die der Philosophie des Prozesses zugrunde liegen:

1. Software wird iterativ entwickelt.
2. Anforderungen sind zu managen.
3. Komponentenbasierte Architekturen werden verwendet.
4. Software wird grafisch modelliert.
5. Die Qualität der Software wird geprüft.
6. Änderungen an der Software werden kontrolliert.

Der Rational Unified Process fokussiert die Objektorientierung als Entwicklungsparadigma und bedient sich hierzu in weiten Teilen der UML als Modellierungssprache.

[2] Der RUP ist in Teilen als Open Unified Process (OUP), ehemals Basic RUP (BUP) im Rahmen des Eclipse Process Frameworks (EPF) frei verfügbar. Die Prozessbeschreibung ist umfangsmäßig reduziert, jedoch beliebig anpassbar. Werkzeugunterstützt wird EPF durch die direkte Integration in Eclipse als Tool für das Authoring des Prozesses. EPF stellt unter anderem eine Basisversion, zumindest technisch, für die Process Workbench von IBM/Rational dar.

Vorteile

- Der Prozess ist durch seine Anpassungsmöglichkeiten flexibel (Hildenbrand et al 2006).
- Für den RUP stehen eine Reihe von Werkzeugen zur Verfügung (Hildenbrand et al 2006).
- Risiken werden explizit betrachtet.
- Zu erstellende Dokumentation wird vom Vorgehensmodell festgelegt.
- Andere Vorgehensmodelle werden als Spezialfälle integriert.
- Der RUP deckt den gesamten Lebenszyklus der Software ab.

Nachteile

- Durch die Anpassung entsteht Zusatzaufwand.
- Der RUP ist durch die starke Verwendung der UML als Modellierungssprache auf deren Einsetzbarkeit in den Zieldomänen beschränkt. Domänen ohne passende UML-Profile sind nur mit Zusatzaufwand bedienbar.
- RUP ist ein außerordentlich komplexer Prozess, der ohne Werkzeugunterstützung nicht umsetzbar und anpassbar ist. Er setzt in weiten Bereichen Expertenwissen voraus.
- Der RUP ist für gering budgetierte Projekte, bzw. für Projekte mit kurzer Laufzeit nur beschränkt geeignet.
- Der RUP stellt hohe Anforderungen an eine Werkzeug-/Projektlandschaft.

Abhängig vom Grad der Anpassung weist der RUP weitere Vor- und Nachteile auf, welche daher an dieser Stelle nicht allgemein formuliert werden können.

Enge der Projektvorgaben

Der Rational Unified Process lässt sich an die jeweiligen Projektvorgaben anpassen.

Umfang

Phasenabdeckung	
Voruntersuchung	abgedeckt
Systemanalyse	abgedeckt
Entwurf	abgedeckt
Erzeugung	abgedeckt
Einführung	abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	abgedeckt
Qualitätssicherung	abgedeckt
Änderungs- und Konfigurationsmanagement	abgedeckt

Abstraktionsgrad

Der Rational Unified Process ist ein weltliches Vorgehensmodell.

Erfüllung von Anforderungen

Der Rational Unified Process hält Vorgehensweisen bereit, die es erlauben mit instabilen Anforderungen umzugehen sowie die tatsächlichen Anforderungen zu entdecken und zu erfüllen. Eine entsprechende Anpassung ist jedoch notwendig.

Anpassbarkeit

Der Prozess ist durch seine Tailoringmöglichkeiten sehr flexibel. Er kann je nach Projekttyp zurechtgeschnitten werden. Andere Vorgehensmodelle werden als Spezialfälle integriert.

Philosophie

Der Rational Unified Process ist ein iteratives Modell. Das antreibende Moment sind Anwendungsfälle.

Prozesssteuerung

Der Rational Unified Process ist aktivitätsorientiert.

Benutzerbeteiligung

Je nach Road Map kann der Benutzer nur zu Beginn und am Ende des Projektes oder in regelmäßigen Abständen im Verlauf des Projektes beteiligt werden.

3.2.3 V-Modell XT

Das V-Modell XT ist eine Weiterentwicklung des V-Modells 97. Es ist ein generisches Vorgehensmodell und erhebt den Anspruch auf Anwendbarkeit in allen Arten von Entwicklungsprojekten im IT-Umfeld. Das V-Modell XT betrachtet neben der Auftragnehmerseite (AN) auch die Auftraggeberseite (AG) eines Projektes. Es deckt alle relevanten Aufgaben eines Entwicklungsprojektes ab. Dabei liegt der Fokus auf den zu erstellenden Produkten.
Eine wesentliche Errungenschaft des V-Modell XT sind seine Tailoring-Möglichkeiten. Das Modell bietet eine Reihe von Vorgehensbausteinen, mit denen sich spezifische Prozessmodelle entwerfen lassen. Dabei geht man wie folgt vor:

1. Projekttyp auswählen: Man wählt einen der vom V-Modell XT unterstützten Projekttypen. Möglichkeiten sind:
    a. Systementwicklungsprojekt (AG)
    b. Systementwicklungsprojekt (AN)
    c. Systementwicklungsprojekt (AG/AN)
    d. Einführung und Pflege eines organisationsspezifischen Vorgehensmodells
2. Vorgehensbausteine auswählen: Abhängig vom Projekttyp müssen bestimmte Vorgehensbausteine verwendet werden. Andere Vorgehensbausteine sind optional.
3. Projektdurchführungsstrategie auswählen: Für jeden Projekttyp bietet das V-Modell XT eine Reihe (mindestens jedoch eine) von Projektdurchführungsstrategien an. Möglichkeiten sind:
    a. Vergabe und Durchführung eines Systementwicklungsprojektes (AG)
    b. Vergabe und Durchführung mehrerer Systementwicklungsprojekte (AG)
    c. Inkrementelle Systementwicklung (AN oder AG/AN)
    d. Komponentenbasierte Systementwicklung (AN oder AG/AN)
    e. Agile Systementwicklung (AN oder AG/AN)
    f. Wartung und Pflege von Systemen (AN oder AG/AN)

Das V-Modell XT hält Open Source Werkzeuge zur Anpassung des Modells in verschiedenen Etappen des Lebenszyklus bereit.

Vorteile

- Im Gegensatz zu anderen Vorgehensmodellen deckt das V-Modell XT sowohl die Auftragnehmer- als auch die Auftraggeberseite ab.
- Das V-Modell XT deckt in großem Maße die Projektmanagementaktivitäten ab (Hildenbrand et al 2006).
- Das V-Modell XT ist durch seine Tailoring-Möglichkeiten sehr flexibel.
- Produkte, Rollen und Verantwortlichkeiten sind sehr detailliert beschrieben.

- Das V-Modell XT wird durch Werkzeuge unterstützt (Hildenbrand et al 2006).
- Andere Vorgehensmodelle können als Spezialfälle integriert werden.
- Risiken werden explizit betrachtet.
- Zu erstellende Dokumentation wird vom Vorgehensmodell festgelegt.
- Die Ausgestaltung von Verträgen zwischen Auftraggeber und Auftragnehmer wird von dem Vorgehensmodell behandelt.
- Die ausführliche Beschreibung der erwarteten Ergebnistypen macht es unwahrscheinlich, dass Teile des Projektes bei der Planung außer Acht gelassen werden.
- Das V-Modell XT bietet große Freiräume in der Ausgestaltung, zum Beispiel mit Methoden und weiteren angepassten Produkte und Vorgehensweisen.

Nachteile

- Die konkreten Tätigkeiten zur Erreichung der Ergebnisse werden nicht beschrieben.
- Die Methodenneutralität des V-Modells erfordert von Projektleitern und insbesondere von Prozessingenieuren ein hohes Maß an Abstraktionsfähigkeit.
- Der Schulungsaufwand für die Einführung des V-Modells ist hoch.

Abhängig vom Tailoring weist das V-Modell XT unterschiedliche Vor- und Nachteile auf, welche daher an dieser Stelle nicht allgemein formuliert werden können.

Enge der Projektvorgaben

Das V-Modell XT lässt sich an die jeweiligen Projektvorgaben anpassen.

Umfang

Phasenabdeckung	
Voruntersuchung	abgedeckt
Systemanalyse	abgedeckt
Entwurf	abgedeckt
Erzeugung	abgedeckt
Einführung	abgedeckt

Prozessabdeckung	
Entwicklung	abgedeckt
Test	abgedeckt
Projektmanagement	abgedeckt
Qualitätssicherung	abgedeckt
Änderungs- und Konfigurationsmanagement	abgedeckt

Abstraktionsgrad

Das V-Modell XT ist ein weltliches Vorgehensmodell.

Erfüllung von Anforderungen

Das V-Modell XT hält Vorgehensweisen bereit, die es unter anderem erlauben mit instabilen Anforderungen umzugehen sowie die tatsächlichen Anforderungen zu entdecken und zu erfüllen. Ein Tailoring ist jedem Fall notwendig.

Anpassbarkeit

Der Prozess ist durch seine Tailoring-Möglichkeiten und die offene Architektur sehr flexibel. Er kann je nach Projekttyp zurechtgeschnitten werden. Andere Vorgehensmodelle können als Spezialfälle integriert werden.

Philosophie

Das V-Modell XT kann sequenziell oder iterativ durchgeführt werden. Das primäre Ziel ist es, die Qualität zu maximieren. Das antreibende Moment sind Dokumente.

Prozesssteuerung

Das V-Modell XT ist ergebnisorientiert.

Benutzerbeteiligung

Ziel des V-Modells ist es, Anwender in regelmäßigen Abständen im Verlauf des Projektes zu beteiligt.

3.3 Zusammenfassung

Wir haben jedes der betrachteten Vorgehensmodelle mittels unseres Kriterienkatalogs analysiert. Die Ergebnisse daraus wollen wir in diesem Kapitel übersichtlich darstellen. Zudem sollen die verschiedenen Vorgehensmodelle einander gegenüber gestellt werden.

In den folgenden Tabellen sind die Ergebnisse hinsichtlich der Enge der Projektvorgaben dargestellt. Zu jedem Vorgehensmodell wird angeführt ob es bzgl. der untersuchten Kriterien vorteilhaft (+), neutral (0) oder nachteilig (-) ist. Unter „benötigt“ sind besondere Bedingungen aufgelistet, die das jeweilige Vorgehensmodell erfordert.

	Wasserfall-modell	Evolutionäre Entwicklung	Explorative Entwicklung	Inkrementelle Entwicklung	Spiralförmige Entwicklung
Zeit-/kostenkritisch	-	0	0	0	0
Groß/verteilt	0	-	-	0	0
Klein/mittel	0	0	0	0	-
Time to Market	0	+	+	+	0
Sicherheits-kritisch	0	-	-	-	0
Benötigt			High- Level Sprachen		Expertenwissen bzgl. Risikobewertung

Tabelle 1: Enge der Projektvorgaben 1

	Prototypische Vorgehensmodelle	Nebenläufige Vorgehensmodelle	Komponentenbasierte Entwicklung	Formale Systementwicklung
Zeit-/kostenkritisch	0	0	+	-
Groß/verteilt	0	-	0	-
Klein/mittel	0	0	0	0
Time to Market	+	0	0	0
Sicherheits-kritisch	0	0	0	+
Benötigt			Objektorientierung	Expertenwissen

Tabelle 2: Enge der Projektvorgaben 2

	XP	RUP	V-Modell XT
Zeit-	0	variabel	variabel

	XP	RUP	V-Modell XT
/kostenkritisch			
Groß/verteilt	-	variabel	variabel
Klein/mittel	+	variabel	variabel
Time to Market	+	variabel	variabel
Sicherheits-kritisch	0	variabel	variabel
Benötigt	schnelles Feedback, abgeflachte Kostenkurve, Objektorientierung		Einführung und Anpassung des Prozesses zur effektiven Anwendung

Tabelle 3: Enge der Projektvorgaben 3

In der folgenden Tabelle ist die Phasenabdeckung der untersuchten Vorgehensmodelle dargestellt. „X“ kennzeichnet Phasen, die vom jeweiligen Vorgehensmodell abgedeckt werden.

	Voruntersuchung	Systemanalyse	Entwurf	Erzeugung	Einführung
Wasserfallmodell		X	X	X	X
Evolutionäre Entwicklung		X	X	X	X
Explorative Entwicklung		X	X	X	
Inkrementelle Entwicklung		X	X	X	X
Spiralförmige Entwicklung	X	X	X	X	X
Prototypische Vorgehensmodelle		X	X	X	X
Nebenläufige Vorgehensmodelle		X	X	X	X
Komponentenbasierte Entwicklung		X	X	X	X
Formale Systementwicklung		X	X	X	
XP		X	X	X	X
RUP	X	X	X	X	X
V-Modell XT	X	X	X	X	X

Tabelle 4: Phasenabdeckung

In der folgenden Tabelle ist die Prozessabdeckung der untersuchten Vorgehensmodelle dargestellt. „X“ kennzeichnet Teilprozesse, die vom jeweiligen Vorgehensmodell abgedeckt werden.

	Entwicklung	Test	Projektmanagement	Qualitätssicherung	Änderungs- und Konfigurationsmanagment
Wasserfallmodell	X	X			
Evolutionäre Entwicklung	X	X			

	Entwicklung	Test	Projekt-management	Qualitäts-sicherung	Änderungs- und Konfigurationsmanagment
Explorative Entwicklung	X	X			
Inkrementelle Entwicklung	X	X			
Spiralförmige Entwicklung	X	X	X	X	
Prototypische Vorgehensmodelle	X	X			
Nebenläufige Vorgehensmodelle	X	X			
Komponentenbasierte Entwicklung	X	X			
Formale Systementwicklung	X	X		X	
XP	X	X	X	X	X
RUP	X	X	X	X	X
V-Modell XT	X	X	X	X	X

Tabelle 5: Prozessabdeckung

In der folgenden Tabelle sind pro Vorgehensmodell der Abstraktionsgrad, Besonderheiten hinsichtlich der Erfüllung von Anforderungen und Aussagen zu seiner Anpassbarkeit aufgelistet.

	Abstraktionsgrad	Erfüllung von Anforderungen	Anpassbarkeit
Wasserfallmodell	universell	Klare und stabile Anforderungen	kein Tailoring; stabiler Kontext
Evolutionäre Entwicklung	universell	Auch unklare und unstabile Anforderungen	kein Tailoring; auch unstabiler Kontext
Explorative Entwicklung	universell	Auch unklare und unstabile Anforderungen	kein Tailoring; auch unstabiler Kontext
Inkrementelle Entwicklung	universell	Auch unklare und unstabile Anforderungen	kein Tailoring; auch unstabiler Kontext
Spiralförmige Entwicklung	universell	Auch unklare und unstabile Anforderungen	Tailoring auch unstabiler Kontext
Prototypische Vorgehensmodelle	universell	Auch unklare und unstabile Anforderungen	kein Tailoring; auch unstabiler Kontext; Integration in andere Vorgehensmodelle
Nebenläufige Vorgehensmodelle	universell	Klare und stabile Anforderungen	kein Tailoring; stabiler Kontext
Komponentenbasierte Entwicklung	universell	Klare und stabile Anforderungen	kein Tailoring; stabiler Kontext; Integration in andere Vorgehensmodelle
Formale Systementwicklung	universell	Klare und stabile Anforderungen	kein Tailoring; stabiler Kontext
eXtreme Programming	weltlich	Auch unklare und un-	kein Tailoring;

	Abstraktionsgrad	Erfüllung von Anforderungen	Anpassbarkeit
		stabile Anforderungen	auch unstabiler Kontext
Rational Unified Process	weltlich	variabel	Tailoring; auch unstabiler Kontext
V-Modell XT	weltlich	variabel	Tailoring; auch unstabiler Kontext

Tabelle 6: Abstraktionsgrad, Erfüllung von Anforderungen und Anpassbarkeit

In der folgenden Tabelle sind die Charakteristika der Vorgehensmodelle hinsichtlich Philosophie, Prozesssteuerung und Benutzerbeteiligung aufgelistet.

	Philosophie	Prozesssteuerung	Benutzerbeteiligung
Wasserfallmodell	sequenziell; Managementaufwand minimieren; Dokumente	ergebnisorientiert	Beginn und Ende
Evolutionäre Entwicklung	iterativ; Entwicklungszeit minimieren; Code	aktivitätsorientiert	regelmäßig
Explorative Entwicklung	iterativ; Anforderungen verstehen; Code	aktivitätsorientiert	kontinuierlich
Inkrementelle Entwicklung	iterativ; Entwicklungszeit und Risiko minimieren; Code	aktivitätsorientiert	regelmäßig
Spiralförmige Entwicklung	iterativ; Risiko minimieren; Risiko	aktivitätsorientiert	regelmäßig
Prototypische Vorgehensmodelle	iterativ; Risiko minimieren; Code	aktivitätsorientiert	kontinuierlich
Nebenläufige Vorgehensmodelle	nebenläufig; Entwicklungszeit minimieren; Zeit	aktivitätsorientiert	kontinuierlich
Komponentenbasierte Entwicklung	sequenziell; Entwicklungszeit und Kosten minimieren; Wiederverwendung	aktivitätsorientiert	Beginn und Ende
Formale Systementwicklung	sequenziell; Korrektheit gewährleisten; formale Transformationen	ergebnisorientiert	Beginn und Ende
eXtreme Programming	iterativ; Kunden zufrieden stellen; Code	aktivitätsorientiert	kontinuierlich
Rational Unified Process	iterativ; Anwendungsfälle	aktivitätsorientiert	variabel
V-Modell XT	variabel; Qualität maximieren; Dokumente	ergebnisorientiert	variabel

Tabelle 7: Philosophie, Prozessteuerung und Benutzerbeteiligung

4 Aktuelle Trends bei der Anwendung und Weiterentwicklung von Vorgehensmodellen

Vorgehensmodelle weisen heute oft einen hohen Interpretationsspielraum auf. Ein hohes Maß an erforderlicher Interpretation bei der Anwendung reduziert aber das Maß an Hilfestellung, das ein Vorgehensmodell einem Projekt bietet. Eine Voraussetzung für die Erhöhung der Erfolgswahrscheinlichkeit eines Projektes durch ein Vorgehensmodell ist die präzise vorgegebene Anwendung und einfache Anwendbarkeit des Vorgehensmodells im Projekt.

Nichtsdestotrotz sollen Vorgehensmodelle einfach an den Kontext eines konkreten Projektes anpassbar und damit einfach anwendbar sein. Insbesondere das „Tailoring“ von Vorgehensmodellen, das heißt die Anpassung eines Vorgehensmodells durch Zuschneiden des Modells auf die im Kontext eines konkreten Projektes relevanten Teile, findet heute in der Praxis großes Interesse (vgl. Deubler et.al. 2003).

Daneben sind auch konkretere Vorgehensmodelle wünschenswert. Dabei kann es sich um organisationsspezifische Modelle, die sich auf das Geschäftsfeld eines Unternehmens ausrichten, oder um domänenspezifische Modelle handeln, beispielsweise für die objektorientierte Entwicklung betrieblicher Informationssysteme. Diese Einschränkung des Spektrums der Anwendbarkeit ermöglicht Vorgehensmodelle, die sich spezifischer auf bestimmte Klassen von Projekten ausrichten.

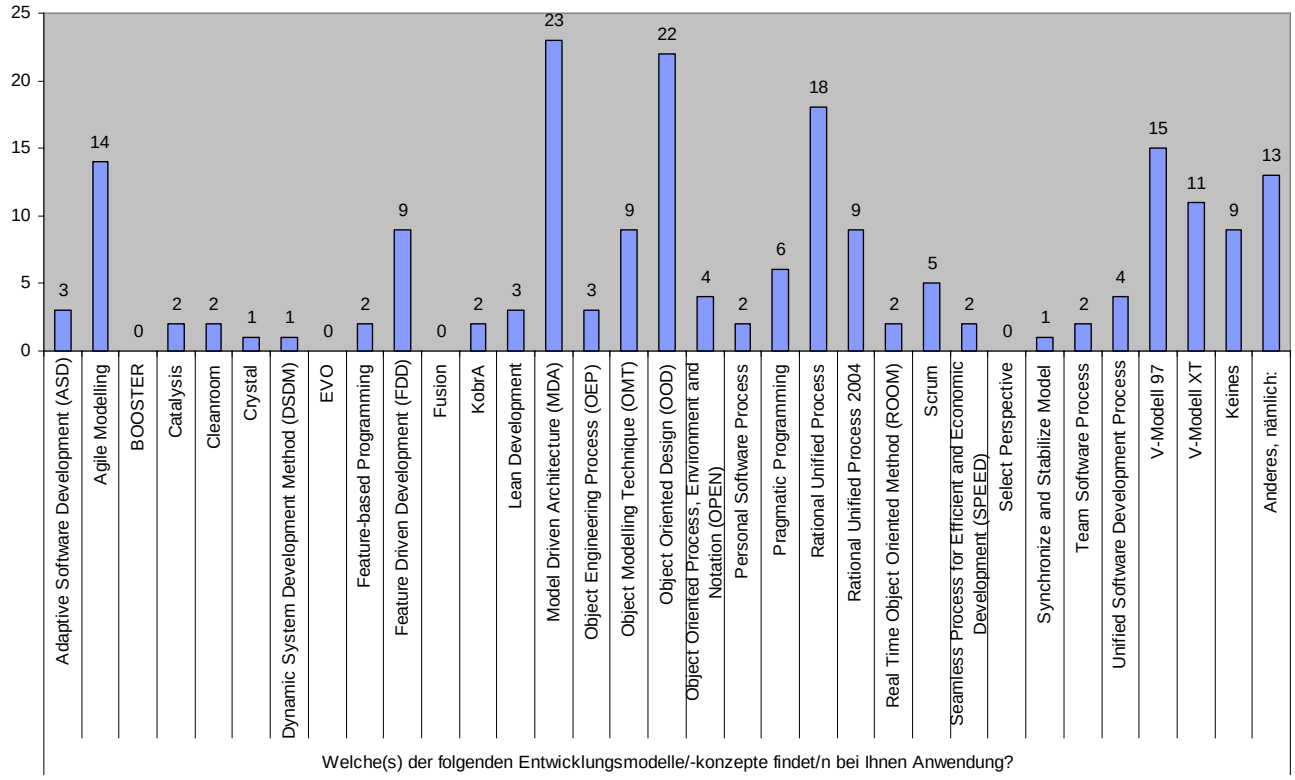
„Die neuere Entwicklung scheint in Richtung einer immer breiteren Anwendbarkeit von Vorgehensmodellen für die unterschiedlichsten Bereiche zu gehen“ (Gnatz 2005, 30).

4.1 Umfrage zu Vorgehensmodellen

Im Rahmen des Projektes IOSE-W^2 (www.iose-w.de) wurde vom Lehrstuhl für Software & Systems Engineering der TU München eine Umfrage zum Einsatz von Softwareentwicklungs-Vorgehensmodellen, Reifegradmodellen, verteilter Entwicklung und Produktlinien durchgeführt. Der dazu entwickelte Fragebogen, sollte Hinweise auf Probleme und Handlungsbedarfe der deutschen Software-Unternehmen liefern.

Die Ergebnisse der Fragen bzgl. Vorgehensmodelle sollen an dieser Stelle vorgestellt werden. Die Resultate werden in Form von Tabellen präsentiert. Interessante Ergebnisse werden kommentiert.

Welche(s) der folgenden Entwicklungsmodelle/-konzepte findet/n bei Ihnen Anwendung?

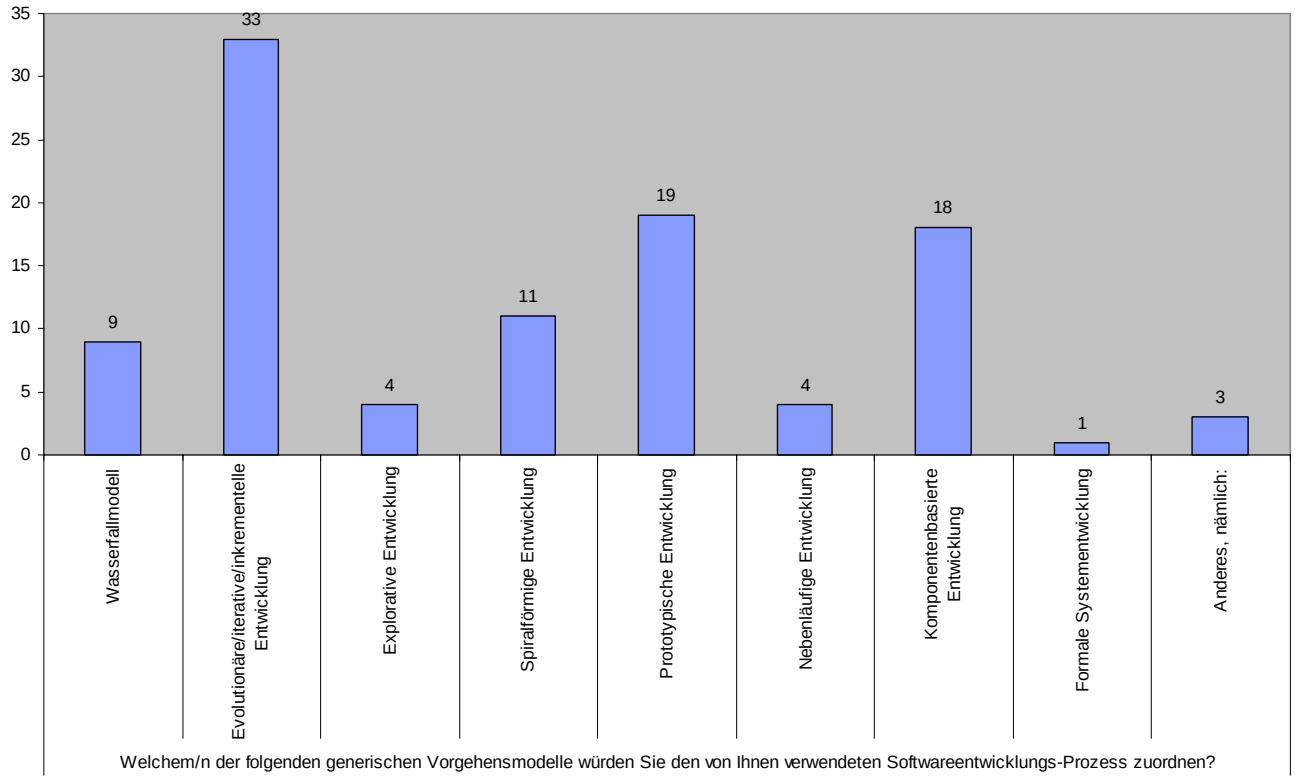


Bei „Anderes, nämlich:“ wurden folgende Antworten gegeben:

- getailortes agiles Modell basierend auf Rational Unified Process oder Kundenvorgehen
- Prototyping auf der Basis eines SAP Systems
- SAP-spez. Modelle (ASAP, PIL)
- Im Unternehmen selbst definierter Software-Entwicklungsprozess
- Hermes
- Eigenes Vorgehensmodell auf Basis von internationalen Standards (RUP, PMI, UML)
- Steinweg
- eXtreme Programming
- Management der Softwareentwicklung, Carl Steinweg, Vieweg Verlag
- PM Book, SE Book
- eXtreme Programming
- Catalan III
- eigene Entwicklungsmethoden

Auffällig ist der hohe Zahl an Entwicklungsmodellen die angegeben werden. Nur neun Unternehmen (13%) gaben an, dass sie kein Entwicklungsmodell verwenden. Besonders häufig werden Model Driven Architecture, Object Oriented Design, der Rational Unified Process sowie das V-Modell angegeben.
Mit 95% Wahrscheinlichkeit verwenden damit mehr als 78% der deutschen Unternehmen ein Entwicklungsmodell.

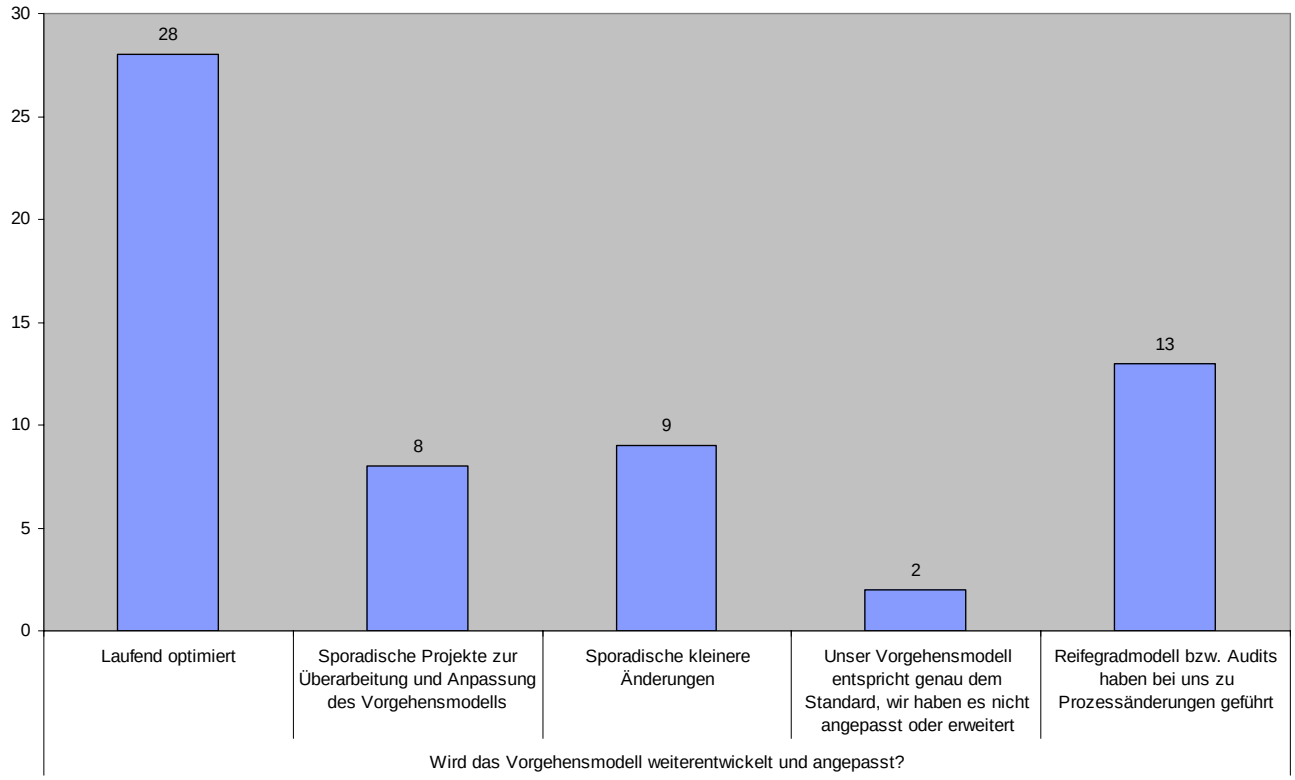
Welchem/n der folgenden generischen Vorgehensmodelle würden Sie den von Ihnen verwendeten Softwareentwicklungs-Prozess zuordnen?



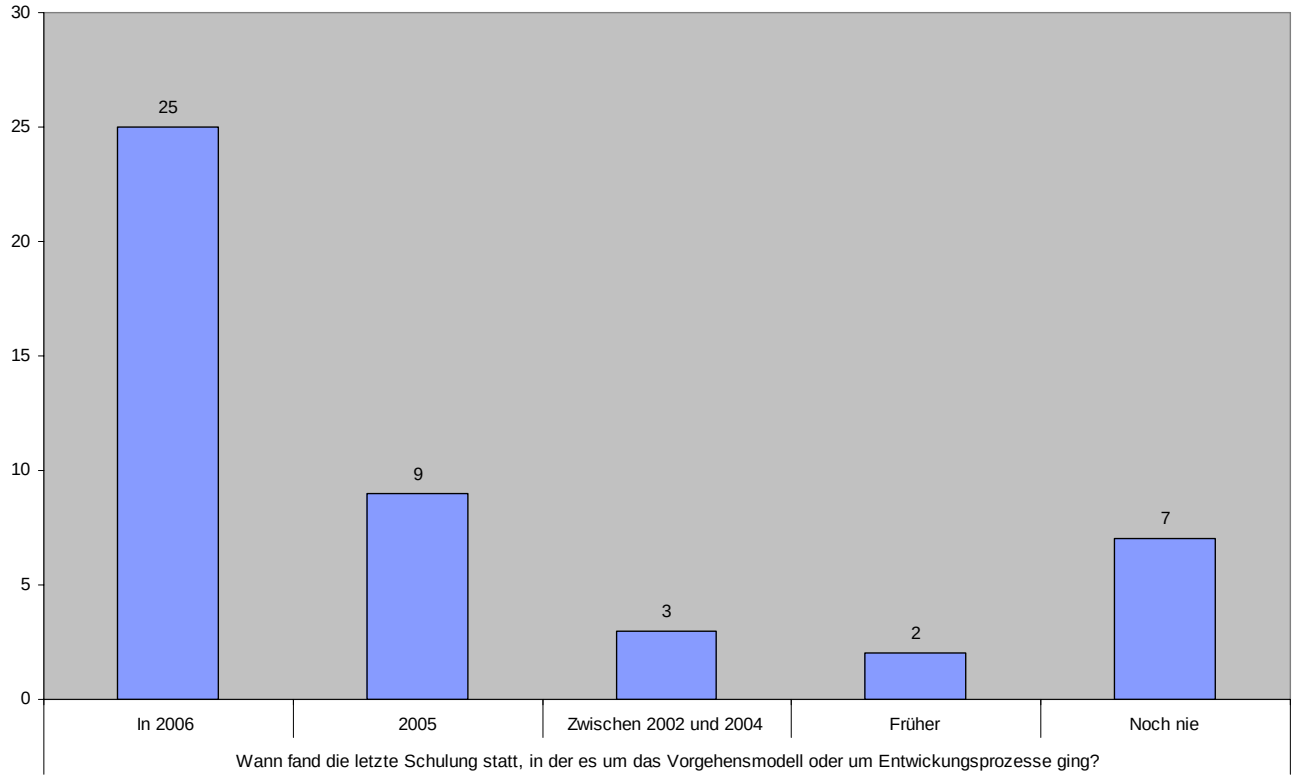
Bei „Anderes, nämlich:“ wurden folgende Antworten gegeben:

- je Projektart (Entwicklung, Einführung) unterschiedlich
- Generative Entwicklung
- Abhängig vom Umfang des Auftrags wird das Vorgehensmodell ausgewählt. Werkzeuggestützte Auswahl mit Do-It-Right

Wird das Vorgehensmodell weiterentwickelt und angepasst?



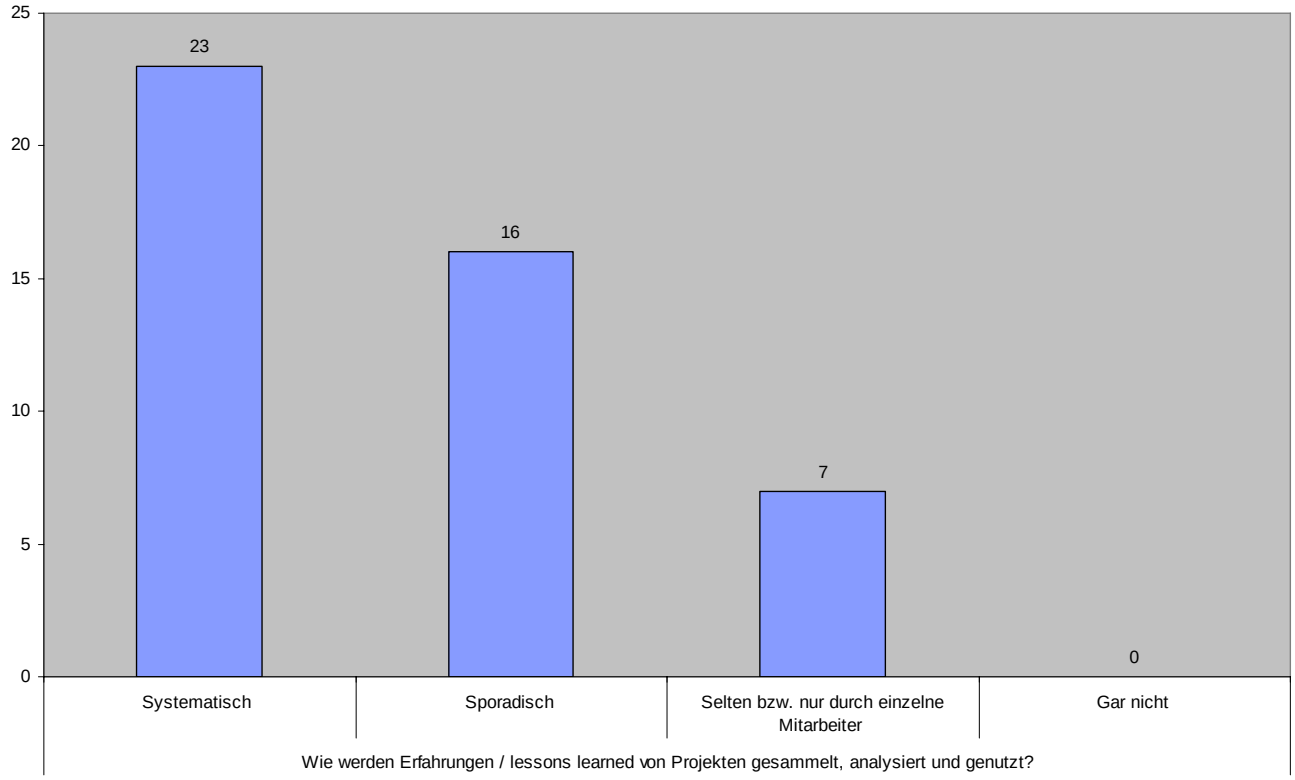
Wann fand die letzte Schulung statt, in der es um das Vorgehensmodell oder um Entwicklungsprozesse ging?



In 73,9% der Unternehmen gab es in den letzten beiden Jahren eine Schulung, in der es um ihr Vorgehensmodell oder um Entwicklungsprozesse ging.

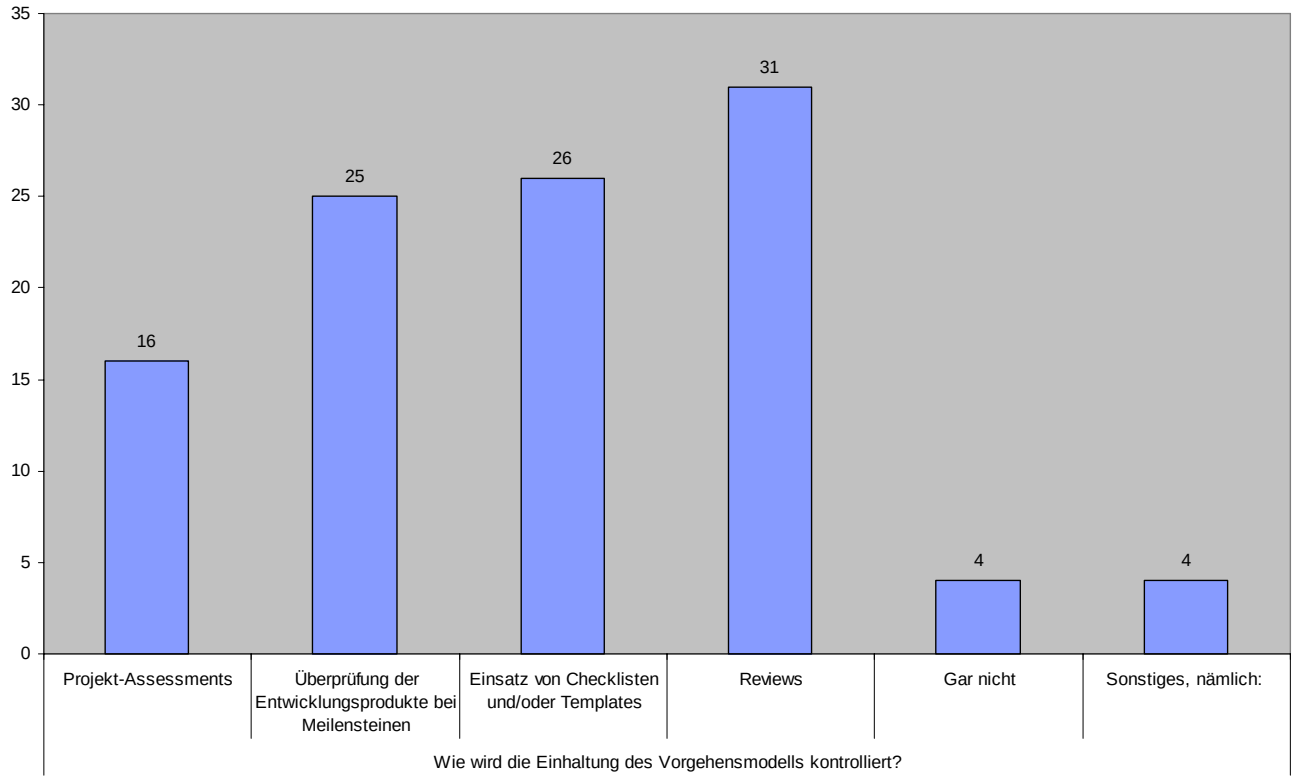
Mit 95% Wahrscheinlichkeit haben damit mehr als 61% der deutschen Unternehmen in den letzten beiden Jahren eine Schulung durchgeführt, in der es um ihr Vorgehensmodell oder um Entwicklungsprozesse ging.

Wie werden Erfahrungen / lessons learned von Projekten gesammelt, analysiert und genutzt?



50% der Befragten geben an, Erfahrungen aus Projekten systematisch zu sammeln, zu analysieren und für zukünftige Projekte zu nutzen.
Mit 95% Wahrscheinlichkeit werden damit bei mehr als 37% der deutschen Unternehmen Erfahrungen aus Projekten systematisch gesammelt, analysiert und für zukünftige Projekte genutzt.

Wie wird die Einhaltung des Vorgehensmodells kontrolliert?

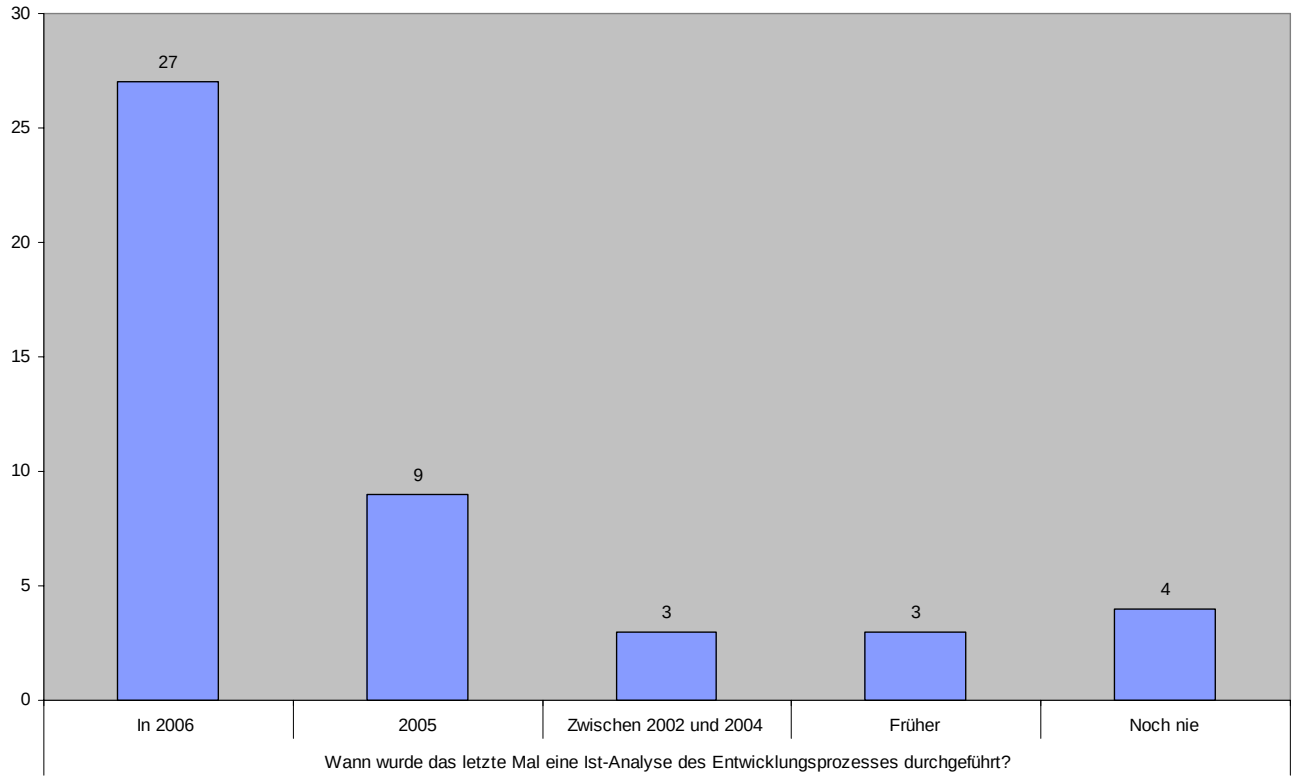


Bei „Sonstiges, nämlich:“ wurden folgende Antworten gegeben:

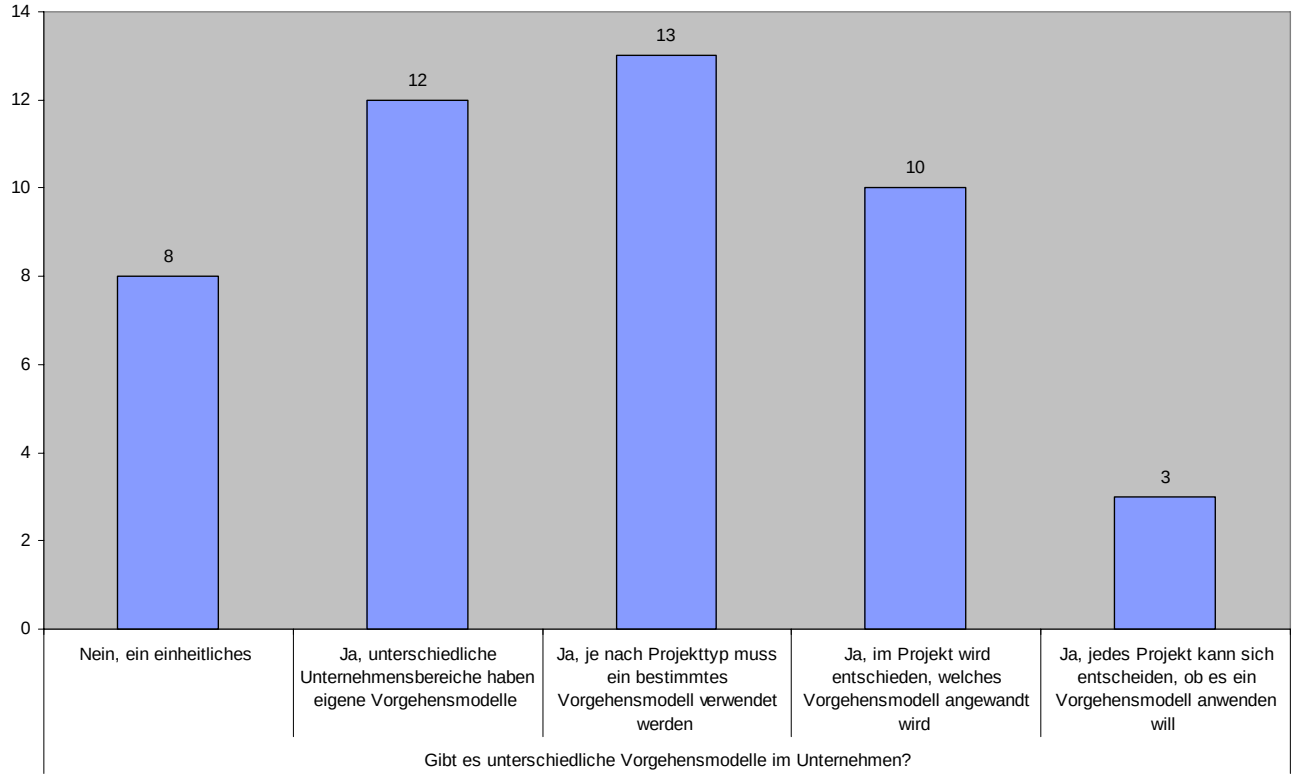
- nicht durchgängig im Unternehmen
- Projektaudits
- Project Controll Office PCO, Werkzeuggestütztes ProjectKit
- interne Audits, CMMI Checks (Kurzassessment)

Fast alle Unternehmen kontrollieren die Einhaltung ihres Vorgehensmodells.

Wann wurde das letzte Mal eine Ist-Analyse des Entwicklungsprozesses durchgeführt?

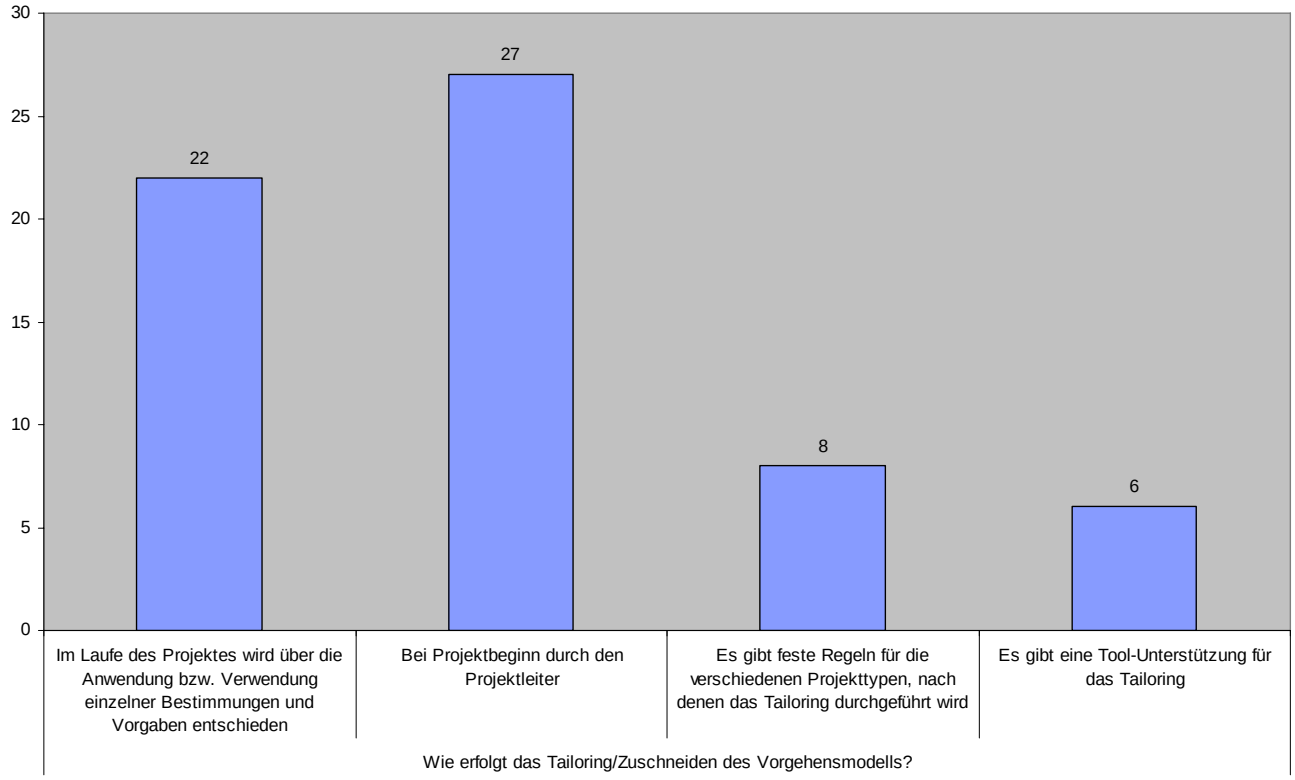


Gibt es unterschiedliche Vorgehensmodelle im Unternehmen?



82,6% der Unternehmen verwenden unterschiedliche Vorgehensmodelle.
Mit 95% Wahrscheinlichkeit verwenden damit mehr als 70% der deutschen Unternehmen unterschiedliche Vorgehensmodelle.

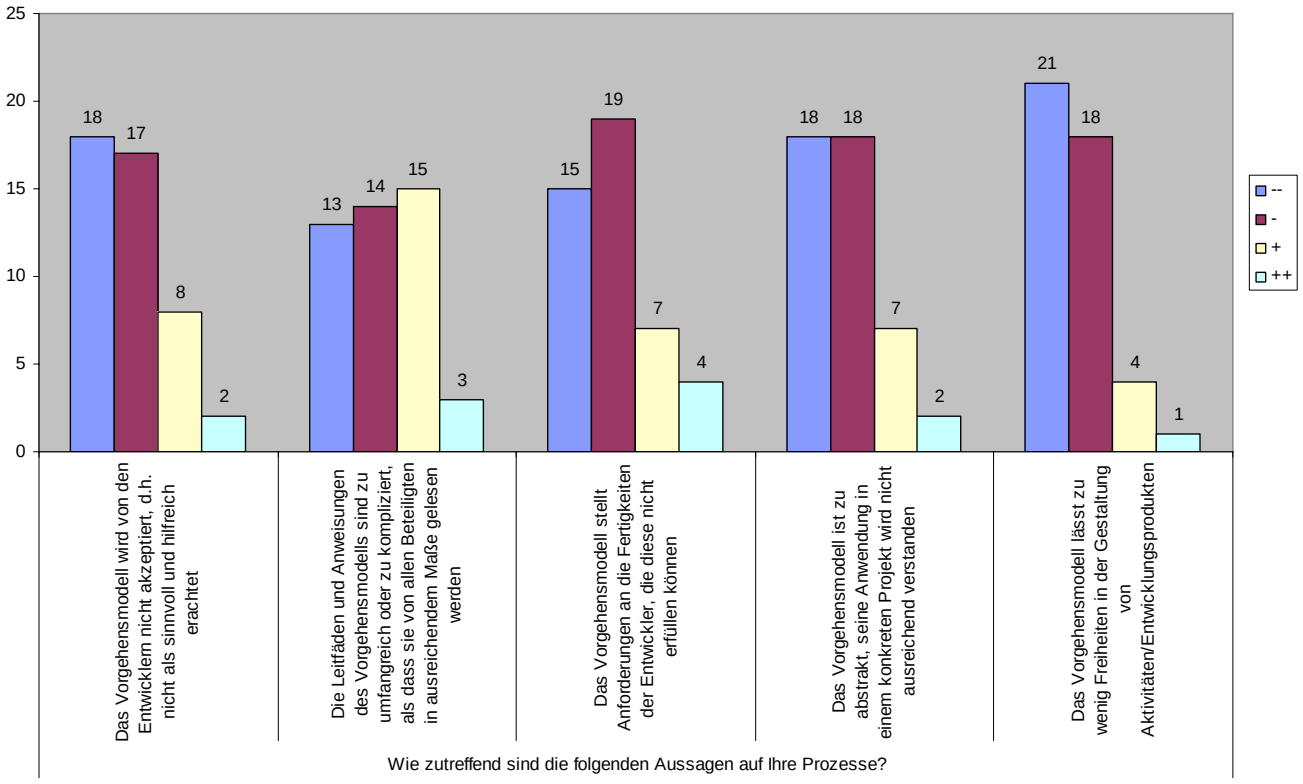
Wie erfolgt das Tailoring/Zuschneiden des Vorgehensmodells?

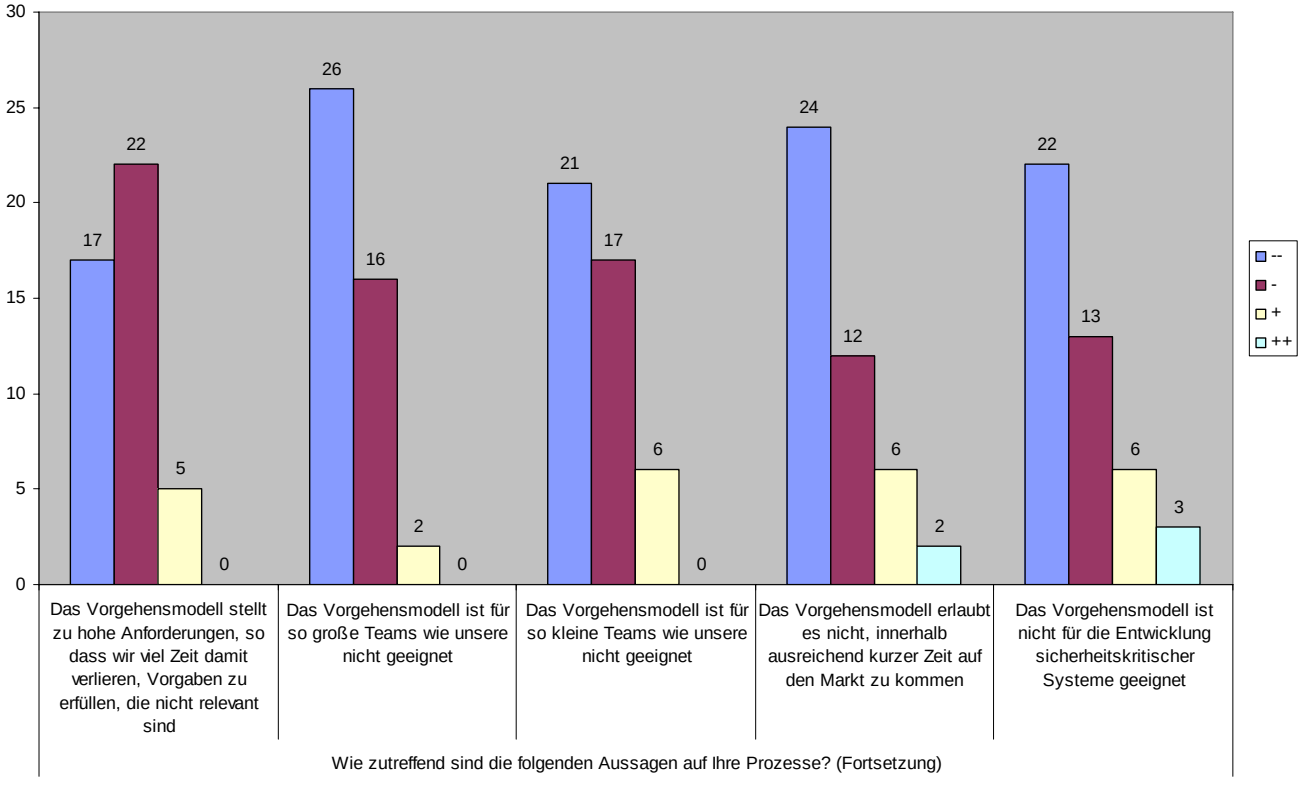


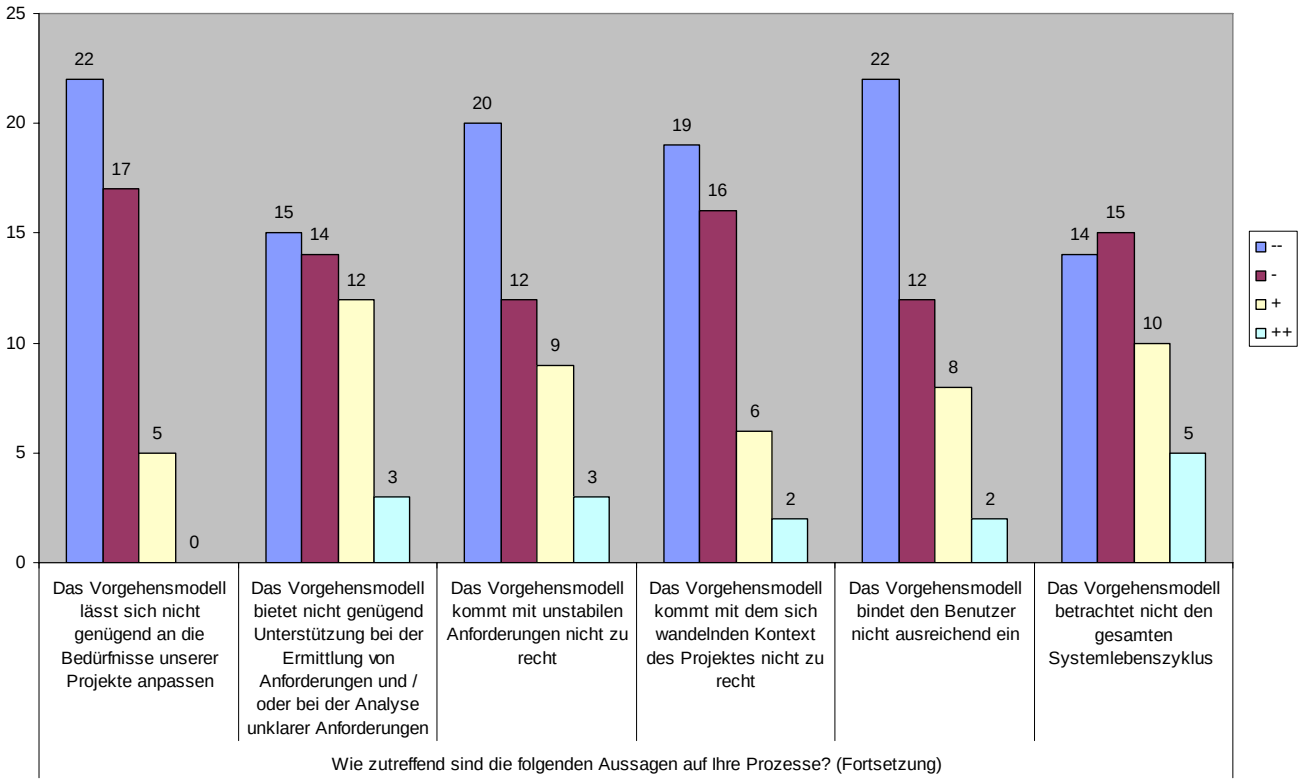
Wie zutreffend sind die folgenden Aussagen auf Ihre Prozesse?

Bei dieser Frage wurde nach der Zustimmung zu bestimmten Thesen gefragt. Dabei gab es entweder die Auswahl zwischen

- -- , bedeutet „trifft gar nicht zu“
- - , bedeutet „trifft eher nicht zu“
- + , bedeutet „trifft eher zu“
- ++ , bedeutet „trifft genau zu“.


25
20
15
10
5
0
18
17
8
2
13
14
15
3
15
19
7
4
18
18
7
2
21
18
4
1
--
-
+
++
Das Vorgehensmodell wird von den Entwicklern nicht akzeptiert, d.h. nicht als sinnvoll und hilfreich erachtet
Die Leitfäden und Anweisungen des Vorgehensmodells sind zu umfangreich oder zu kompliziert, als dass sie von allen Beteiligten in ausreichendem Maße gelesen werden
Das Vorgehensmodell stellt Anforderungen an die Fertigkeiten der Entwickler, die diese nicht erfüllen können
Das Vorgehensmodell ist zu abstrakt, seine Anwendung in einem konkreten Projekt wird nicht ausreichend verstanden
Das Vorgehensmodell lässt zu wenig Freiheiten in der Gestaltung von Aktivitäten/Entwicklungsprodukten
Wie zutreffend sind die folgenden Aussagen auf Ihre Prozesse?


30
25
20
15
10
5
0
17
22
5
0
26
16
2
0
21
17
6
0
24
12
6
2
22
13
6
3
--
-
+
++
Das Vorgehensmodell stellt zu hohe Anforderungen, so dass wir viel Zeit damit verlieren, Vorgaben zu erfüllen, die nicht relevant sind
Das Vorgehensmodell ist für so große Teams wie unsere nicht geeignet
Das Vorgehensmodell ist für so kleine Teams wie unsere nicht geeignet
Das Vorgehensmodell erlaubt es nicht, innerhalb ausreichend kurzer Zeit auf den Markt zu kommen
Das Vorgehensmodell ist nicht für die Entwicklung sicherheitskritischer Systeme geeignet
Wie zutreffend sind die folgenden Aussagen auf Ihre Prozesse? (Fortsetzung)


25
20
15
10
5
0
22
17
5
0
15
14
12
3
20
12
9
3
19
16
6
2
22
12
8
2
14
15
10
5
--
-
+
++
Das Vorgehensmodell lässt sich nicht genügend an die Bedürfnisse unserer Projekte anpassen
Das Vorgehensmodell bietet nicht genügend Unterstützung bei der Ermittlung von Anforderungen und / oder bei der Analyse unklarer Anforderungen
Das Vorgehensmodell kommt mit unstabilen Anforderungen nicht zu recht
Das Vorgehensmodell kommt mit dem sich wandelnden Kontext des Projektes nicht zu recht
Das Vorgehensmodell bindet den Benutzer nicht ausreichend ein
Das Vorgehensmodell betrachtet nicht den gesamten Systemlebenszyklus
Wie zutreffend sind die folgenden Aussagen auf Ihre Prozesse? (Fortsetzung)

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

Balzert, H., Lehrbuch der Software-Technik - Software-Managament, Software-Qualitätssicherung und Unternehmensmodellierung, Heidelberg, Spektrum Akademischer Verlag, 1998.

Beck, K., *Extreme Programming explained - Embrace change*, 3. Aufl., Boston, Addison-Wesley, 2000 a.

Beck, K., *Extreme Programming - Das Manifest*, München, Addison-Wesley, 2000 b.

Benington, H. D., "Production of Large Computer Programs", *Symposium on Advanced Programming Methods for Digital Computers, Proceedings*, Washington, 1956, S. 15-27.

Biskup, H., „Nutzen und Nutzung von Vorgehensmodellen", *13.Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik*, 2006.

BMBF-Studie, *Analyse und Evaluation der Softwareentwicklung in Deutschland*, 2000, http://www.isi.fhg.de/publ/downloads/isi00b69/software.pdf (2.8.2006).

Boehm, B. W., "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, **21**(5), 1988, S. 61-72.

Boehm, B. W. und Turner, R., *Balancing Agility and Discipline – A Guide for the Perplexed*, Boston, Addison-Wesley, 2004.

Bradac, M. G., Perry, D. E. und Votta, L. G., "Prototyping a Process Monitoring Experiment", *IEEE Transactions on Software Engineering*, **20**(10), 1994, S. 774-784.

Budde, R. et al, *Prototyping - An Approach to Evolutionary System Development*, Heidelberg, Springer-Verlag, 1992.

Bunse, C. und Knethen, A. von, *Vorgehensmodelle kompakt*, Heidelberg, Spektrum Akademischer Verlag, 2002.

Butler, J., "Rapid Application Development in Action" *Managing System Development, Applied Computer Research*, **14**(5), 1994, S.6-8.

Chroust, G., *Modelle der Software-Entwicklung*, München, Oldenbourg Verlag, 1992.

Cockburn, A., *Crystal Clear – A Human-Powered Methodology for Small Teams*, Boston, Addison-Wesley, 2005.

Davis, A. M., Bersoff, E. H. und Comer, E. R., "A strategy for comparing alternative software development lifecycle models", *IEEE Transactions on Software Engineering*, **14**(10), 1988, S. 1453-1461.

Dogs, C. und Klimmer, T., Agile *Software-Entwicklung kompakt*, Bonn, mitp-Verlag, 2005.

D'Souza, D. F. und Wills, A. C., *Objects, Components and Frameworks with UML – The Catalysis Approach*, Boston, Addison-Wesley, 1998.

Dyer, M., *The Cleanroom Approach to Quality Software Development*, New York, Wiley, 1992.

Eckstein, J., *Agile Softwareentwicklung im Großen – Ein Eintauchen in die Untiefen Erfolgreicher Projekte*, Heidelberg, dpunkt.verlag, 2004.

Essigkrug, A. und Mey, T., *Rational Unified Process kompakt*, Heidelberg, Spektrum Akademischer Verlag, 2003.

Filßer, C., *Vergleichsmethoden für Vorgehensmodelle*, Dipl.-Arb., Dresden, 1995.

Gnatz, M., *Vom Vorgehensmodell zum Projektplan*, Diss., München, 2005.

Gordon, V. S. und Bieman, J. M., „Rapid Prototyping - Lessons Learned“, *IEEE Software*, **12**(1), 1995, S.85-95.

Hammer, M. und Champy, J., *Reengineering the Corporation - A Manifesto for Business Revolution*, New York, Harper Business, 1993.

Hanks, B. F., „Distributed Pair Programming - An Empirical Study", *Extreme Programming and Agile Methods - XP/ Agile Universe 2004, Proceedings*, 2004, S. 81-91.

Highsmith, J. A., *Adaptive Software Development – A Collaborative Approach to Managing Complex Systems*, New York, Dorset House Publishing, 2000.

Hildenbrand, T. et al, *Entwicklungsmethodiken zur kollaborativen Softwareerstellung - Stand der Technik*, 2006, http://wifo1.bwl.uni-mannheim.de/fileadmin/files/publications/CollaBaWue-KSE-Arbeitsbericht-Stand_der_Technik-Methodiken.pdf (16.6.2006).

Humphrey, W.S., *Managing the Software Process*, Boston, Addison-Wesley, 1989.

Humphrey, W., *Managing the Software Process*, Boston, Addison-Wesley, 1990.

Humphrey, W., *Introduction to the Personal Software Process*, Boston, Addison-Wesley, 1997.

Humphrey, W., *Introduction to the Team Software Process*, Boston, Addison-Wesley, 2000.

Humphrey, W., „Comments on eXtreme Programming”, in *eXtreme Programming Pros and Cons - What Questions Remain?, IEEE Computer Society Dynabook*, 2001.

Janker, M., *(Ways to Improve) Software Economics*, 2002, http://www.cosy.sbg.ac.at/~pmm/teaching/SS02/se/papers/0020025.pdf (27.02.2007).

Kettunen, P. und Laanti, M., „How to Steer an Embedded Software Project - Tactics for Selecting the Software Process Model“, *Information and Software Technology*, **47**(9), 2005.

Kneuper, R., Müller-Luschnat, G. und Oberweis, A., *Vorgehensmodelle für die betriebliche Anwendungsentwicklung*, Stuttgart, Teubner, 1998.

Kruchten, P., Der *Rational Unified Process*, München, Pearson Education Deutschland, 1999.

Kruchten, P., *The Rational Unified Process - An Introduction*, Boston, Addison-Wesley 2003.

Larman, C., *Agile & Iterative Development – A Mangers Guide*, Boston, Addison-Wesley 2004.

Lippert, M., Roock, S. und Wolf, H., *Software entwickeln mit eXtreme Programming – Erfahrungen aus der Praxis*, Heidelberg, dpunkt Verlag, 2002.

Mathiassen, L., Seewaldt, T. und Stage, J., „Prototyping and Specifying: Principles and Practices of a Mixed Approach“, *Scandinavian Journal of Information Systems*, **7**(1), 1995.

McConnell, S., *Rapid Development – Taming Wild Software Schedules*, Redmond, Microsoft Press, 1996.

McFeeley, B., *IDEAL SM - A User's Guide for Software Process Improvement*, CMU/SEI-96-HB-001, 1996.

Models for Action Project, *A Survey of System Development Process Models*, http://www.ctg.albany.edu/publications/reports/survey_of_sysdev/survey_of_sysdev.pdf (19.5.2006).

Noack, J. und Schienmann, B., „Objektorientierte Vorgehensmodelle im Vergleich“, *Informatik-Spektrum*, **22**(3), 1999, S. 166-180.

Oestereich, B. et al, Erfolgreich mit Objektorientierung – Vorgehensmodelle und Managementpraktiken für die objektorientierte Softwareentwicklung, München, Oldenbourg Verlag, 1999.

Palmer, S. R und Felsing, J. M., *A Practical Guide to Feature-Driven Development*, Upper Saddle River, Prentice Hall, 2002.

Paulk, M. C., „Extreme Programming from a CMM Perspective”, *IEEE Software*, **18**(6), 2001.

Pressman, R. S., Software *Engineering - A Practitioner’s Approach*, Boston, McGraw-Hill, 2001.

Prowell, S. J. et al, *Cleanroom Software Engineering - Technology and Process*, Boston, Addison-Wesley, 1999.

Read, K. und Maurer, F., „Issues in Scaling Agile Using an Architecture-Centric Approach - A Tool-Based Solution“, *Extreme Programming and Agile Methods – XP/Agile Universe 2003, Proceedings of the Third XP and Second Agile Universe Conference*, New Orleans, 2003.

Reeves, M. und Zhu, J., „Moomba – A Collaborative Environment for Supporting Distributed Extreme Programming in Global Software Development“, in Extreme *Programming And Agile Processes in Software Engineering*, Berlin, Springer, 2004.

Rezagholi, M., *Prozess- und Technologiemanagement in der Softwareentwicklung –Ein Metrik basierter Ansatz zur Bewertung von Prozessen und Technologien*, München, Oldenbourg, 2004.

Richter, S., *Feature-based Programming – Planung, Programmierung, Projektmanagement - Über die Kunst systematisch zu planen und mit Agilität umzusetzen*, München, Addison-Wesley, 2003.

Robinson, H. und Sharp, H., „The Characteristics of XP Teams in Agile Software Development“, *Extreme Programming and Agile Processes in Software Engineering: 5th International Conference*, Garmisch-Partenkirchen, 2004, S. 139-147.

Rumpe, B., *Agile Softwareentwicklung mit der UML*, Habil.-Schr., München, 2003.

Schwaber, K. und Beedle, M., *Agile Software Development with Scrum*, Upper Saddler River, Prentice Hall, 2002.

Sommerville, I., *Software Engineering*, Boston, Addison Wesley, 1989 (7.Aufl. 2004).

Sommerville, I., *Software Engineering*, 6.Aufl., München, Pearson Studium, 2001.

Spinellis, D., “Taking Common Sense to the Extreme”, in *eXtreme Programming Pros and Cons: What Questions Remain?, IEEE Computer Society Dynabook*, 2001.

Succi, G. und Marchesi, M., *Extreme Programming Examined*, Boston, Addison-Wesley, 2001.

Versteegen, G., *Projektmanagement mit dem Rational Unified Process*, Berlin, Springer 2000.

V-Modell XT Release 1.2, http://ftp.uni-kl.de/pub/v-modell-xt/Release-1.2/Dokumentation/pdf/V-Modell-XT-Komplett.pdf (14.8.2006).

Wake, W. C., *Extreme Programming Explored*, Boston, Addison-Wesley, 2001.

Wäyrynen, J., Bodén, M. und Boström, G., „Security Engineering and eXtremeProgramming - An Impossible Marriage?”, *Extreme Programming and Agile Methods – XP/ Agile Universe 2004, Proceedings of 4th Conference on Extreme Programming and Agile Methods*, Calgary, 2004.

Wolf, H., Roock, S. und Lippert, M., *eXtreme Programming in Action – Practical Experiences from Real World Projects*, New York, John Wiley, 2002.