Kapitel

3 Software Engineering-Prozesse

Die Arbeit kann man aufteilen, den Geist, der über sie herrscht, nicht.

Oscar Wilde

Begriffe in diesem Kapitel

Softwareentwicklungsprozess (*Entwicklungsprozess, Prozess*): Beschreibt die Summe aus einem Vorgehensmodell, den konkreten Aktivitäten und deren Abfolge sowie den angewandten Methoden.

Vorgehensmodell: Ein Vorgehensmodell bestimmt die Abfolge von Phasen und Meilensteinen eines Projekts. Entsprechend der Abfolge kann man sequenzielle und iterative Vorgehensmodelle unterscheiden. In sequenziellen Vorgehensmodellen wird eine bestimmte Phase (die in solchen Modellen meist mit einem Arbeitsschritt gleichzusetzen ist) einmal durchlaufen. Der Anfang und das Ende jeder Phase sind durch Meilensteine festgelegt. In iterativen Vorgehensmodellen werden Phasen mehrmals durchlaufen, um zu einer höheren Produktreife zu gelangen. Die Kriterien für den Übertritt von einer Phase zur nächsten bzw. einer Iteration zur nächsten werden meist in Abhängigkeit vom aktuellen Projektstatus zu Beginn einer Phase bzw. einer Iteration festgelegt.

Phase: Eine Zeitspanne im Projekt, welche sich durch einen besonderen Fokus kennzeichnet. Der Fokus bezieht sich auf das Gesamtprojekt, nicht auf einzelne Produkte oder Arbeitsschritte.

Arbeitsschritt: Ein Arbeitsschritt ist ein inhaltlich abgeschlossener Teil eines Projekts, dessen Aktivitäten und durchführende Rollen sich grundlegend von den Aktivitäten und ausführenden Rollen eines anderen Arbeitsschritts unterscheiden. Ein Arbeitsschritt erzeugt inhaltlich zusammengehörende und auf demselben Abstraktionsniveau befindliche Produkte als Grundlage für einen folgenden Arbeitsschritt.

Aktivität: Ein in sich abgeschlossener Arbeitsprozess, dessen Ergebnis ein konkretes Produkt oder eine Leistung ist.

Bei der Betrachtung von Software Engineering Prozessen gibt es leider nach wie vor eine große Begriffsverwirrung, die sich durchaus auch in der täglichen Arbeit niederschlägt, da Kommunikation über Software Engineering Prozesse damit schwierig ist. Dieses

Kapitel erklärt in den beiden ersten Abschnitten die notwendige Begrifflichkeit und stellt einige abstrakte Prozessmodelle vor. Die weiteren Abschnitte des Kapitels widmen sich der Einführung von gängigen Prozessen bzw. Frameworks, allen voran natürlich der Unified Process.

3.1 Makroprozesse – Vorgehensmodelle

Ein Vorgehensmodell entspricht einer Strategie für die Durchführung eines Projekts. Die Art und Weise, wie die Arbeitsschritte angeordnet und durchlaufen werden, sowie die Definition von übergeordneten Phasen sind in den einzelnen Vorgehensmodellen sehr unterschiedlich. In unterschiedlichen Softwareentwicklungsprozessen, die den inhaltlichen Vorgaben einer Methode folgen, unterscheidet sich somit nur die Strategie, jedoch nicht der tägliche Arbeitsprozess in Form der Arbeitsschritte und Tätigkeiten. Aus diesem Grund ist es meist auch möglich, in einem Unternehmen je nach Projekt mehrere verschiedene Vorgehensmodelle zu verwenden, ohne dass sich für den einzelnen Entwickler in seiner täglichen Arbeit etwas ändert.

3.1.1 Build-and-Fix-Cycle

Hierbei handelt es sich um das einfachste Vorgehen, welches jeder Programmierer zweifellos bereits angewendet hat. Jemand hat eine Idee für eine bestimmte Software. Er implementiert dieses System entsprechend der Vorstellungen in seinem Kopf. Dieses System wird so lange entwickelt, bis es den Qualitätsansprüchen seines Programmierers genügt. Treten während des folgenden Betriebs Änderungswünsche auf, werden diese vom Programmierer in das System eingebracht.

Es existiert keinerlei Dokumentation, kein strukturiertes Vorgehen bzw. keine Einteilung in Arbeitsschritte wie Analyse oder Entwurf. Alle Tätigkeiten werden vom Programmierer zum Zeitpunkt der Erstellung der Software durchgeführt. Software, die auf diese Weise entsteht, kann durchaus von hoher Qualität sein und manche nützliche Anforderung umsetzen. Der Umfang von so entstehender Software ist jedoch äußerst begrenzt, die Wartung kann ausschließlich vom Programmierer selbst durchgeführt werden und auch der Einsatz durch andere Personen ist meist nur bedingt möglich.

3.1.2 Der Software-Life-Cycle

Die im Folgenden vorgestellten Vorgehensmodelle basieren auf der grundlegenden Idee des Software-Life-Cycle ([Pomb93]), welcher ein strukturiertes Vorgehen bei der Erstellung von Software vorsieht. Während der Lebensdauer eines Systems ist eine zumindest einmalige sequenzielle Durchführung aller notwendigen Arbeitsschritte (Anforderungen, Analyse, Entwurf, Implementierung, Test und Inbetriebnahme/Wartung) vorgesehen. Änderungen der Anforderungen werden erst im Zuge eines neuen Projekts durchgeführt,

welches wiederum alle Arbeitsschritte in der vorgesehenen Reihenfolge durchläuft. Das größte Problem bei einem solchen starren Vorgehen stellt die fehlende Möglichkeit eines oder mehrerer Schritte zurück dar, wenn bereits während der Durchführung eines Projekts (auch im Zuge der Wartung) Änderungen der Anforderungen notwendig werden sollten.

3.1.3 Das Wasserfallmodell

Die fehlende Möglichkeit von Rückschritten berücksichtigt zumindest teilweise das Wasserfallmodell ([Royc70]). Dieses sieht eine Abfolge der Arbeitsschritte Anforderungen, Analyse, Entwurf, Implementierung, Test und Betrieb vor und lässt einen Rückwärtsschritt von einem Arbeitsschritt auf den direkten Vorgänger zu. Es bleibt jedoch die Bedingung bestehen, dass ein Arbeitsschritt erst dann abgeschlossen werden kann, wenn alle vorgesehenen Produkte dafür fertig gestellt worden sind. Dies soll zu einer Risikominimierung für den nächsten Arbeitsschritt führen. Das Wasserfallmodell ist ein stark verbreitetes Vorgehensmodell, von dem zahlreiche Varianten beschrieben wurden. In [Birr88] findet sich eine Übersicht über mögliche Ausprägungen des Software-Life-Cycle und des Wasserfallmodells. Zwei Beispiele davon sind in *Abbildung 3.1* dargestellt.

Das Wasserfallmodell eignet sich gut für Projekte, in denen die Arbeitsschritte deutlich voneinander getrennt werden können. Dazu müssen alle Risiken bereits vor Projektbeginn ausreichend ausgeschlossen werden können. Weiters lassen sich Projekte mit dem Wasserfallmodell bei kleinen Arbeitsgruppen gut durchführen, da alle Mitarbeiter gleichzeitig an einem Arbeitsschritt arbeiten können. Bei großen Arbeitsgruppen ist es aufgrund der Anzahl und der unterschiedlichen Qualifikationen unmöglich, alle Gruppenmitglieder an einem Arbeitsschritt arbeiten zu lassen. Für solche Projekte sollte ein iteratives Modell gewählt werden.

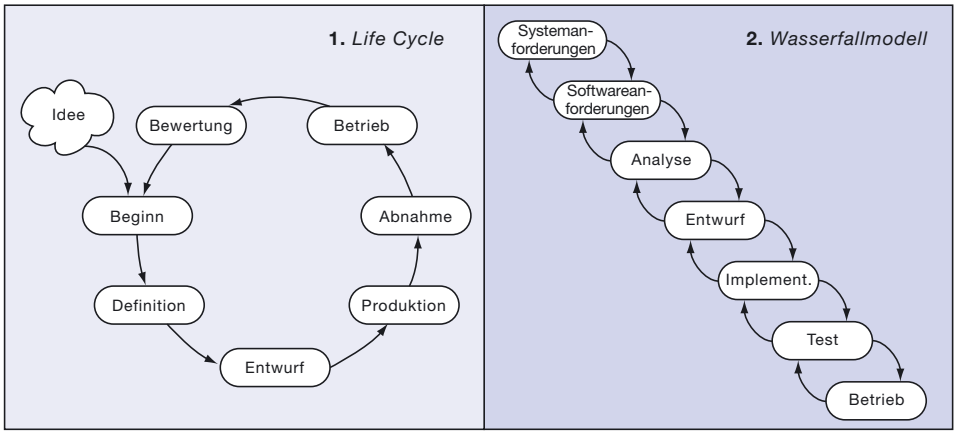

Abbildung 3.1: Varianten des Software-Life-Cycle und des Wasserfallmodells

3.1.4 Das V-Modell

Das V-Modell (siehe [Boeh79]) gliedert den Softwareentwicklungsprozess in miteinander korrelierende Phasen. Die zweite Hälfte der Phasen bildet die Tests für die Produkte der ersten Phasen. Der Klassentest testet beispielsweise den Code, welcher aus der Implementierung hervorgegangen ist und den höchsten Detaillierungsgrad aller im Laufe der Softwareentwicklung entstandenen Produkte aufweist. Die Produkte und ihre Tests entsprechen den möglichen verschiedenen Sichten auf ein System. Aus der Anordnung nach diesen Sichten ergibt sich das für dieses Modell namensgebende „V“.

Das V-Modell ist kein eigenständiges Vorgehensmodell im eigentlichen Sinn. Es fügt lediglich dem Wasserfallmodell, die Produkte im V-Modell werden ja nach wie vor streng sequenziell erstellt, eine zusätzliche strukturelle Sicht hinzu, nämlich dass für jedes zu erstellende Produkt ein entsprechend korrespondierender Test existieren und durchgeführt werden sollte.

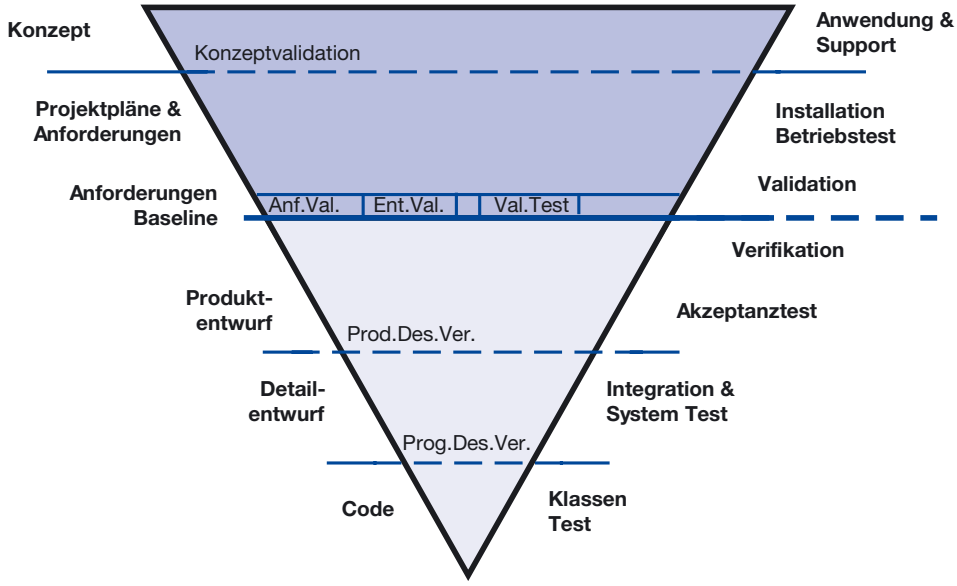

Abbildung 3.2: Das V-Modell (nach [Boeh79])

Barry Boehm ist Professor für Software Engineering an der University of Southern California und Leiter des Centers for Software Engineering an dieser Universität. Barry Boehm setzt seit 30 Jahren regelmäßig wegweisende Akzente in der wissenschaftlichen Beschäftigung mit Softwareentwicklung. Er ist unter anderem verantwortlich für entscheidende Beiträge zu den Themen COCOMO, Spiralmodell oder Software Engineering Economics.

3.1.5 Das Spiralmodell

Das Spiralmodell trägt den möglichen Projektrisiken als erstes der beschriebenen Modelle Rechnung. Der gesamte Prozess ist in vier Phasen gegliedert, welche im Zuge einer evolutionären Softwareentwicklung mehrmals durchlaufen werden. Das heißt, dass in jedem Durchlauf nur bestimmte Produkte entwickelt werden, welche auf den Produkten des vorhergehenden Durchlaufs aufbauen und für den nächsten Durchlauf als Grundlage dienen. Die vier Phasen und deren wichtigste Tätigkeiten sind:

- ***Zielbestimmung***: Jede Phase beginnt mit der Bestimmung der genauen Ziele und Produkte dieses Durchlaufs. Es werden auch Alternativen (z.B. Entwurfsvarianten) und Restriktionen (z.B. aufgrund des Zeitplans) untersucht und festgehalten.
- ***Risikoanalyse***: In der nächsten Phase werden die Ziele und Alternativen unter Berücksichtigung der Restriktionen bewertet und die darin enthaltenen Risiken festgestellt. Für die gefundenen Risiken werden Lösungsstrategien zur Beseitigung der Ursachen entwickelt (z.B. Erstellung eines Prototypen, Simulation, Befragung der Benutzer usw.).
- ***Arbeitsschritte durchführen***: In dieser Phase werden die für diesen Durchlauf vorgesehenen Produkte erstellt. Die in *Abbildung 3.3* dargestellten Produkte sind nur ein Beispiel für eine mögliche Auswahl von Produkten für ein bestimmtes Projekt. Je nach Projekt und identifizierten Risiken können die Anzahl und Abfolge der Produkte in den einzelnen Durchläufen stark variieren.
- ***Nächste Phase planen***: Basierend auf den Ergebnissen eines Reviews, welches als Abschluss für jeden Durchlauf vorgesehen ist, wird der nächste Zyklus geplant.

Abbildung 3.3 zeigt eine mögliche Form des Spiralmodells mit vier Durchläufen. Der letzte Durchlauf, welcher erst nach der Beseitigung des letzten Risikos durchlaufen wird, zeigt die Erstellung des Systems. Die Systemerstellung selbst kann zum Beispiel durch ein eingebettetes Wasserfall- oder V-Modell strukturiert werden.

Das Spiralmodell kann für sehr große und komplexe Projekte verwendet werden, da es der Komplexität durch das risikogesteuerte Vorgehen Rechnung trägt. Die Anzahl der Durchläufe ergibt sich erst während des Projekts und wird durch die auftretenden Risiken bestimmt. Dies hat zur Folge, dass zu Beginn des Projekts ein Zeit- und Kostenplan nur schwer zu erstellen ist. Die Risikoanalyse kann nur durch erfahrene Projektleiter durchgeführt werden. Bei zu zaghaftem Vorgehen kann sich das Projekt unnötigerweise verlängern, was zu erhöhten Kosten führt. Zu schnelles Vorgehen kann Risiken vernachlässigen und zu Problemen in folgenden Durchläufen führen.

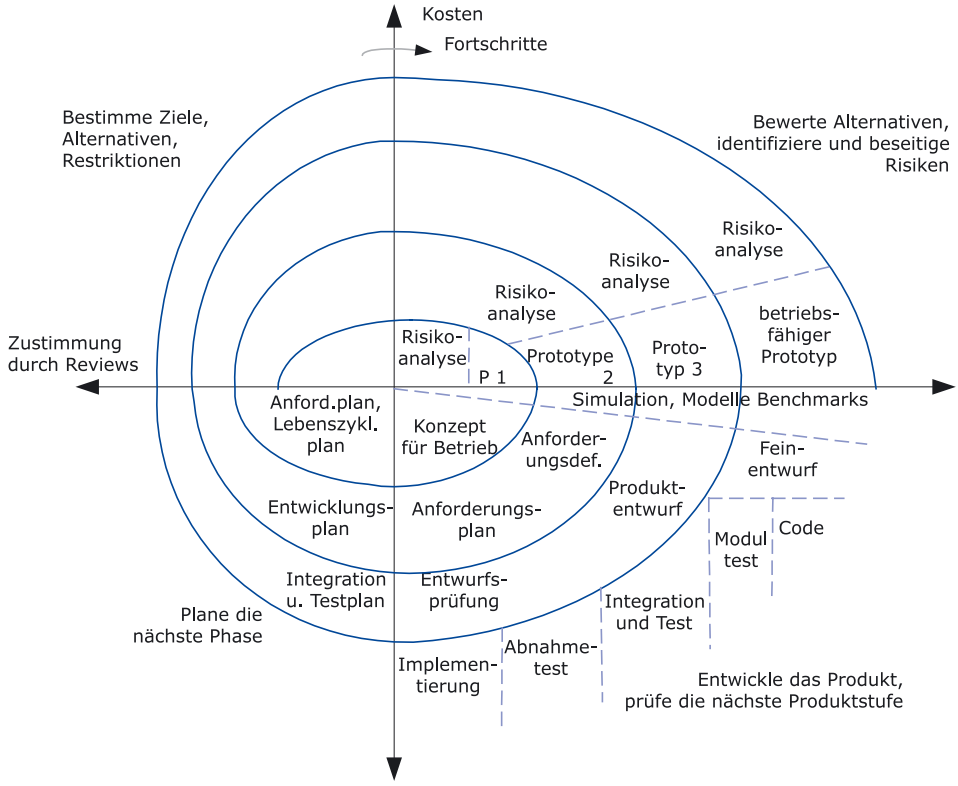

Abbildung 3.3: Das Spiralmodell (nach [Boeh88])

3.1.6 Das inkrementelle Modell

Im Wasserfallmodell und auch im V-Modell ist die abgeschlossene Analyse Voraussetzung für den Entwurf und die Implementierung. Gibt es unklare Anforderungen oder sind die Anforderungen noch nicht komplett, kann mit dem Entwurf nicht begonnen werden, ehe nicht die letzte Anforderung restlos geklärt ist. Nach erfolgter Implementierung bekommt der Kunde das fertige System als Ganzes geliefert.

Bei großen Projekten hat dies zur Folge, dass die Auslieferung des Produkts ein oder mehrere Jahre dauern kann. Der Kunde muss eine lange Zeit warten, bis er sein System begutachten kann. Noch dazu ist es bei Projekten in dieser Größe sehr leicht möglich, dass sich Anforderungen aufgrund des langen Zeitraums ändern. Somit besteht das Risiko, dass das ausgelieferte Produkt bereits nicht mehr den aktuellen Anforderungen entspricht und eine sofortige Überarbeitung notwendig ist, oder der Kunde mit Einschränkungen auskommen muss.

Das inkrementelle Modell versucht, diese Probleme zu lösen. Anstatt auf die Fertigstellung der gesamten Anforderungen zu warten, werden – sobald eine ausreichende Anzahl

an Kernanforderung beschrieben ist – für diese ein Entwurf und die Implementierung gestartet. Je mehr Anforderungen hinzukommen, desto mehr zusätzliche Inkremente werden gestartet.

Nach [Balz98] hat das inkrementelle Modell folgende Eigenschaften:

- Die Entwicklung des Software-Produkts erfolgt stufenweise. Die Entwicklung wird durch die Erfahrungen der Entwickler und des Kunden mit dem wachsenden Produkt gesteuert.
- Die Wartung wird als Erstellung einer neuen Version des bestehenden Produkts betrachtet.
- Dieses Vorgehensmodell ist gut geeignet, wenn der Kunde seine Anforderungen noch nicht vollständig überblickt bzw. sich der Möglichkeiten zur Realisierung der Anforderungen nicht bewusst ist und deshalb die Anforderungen nicht formulieren kann.
- Die Entwicklung wird hauptsächlich durch den Code betrieben. Im Zentrum des Interesses stehen lauffähige Systemteile.

Es ist mit diesem Modell möglich, Teile des Systems bereits vor Fertigstellung des gesamten Systems beim Kunden einzuführen (z.B. ein oder mehrere Subsysteme). Mit diesen Teilen kann der Kunde eine eingeschränkte Anzahl an Anforderungen realisieren. Die Zeitspanne zwischen Auftragsvergabe und Einsatz von zumindest Systemteilen wird somit geringer.

Weiters kann auf Anforderungsänderungen besser und schneller reagiert werden. Im besten Fall ist die zu ändernde Anforderung noch nicht implementiert und sie verursacht keinen zusätzlichen Aufwand. Die Entwicklung kann bei jedem inkrementellen Schritt gesteuert werden, da das Produkt in vielen kleinen überschaubaren Schritten erzeugt wird.

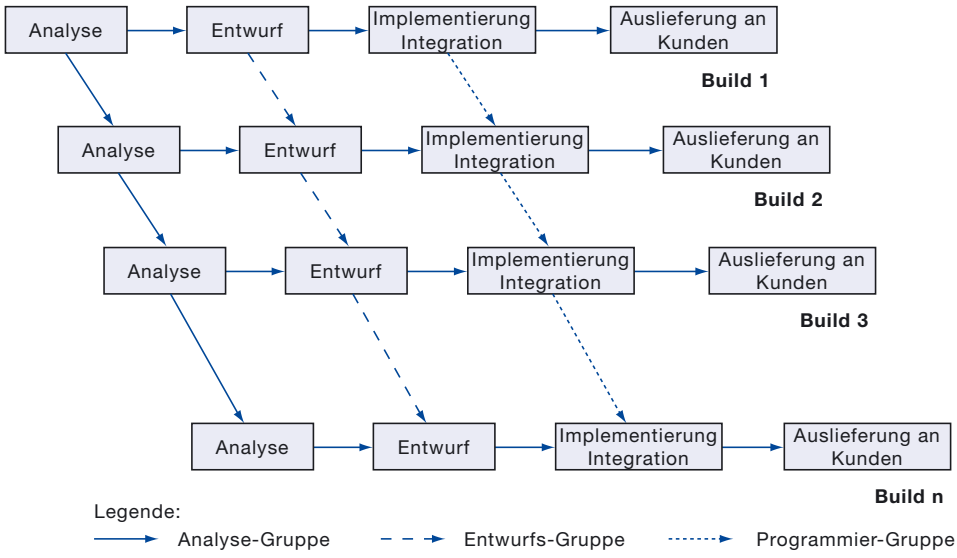

Abbildung 3.4: Inkrementelles Modell (aus [Scha99] S. 76)

Abbildung 3.4 zeigt eine Darstellung dieses Modells. Das System wird in mehreren „Builds“ ausgeliefert. Alle Entwickler erweitern parallel ihre Modelle (Analyse, Entwurf, Programm).

Ein wesentlicher Nachteil dieses ursprünglichen Modells ist das Risiko, dass in einem fortgeschrittenen Stadium der Produktentwicklung die Architektur für die Realisierung der verbleibenden Anforderungen nicht geeignet ist. In einem solchen Fall muss die Architektur unter hohem Aufwand nachträglich geändert werden. Um diesen Fall zu vermeiden, kann man das inkrementelle Modell so weit einschränken, dass vor dem Start des Entwurfs zumindest alle entscheidenden Anforderungen bekannt sein müssen, welche die Architektur des Systems beeinflussen können. In manchen Fällen kann diese Einschränkung überhaupt die Analyse aller Anforderungen vor dem Entwurf bewirken.

3.2 Mikroprozesse – Arbeitsschritte und Aktivitäten

Steht das Vorgehen für ein Projekt fest, so muss der Projektleiter für alle Mitarbeiter die konkreten Tätigkeiten im Rahmen der Arbeitsschritte vorgeben. Diese Tätigkeiten werden nicht für jedes Projekt neu bestimmt, sondern sind meist durch den Softwareentwicklungsprozess eines Unternehmens standardisiert. Dennoch kann es ein Projekt erfordern (aufgrund seiner Größe oder einer hohen Komplexität der Anforderungen), dass zusätzliche Tätigkeiten definiert und durchgeführt werden. Für kleinere oder einfachere Projekte als sonst üblich ist es auch denkbar, dass Tätigkeiten weggelassen werden.

Der folgende Abschnitt beschreibt die Arbeitsschritte Analyse, Entwurf, Implementierung, Test, Inbetriebnahme und Wartung sowie die begleitenden Arbeitsschritte Projektmanagement und Qualitätsmanagement. Der darauf folgende Abschnitt erläutert kurz Aktivitäten.

3.2.1 Arbeitsschritte

Ein Arbeitsschritt ist wesentlich durch seine bestimmte Sicht auf den Gegenstand des Projekts gekennzeichnet und erzeugt dadurch ein bestimmtes Modell dieses Projekts, welches sich in Form und Abstraktionsgrad von den Modellen anderer Arbeitsschritte unterscheidet. Jede dieser Sichten bzw. jedes dieser Modelle verfolgt einen genau festgelegten Zweck und ist für eine spezielle Zielgruppe bestimmt. Aufgrund der unterschiedlichen Zielsetzung eines jeden Arbeitsschritts unterscheiden sich die Tätigkeiten und die beteiligten Rollen wesentlich voneinander.

So betrachtet zum Beispiel der Arbeitsschritt Analyse ein Projekt aus der Sicht der Anforderungen der zukünftigen Anwender. Erst im Arbeitsschritt Entwurf wird das System aus technischer Perspektive betrachtet. Die technische Sicht hätte im Arbeitsschritt Analyse wenig Sinn, da nur wenige Anwender genug technisches Verständnis besitzen, um diese Darstellung verstehen zu können. Für die dem Entwurf folgende Implementierung durch

Programmierer ist die technische Sicht aber notwendig, da die direkte Umsetzung der Anforderungen nicht möglich ist (das wäre, als wolle jemand ein Haus ohne Bauplan bauen).

Die Arbeitsschritte Analyse, Entwurf, Implementierung, Test, Inbetriebnahme und Wartung sowie die Bereiche Projektmanagement und Qualitätsmanagement werden im Folgenden kurz beschrieben.

Arbeitsschritt Analyse: Im Arbeitsschritt Analyse werden in Gesprächen mit dem Kunden die Anforderungen an das System ermittelt und diese in einem objektorientierten Systemmodell beschrieben. Der Arbeitsschritt Analyse dient zur Vorbereitung aller nachfolgenden Arbeitsschritte und schließt mit einer groben Projektplanung für die Realisierung ab.

Arbeitsschritt Entwurf: Während des Entwurfs werden alle technischen Vorbereitungen und Planungen getroffen, welche für eine fehlerfreie und rasche Implementierung des Systems notwendig sind. Parallel zum Entwurf wird auch der Testplan erstellt, welcher einerseits die Erfüllung der Anforderungen und andererseits die korrekte Funktionsweise der Implementierung überprüfen soll.

Arbeitsschritt Implementierung: Die Implementierung dient zur Umsetzung des Entwurfs in einer konkreten Programmiersprache und der Erstellung eines lauffähigen und lieferbaren Systems. Die Implementierung gliedert sich in die Erstellung einzelner Module und deren abschließende Integration zum gesamten System.

Arbeitsschritt Test: Zum Abschluss werden im Arbeitsschritt Test die angestrebte Korrektheit des Systems in Bezug auf Anforderungen und Technik sichergestellt und gefundene Fehler vor der Auslieferung an den Kunden beseitigt.

Arbeitsschritt Inbetriebnahme und Wartung: Das in Betrieb befindliche Produkt ist ständigen Änderungen unterworfen. Zum einen müssen Fehler, die zu diesem Zeitpunkt noch auftreten, ausgebessert werden, und zum anderen müssen für Änderungen der Anforderungen alle Arbeitsschritte nochmals im kleinen Rahmen durchgeführt werden.

Projektmanagement: Das Projektmanagement garantiert den korrekten Ablauf der Erstellung einer Software. Wesentliche Aufgaben im Rahmen des Projektmanagements sind die Planung, die Kontrolle, die Organisation und Koordination von allen Tätigkeiten im Rahmen der Arbeitsschritte, welche zur Erreichung der vorgegebenen Ziele notwendig sind. Eine der wichtigsten Aufgaben des Projektmanagements ist die Teambildung zu Beginn eines Projekts.

Das *Teammanagement* betrifft die konkrete Zuteilung und Koordination von Arbeit für die Projektmitarbeiter. Diese kann nur unter Berücksichtigung von Fähigkeiten und Verfügbarkeit zum jeweils aktuellen Zeitpunkt geschehen. Im Teammanagement werden weiters auftretende soziale Probleme mit Einfluss auf das Projekt geregelt. Nicht zuletzt ist das Teammanagement wesentlich für die Motivation der Mitarbeiter verantwortlich.

Qualitätsmanagement: Das Qualitätsmanagement soll einerseits während eines Projekts die Einhaltung von Qualitätskriterien steuern und überprüfen und andererseits zur Verbesserung des Softwareentwicklungsvorgangs innerhalb eines Unternehmens beitragen.

3.2.2 Aktivitäten

Aktivitäten sind jene Arbeitseinheiten, deren Durchführung stets ein konkretes Produkt oder die Erbringung einer einzelnen Leistung zum Ziel hat. Während an einem Arbeitsschritt eines kompletten Softwareprojekts mehrere oder gar viele Rollen beteiligt sind, kann eine Aktivität durch eine Rolle alleine durchgeführt werden. Aktivitäten sind daher auch Gegenstand von konkreten Arbeitsaufträgen seitens der Gruppen- oder Projektleitung. *Abbildung 3.5* zeigt alle Aktivitäten, welche im Rahmen des Arbeitsschritts Entwurf durchgeführt werden müssen.

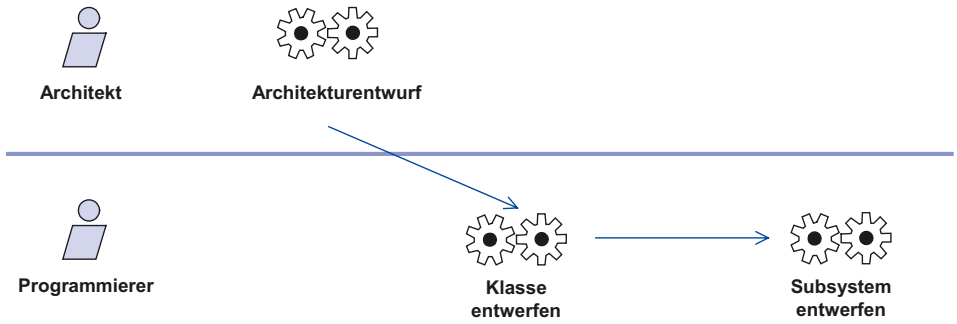

Abbildung 3.5: Aktivitäten des Arbeitsschritts Entwurf

Tabelle 3.1 zeigt alle Einheiten der Projektarbeit, die diese strukturieren, im Überblick. Dabei werden deren durchschnittliche Dauer und die Anzahl der normalerweise daran beteiligten Personen für ein normal großes Softwareprojekt mit einer Dauer von ca. einem halben Jahr angegeben.

Element	ungefähre Dauer	Größe der Arbeitsgruppe
Vorgehensmodell	Projektdauer	7
Phase	Wochen bis Monate	4-7
Arbeitsschritt	eine oder mehrere Wochen	2-5
Aktivität	Tage	1-2

Tabelle 3.1: Strukturelemente eines Software-Prozesses

3.3 Der Unified Process

Der Unified Process bildet die Grundlage für den in diesem Buch vorgestellten Softwareentwicklungsprozess. Nach einem kurzen geschichtlichen Überblick wird in diesem Abschnitt auf seine wichtigsten Konzepte einführend eingegangen.

3.3.1 Geschichte des Unified Process

Der „Unified Software Development Process“ ([Jaco99]), wie er im gesamten Wortlaut heißt, ist keine Neuerfindung, sondern vielmehr die Weiter- bzw. Zusammenführung von in der Industrie durchaus etablierten Ideen mehrerer Personen und Unternehmen im Softwarebereich.

Der Unified Process wurde maßgeblich von den drei Spezialisten James Rumbaugh, Grady Booch und Ivar Jacobson (genannt die drei Amigos) im Rahmen ihrer gemeinsamen Tätigkeit bei Rational entwickelt. Alle drei hatten zuvor schon eigene objektorientierte Methoden bzw. Prozesse entwickelt (OMT, OOAD und OOSE). Rumbaugh und Jacobson verkauften ihre Methoden zusammen mit darauf abgestimmten Werkzeugen sehr erfolgreich an Software-Unternehmen.

Grady Booch arbeitet seit Beginn der 80er Jahre bei Rational und betreute zahlreiche Großprojekte der Softwareentwicklung, aus denen er sein umfangreiches Wissen über Software Engineering bezog. Ende der 80er Jahre veröffentlichte er bereits zwei Bücher zum Thema Software-Erstellung mit Ada. OOAD (Object Oriented Analysis and Design) wurde im gleichnamigen Buch Anfang der 90er Jahre vorgestellt.

OMT (Object Modeling Technique) von **James Rumbaugh** wurde erstmals 1991 vorgestellt [Rumb91]. 1994 wurde von OMT eine weiterentwickelte Version veröffentlicht. James Rumbaugh wechselte 1994 zu Rational, wo er zusammen mit Grady Booch an der Vereinheitlichung ihrer beiden Methoden zu arbeiten begann. Als erstes Ergebnis wurde im Oktober 1995 die Version 0.8 von UML veröffentlicht (siehe auch [Rumb95b]).

Ivar Jacobson hatte schon während seiner Zeit bei Ericsson (einem der führenden Unternehmen der Telekommunikationsindustrie) zahlreiche Erfahrungen in der Softwareentwicklung gesammelt. In diesem Bereich gab es bereits in den 70er Jahren einen Standard namens SDL (Specification and Description Language), welches Systeme in Blöcke aufteilte, welche untereinander mittels Nachrichten kommunizierten. Zur Darstellung solcher Systeme wurden Diagramme verwendet, welche den heute gebräuchlichen UML-Diagrammen ähnlich waren.

1987 gründete Jacobson seine eigene Firma Objectory AB. Im Jahre 1987 veröffentlichte er den Artikel „*Object-Oriented Development in an Industrial Environment*“ ([Jaco87]), in dem der Begriff des „Use Case“ erstmals erläutert wird. Er entwickelte das weltweit erfolgreiche Produkt Objectory – ein Werkzeug zur Erstellung von Software aufbauend auf der Beschreibung von Anwendungsfällen und einem Analysemodell. Während der Jahre 1987 bis 1995 wurde Objectory in mehreren Versionen kontinuierlich weiterentwickelt.

Jacobson veröffentlichte 1992 ein Buch, welches die von Objectory unterstützte Methode beschrieb ([Jaco96]). Das Konzept der Anwendungsfälle wurde im Laufe der Jahre von einer größeren Menge von Software-Entwicklern akzeptiert und übernommen (siehe [Jaco95a]).

Bei Rational wurden ab 1981 intern zahlreiche Arbeiten zur Entwicklung eines Softwareentwicklungsprozesses verfasst und auch umgesetzt. Von Mitarbeitern Rationals stammen wichtige Beiträge zu Themen wie iterative und inkrementelle sowie architekturzentrierte Softwareentwicklung.

1995 wurde Objectory AB von Rational aufgekauft und mit der Vereinheitlichung und Weiterentwicklung des Objectory-Prozesses und des Ansatzes von Rational begonnen. Die Grundlage der Entwicklung bildete die letzte Version des Objectory-Prozesses (Version 3.8). Von Seiten Rationals wurden vor allem die iterative Entwicklung und das Phasenmodell beigesteuert. Während der Jahre 1996 und 1997 wurde so der Rational-Objectory-Prozess 4.1 beschrieben.

Erstes Ergebnis der Bemühungen um die Vereinheitlichung ist die Veröffentlichung der einheitlichen Notation UML 1997. In weiterer Folge wurden in die Weiterentwicklung des Prozesses weitere Unternehmen eingebunden, welche ihr Wissen in verschiedenen Bereichen beisteuerten. 1999 wurde schließlich der Unified Process ([Jaco99]) veröffentlicht.

Abbildung 3.6 zeigt die Entwicklung des Rational Unified Process als eine spezielle Implementierung des Unified Process.

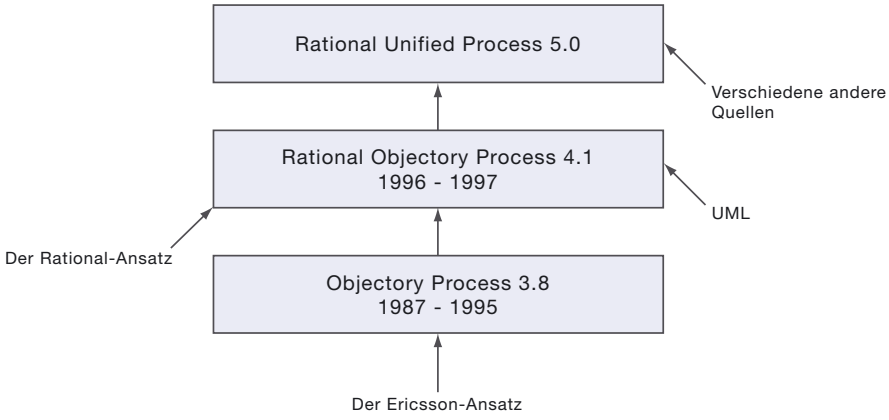

Abbildung 3.6: Entwicklung des Unified Process (aus [Jaco99] S.21)

3.3.2 Eigenschaften des Unified Process

Der Unified Process wird durch drei wesentliche Prinzipien gestaltet:

1. Er wird durch Anwendungsfälle gesteuert: Bevor ein System entworfen und implementiert werden kann, müssen die Anforderungen an das System eindeutig beschrieben werden. Zu diesem Zweck muss die Gruppe der zukünftigen Benutzer eingeschränkt werden, um deren Anforderungen untersuchen zu können. Bei den Benutzern kann es sich um Personen, aber auch um andere Systeme handeln, welche über Schnittstellen mit dem gegenwärtigen System zusammenarbeiten.

   Der gesamte Unified Process wird ausgehend von der Beschreibung der Anforderungen mit Hilfe von Anwendungsfällen gesteuert. Alle Modelle des Entwurfs müssen in Hinblick auf die Umsetzung der Anwendungsfälle erstellt werden. Bei der Implementierung wird immer auf die korrekte Erfüllung der Anwendungsfälle geachtet. Bei der Erstellung des Testplans wird unter anderem die Überprüfung der korrekten Umsetzung der Anwendungsfälle berücksichtigt. Die Anwendungsfälle setzen somit alle Arbeitsschritte der Softwareentwicklung in Verbindung.

2. Der Unified Process ist ein iterativer und inkrementeller Prozess: Aufgrund der Komplexität von vielen Softwareprojekten können die Beschreibung und die Erstellung des Systems nicht in einer einmaligen Abfolge der Arbeitsschritte erfolgen, in der die Produkte der Softwareentwicklung sequenziell erstellt werden. Nach anfänglichen einmaligen Planungsschritten muss jeder Arbeitsschritt in so genannten Iterationen mehrmals durchlaufen werden. Während der wiederholten Ausführung jedes Arbeitsschritts werden der Umfang und die Qualität der Produkte schrittweise verbessert. Im Zuge einer Iteration werden so viele Produkte wie möglich und notwendig parallel weiterentwickelt. Das gesamte Projekt wächst somit inkrementell. Nach genügend Iterationen, wenn also alle gewünschten Produkte in der erforderlichen Qualität vorhanden sind, wird das System ausgeliefert. *Abbildung 3.7* stellt dieses iterative Vorgehen dar.

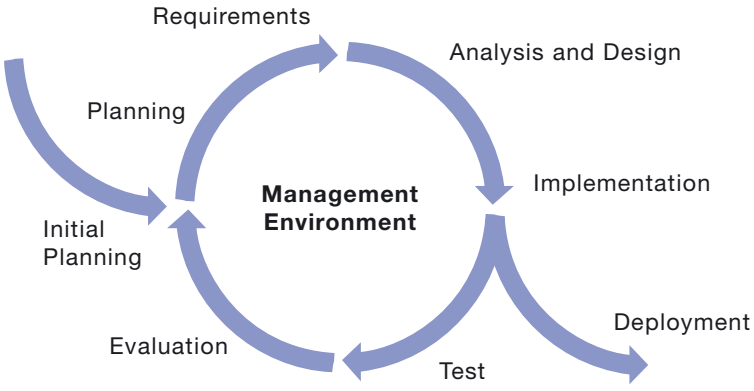

Abbildung 3.7: Ein iterativer und inkrementeller Prozess (nach [Kruc98])

Dieses Vorgehen bringt wesentliche Vorteile im Entwicklungsfortschritt während eines Softwareprojekts mit sich. Das Wasserfallmodell beginnt zwar mit einem linearen Anstieg des Fortschritts während der Analyse, des Entwurfs und der Implementierung, jedoch erleidet ein Projekt mit diesem Ansatz wahrscheinlich spätestens bei der erstmaligen Integration einen wesentlichen Rückfall. Spätestens zu diesem Zeitpunkt treten Mängel der Analyse oder des Entwurfs hervor. Das restliche Projekt ist meist durch ein stetiges Auf und Ab (bedingt durch Änderungen und neu auftretende Mängel) gekennzeichnet.

Bei einer iterativen und inkrementellen Entwicklung werden diese späten Rückschläge vermieden, da die Entwicklung fast aller Produkte, wenn auch nur im geringen Umfang, bereits zu einem frühen Zeitpunkt gestartet wird. Durch das inkrementelle Vorgehen sind Rückschritte, die es dennoch gibt, aber nur von geringem Umfang, da die Änderungen bzw. Weiterentwicklungen von einer Iteration zur nächsten ebenfalls nur von geringem Ausmaß sind. Durch den Wegfall solcher Rückschläge und der damit verbundenen Zeiteinbußen kann ein Projekt mit diesem Vorgehen früher abgeschlossen werden. *Abbildung 3.8* zeigt eine Gegenüberstellung des Projektfortschritts im Laufe der Zeit. Die Entwickler können überdies zielorientierter arbeiten, da die Ziele einer jeden Iteration näher sind als das Projektende. Weiters wird dem Umstand Rechnung getragen, dass die Anforderungen zu Beginn eines Projekts meist nicht vollständig beschrieben werden können oder zumindest Änderungen zu einem späteren Zeitpunkt sehr wahrscheinlich sind.

3. Der Unified Process ist architekturzentriert: Die Bedeutung der Architektur unterscheidet sich im Software Engineering kaum von der Wichtigkeit der Architektur im Bauwesen. Die Architektur bildet die Grundlage für das gesamte System. Ohne passende Architektur kann kein System erfolgreich gebaut werden. Es werden die Bedingungen der Zielumgebung (z.B. Betriebssystem), Komponenten, die zur Erstellung des Systems verwendet werden können, und auch nichtfunktionale Anforderungen bei der Beschreibung der Architektur berücksichtigt.

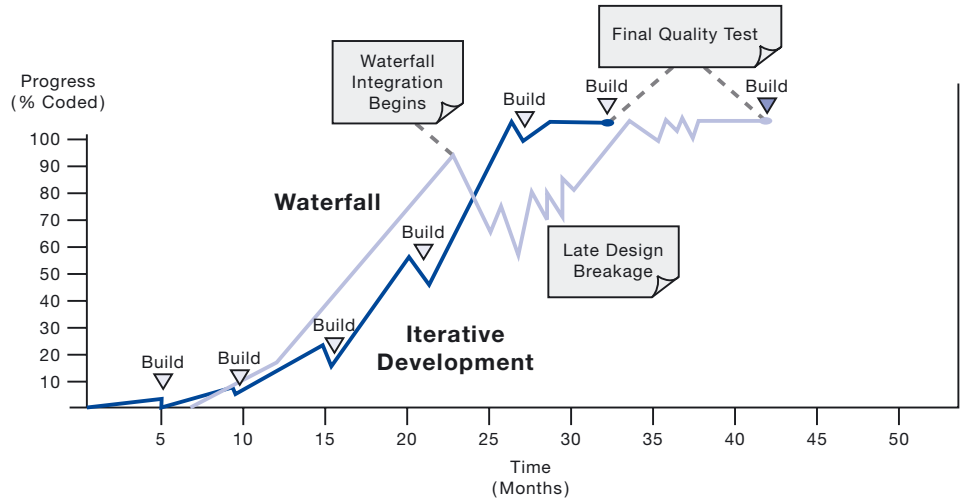

Abbildung 3.8: Iteratives Modell und Wasserfallmodell im Vergleich (aus [Jaco99] S. 93)

Die Architektur gibt dem System die grundlegende Form, welche der Umsetzung der Anwendungsfälle dienen soll. Zur Bestimmung der Architektur reicht normalerweise die Berücksichtigung der wichtigsten Anwendungsfälle aus. Im Zuge der Architekturbeschreibung werden zuerst jene Teile der Architektur beschrieben, welche unabhängig von den Anwendungsfällen sind (z.B. Zielplattform), bevor anschließend für die Architektur die Umsetzung der wichtigsten Anwendungsfälle in Subsystemen, Klassen und Komponenten entworfen wird.

Die Beschreibung der Architektur beginnt bereits sehr früh parallel zur Beschreibung der Anwendungsfälle. So können eventuelle Probleme beim Entwurf der Architektur früh untersucht und gelöst werden. Im Zuge der Detaillierung der Anwendungsfälle im Rahmen der inkrementellen Entwicklung wird auch die Architektur Schritt für Schritt ausgereifter.

3.3.3 Die Elemente des Unified Process

Die statische Struktur des UP beschreibt, wer wann und wie etwas tut, und benutzt dazu vier Modellierungselemente:

- Rollen (wer),
- Aktivitäten (wie),
- Artefakte (was),
- Vorgehen (wann).

Rollen

Das zentrale Konzept im Prozess ist jenes der Rollen. Eine Rolle definiert das Verhalten und die Verantwortlichkeiten eines Individuums oder einer Gruppe von Individuen, die zusammen als ein Team arbeiten. Dieses Verhalten wird in Aktivitäten beschrieben und jede Rolle ist mit einer gewissen Menge an Aktivitäten assoziiert. Die Verantwortlichkeiten wiederum werden durch Artefakte ausgedrückt, mit denen eine Rolle in Verbindung gebracht wird, die sie kontrolliert, erstellt oder modifiziert.

Eine Person kann mehrere Rollen übernehmen und ist damit auch berechtigt, ganze Sammlungen an Artefakten zu erstellen und zu verändern. Beispiele für Rollen wären ein Systemanalyst, ein Designer oder ein Tester.

Aktivitäten

Den Rollen sind Aktivitäten zugeordnet, welche die Arbeit, die sie zu verrichten haben, definieren. Eine Aktivität ist eine Einheit von Arbeitsschritten, die ein Individuum durchführen kann. Jede Aktivität muss ein konkretes Ziel und einen Zweck aufweisen. Mögliche Aktivitäten wären zum Beispiel das Erstellen eines Artefaktes oder das Verändern einer Klasse. Die Granularität einer Aktivität reicht im Allgemeinen von ein paar Stunden bis hin zu ein paar Tagen, und sollte im Normalfall nur mit einer kleinen Anzahl von Artefakten in Verbindung gebracht werden. Aktivitäten können mehrmals durchgeführt

werden, vor allem in unterschiedlichen Iterationen, wenn das System laufend erweitert wird. Mehrmalige Aktivitäten werden zwar von derselben Rolle ausgeführt, was aber nicht notwendigerweise bedeutet, dass es sich jedes Mal um dieselbe Person handelt. Beispiele für Aktivitäten sind die Planung einer Iteration, das Erstellen eines Anwendungsfalls oder das Durchführen eines Tests.

Artefakte

Die zuvor beschriebenen Aktivitäten besitzen Eingangs- und Ausgangsartefakte. Ein Artefakt ist ein Stück Information, das produziert, verändert und vom Prozess verwendet wird. Artefakte stellen sozusagen Neben- oder Teilprodukte des Prozesses dar, während sich der Prozess dem eigentlichen Endprodukt nähert. Artefakte können unterschiedliche Formen annehmen und sind nicht nur auf reine Textdokumente beschränkt. Beispiele sind Modelle, Modellelemente, Dokumente oder Teile des Quellcodes. Artefakte können wiederum aus anderen Artefakten bestehen und sind Gegenstand einer Versionskontrolle.

Da der UP ein iterativer Prozess ist, werden die unterschiedlichen Artefakte in allen Phasen des Prozesses benötigt und verändert, sie entwickeln sich im Laufe des Prozesses weiter. Einige Artefakte erreichen ihren Endzustand allerdings früher als andere.

Vorgehen

Eine reine Aufzählung von Rollen, Aktivitäten und Artefakten ergibt zusammen noch keinen Prozess. Um diese Lücke zu schließen, gibt es ein Vorgehensmodell, welche die einzelnen Aktivitäten aneinander fügen und damit den gesamten Prozess strukturiert.

3.3.4 Vorgehen im Unified Process

Im Unified Process werden fünf grundlegende Arbeitsschritte beschrieben, welche zur Erstellung von Software notwendig sind: Anforderungen, Analyse, Entwurf, Implementierung und Test. Die Ziele, Tätigkeiten und Produkte jedes Arbeitsschritts werden in den folgenden Kapiteln ausführlich beschrieben.

Jeder dieser Arbeitsschritte wird in einer Iteration einmal durchgeführt und führt zur Erstellung eines bestimmten Produkts, welches als Ausgangspunkt für Verbesserungen oder als Grundlage zur Erstellung weiterer Produkte in folgenden Iterationen dient. Die Abfolge der Iterationen ist in Phasen gegliedert, wobei in jeder Phase eine oder mehrere Iterationen durchlaufen werden können. Die Beginnphase dient der Suche nach ersten zentralen Anforderungen, damit das Projekt gestartet werden kann. In der Ausarbeitungsphase werden die Anforderungen größtenteils fertig gestellt. Darauf aufbauend wird eine Analyse erstellt und ein Entwurf begonnen. Parallel dazu erfolgen der Entwurf und die Implementierung des Kerns der Architektur. Während der Konstruktionsphase wird das System fertig entworfen, implementiert und getestet. In der Umsetzungsphase werden alle abschließenden Arbeiten durchgeführt, um das Produkt beim Kunden installieren und in Betrieb nehmen zu können.

Das Ende einer Iteration wird in einem formellen Review festgestellt. In diesem werden alle Produkte auf ihre Reife für den Start einer neuen Iteration geprüft. Einzelne Produkte sollten keine maßgeblichen Mängel aufweisen, da eine sinnvolle Erweiterung bzw. Verbesserung dieser Produkte aufgrund einer fehlenden Basis nicht möglich ist und der Sinn einer Iteration dadurch abhanden kommt. Produkte sollten aber auch nicht zu lange verbessert werden, bis der Start einer neuen Iteration durchgeführt wird, da dies andere auf diesem Produkt aufbauende und auch parallel dazu stattfindende Tätigkeiten unnötig verzögern kann. Der Sinn der folgenden Iteration besteht schließlich ohnehin in der weiteren Verbesserung des Produkts. In diesen Reviews wird vor allem die technische Qualität der Produkte behandelt und das weitere interne Vorgehen beschlossen.

Ein Status Review findet am Ende einer jeden Phase statt. Es wird festgestellt, ob das Projekt die vordefinierten Pläne und Standards erfüllt. Alle Abweichungen und die dazugehörigen Risikoeinschätzungen werden festgehalten und Maßnahmen zu deren Beseitigung werden beschlossen. Sobald alle Abweichungen ausreichend verbessert und alle das Projekt gefährdenden Risiken durch entsprechende Maßnahmen abgesichert worden sind, kann das Projekt in die nächste Phase übergehen.

Weitere Ausführungen zu Reviews, deren Aufbau, Zweck und den daran beteiligten Personen werden im *Kapitel 5* beschrieben.

Die genaue Anzahl der Iterationen kann von Projekt zu Projekt in Abhängigkeit von dessen Größe unterschiedlich sein. In [Kruc98] werden als Richtwerte folgende Werte für die Zahl an Iterationen pro Phase angegeben:

Komplexität des Projekts	Summe Iterationen	Iterationen Beginn	Iterationen Ausarbeitung	Iterationen Konstruktion	Iterationen Umsetzung
Niedrig	3	0	1	1	1
Normal	6	1	2	2	1
Hoch	9	1	3	3	2

Tabelle 3.2: Anzahl von Iterationen abhängig von der Projektkomplexität

Abbildung 3.9 zeigt den Zusammenhang zwischen Arbeitsschritten, Phasen und Iterationen in einem Projekt. Die grau hinterlegten Flächen zeigen die Verteilung des Aufwands eines jeden Arbeitsschritts während aller Phasen. Die gesamte Fläche stellt 100 Prozent des Aufwands pro Arbeitsschritt dar. Bei den einzelnen Arbeitsschritten lässt sich dadurch ablesen, in welcher Phase wie viel Aufwand für diesen Arbeitsschritt notwendig ist. So wird zum Beispiel mit der Anforderungsbeschreibung in der Konzeptionsphase angefangen, in der Entwurfsphase wird der Großteil beschrieben und zu Beginn der Konstruktion werden noch letzte Ergänzungen bzw. Änderungen hinzugefügt.

Alle Flächen gemeinsam ergeben 100 Prozent des Projektaufwands. Dadurch ist erkennbar, wie viel Aufwand ein Arbeitsschritt vom Gesamtaufwand verursacht.

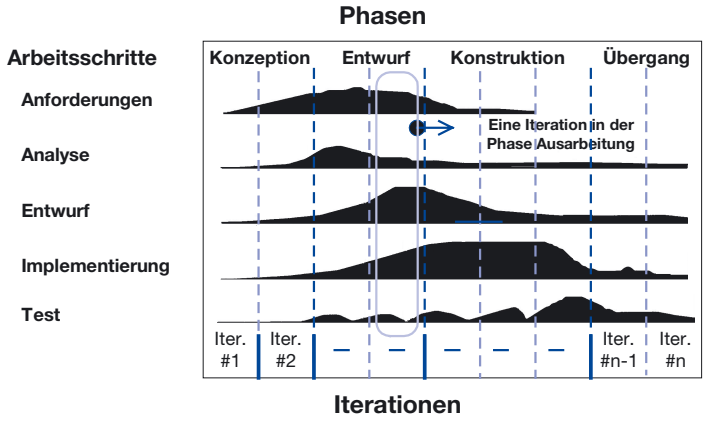

Abbildung 3.9: Phasen, Iterationen und Arbeitsschritte im Unified Process (aus [Jaco99] S. 104.)

Konzeptionsphase

Der Schwerpunkt der Arbeit in der ersten Phase liegt in der Einrichtung der Projektumgebung und der Festlegung aller Projektbedingungen. Dazu gehört einerseits die Initiierung der fünf Arbeitsschritte Anforderungen, Analyse, Entwurf, Implementierung und Test sowie der begleitenden Aktivitäten Projektmanagement und Qualitätsmanagement. Die Aktivität Projektinfrastruktur wird zum großen Teil in der Konzeptionsphase erledigt. Andererseits ist es äußerst wichtig, die zentralen Risiken des Projekts zu identifizieren, um entsprechende Maßnahmen vornehmen zu können.

Nachdem die Geschäftsleitung das Projekt vertraglich fixiert hat, beginnt das Projektteam (das zu diesem Zeitpunkt aus nur wenigen Personen bestehen kann, z.B. dem Projektleiter, einem Analytiker und einem Architekten), die Rahmenbedingungen für das Projekt festzulegen und das Projekt zu starten.

In einem Kick-off-Meeting werden alle zu diesem Zeitpunkt beteiligten Projektmitarbeiter über das Projekt und dessen Ziele informiert. Nach einem solchen Treffen sollten alle Projektmitarbeiter über alle für ihre Arbeit notwendigen Informationen verfügen.

Ausgehend von sehr wenigen Informationen (oft nur eine einseitige Vision) muss der Projektleiter zu Beginn dieser Phase zwei wichtige Tätigkeiten ausführen: Ein erster Projektplan muss erstellt werden und es müssen erste Kriterien für den Abschluss dieser Phase gefunden werden. Der Projektplan wird mit Zunahme der verfügbaren und konsolidierten Informationen ständig überarbeitet, um mit Abschluss der Phase eine erste gültige Version präsentieren zu können. Für mögliche Abschlusskriterien werden in [Jaco99] vier allgemeingültige Kriterien vorgeschlagen:

- Festlegung des ***Fokus des Systems:*** Dazu gehören die Bestimmung der Systemgrenzen, der Akteure und der generellen Schnittstellen zum System und zu anderen Systemen.
- ***Beseitigung aller Unstimmigkeiten*** in den Anforderungen, welche in dieser Phase beschrieben werden: Die Kernanforderungen des Systems sollten identifiziert und alle

Mehrdeutigkeiten beseitigt worden sein. Auch nichtfunktionale Anforderungen sollten ausreichend festgelegt worden sein.

- ***Entwurf einer Architektur:*** Zusätzlich zu den Anforderungen sollten Überlegungen bezüglich der Architektur nicht vergessen werden. Bei dieser Tätigkeit können bereits sehr früh Risiken und Schwierigkeiten für das Projekt identifiziert werden, die aus den Anforderungen alleine nicht erkennbar wären.
- ***Rechtfertigung des Projekts*** gegenüber dem Kunden: Bis zum Ende der Phase sollten dem Kunden ausreichend fundierte Fakten und Zahlen präsentiert werden können, welche die Entwicklung des Systems für ihn tatsächlich attraktiv erscheinen lassen.

Nebenbei wird die für das Projekt notwendige Infrastruktur aufgebaut. Diese Tätigkeit erfolgt meist durch einen Techniker, der dem eigentlichen Projektteam nicht angehören muss. Bei größeren Projekten ist aufgrund des hohen Ausmaßes an anfallenden Tätigkeiten auch während des Projekts ein eigens für das Projekt abgestellter Techniker denkbar. Diese Tätigkeiten umfassen das Einrichten der Arbeitsplätze, das Anlegen von Verzeichnissen, die Vergabe von Benutzerberechtigungen und vieles mehr. Der Projektleiter koordiniert die restliche Infrastruktur wie Sitzungszimmer, Präsentationshilfsmittel (z.B. Projektor) und jegliche weitere nicht technische Infrastruktur.

Der Analytiker sollte in dieser Phase die wesentlichen Anforderungen für das System finden und beschreiben. Dazu bedarf es zuerst eines tieferen Verständnisses des Kontexts, in welchem das System eingesetzt werden soll. Um dies zu ermöglichen, kann ein Geschäftsmodell erstellt werden (hierfür bietet UML eine bereits vordefinierte Erweiterung an). Nachdem alle notwendigen Begriffe geklärt und die Akteure identifiziert worden sind, kann der Analytiker beginnen, die ersten Anwendungsfälle zu beschreiben. Zu Beginn der Beschreibung sollte mit Hilfe eines Anwendungsfalldiagramms eine Übersicht geschaffen werden und nach einer Festlegung der Wichtigkeit der einzelnen Fälle sollten wenige sehr wichtige Anwendungsfälle genauer beschrieben werden.

Der Architekt erstellt parallel zur Anforderungsanalyse eine erste Analyse und einen ersten Entwurf der Systemarchitektur. Besonderes Gewicht wird hierbei auf die Festlegung von Schnittstellen und die Aufteilung des Systems in Subsysteme gelegt. Um vor allem die Schnittstellen ausreichend beschreiben zu können, muss auch die Zielplattform bereits zu diesem frühen Zeitpunkt genügend spezifiziert werden.

Der Projektleiter ist für die Überprüfung der Ergebnisse dieser Phase verantwortlich. Weiters sollte der Projektleiter basierend auf den zunehmenden Informationen die wesentlichen Risiken für das Projekt identifizieren und geeignete Maßnahmen für deren Vermeidung bereitstellen. Gegen Ende der Phase plant der Projektleiter das Vorgehen für die nächste Phase.

Nach der Konzeptionsphase sind die Artefakte in *Tabelle 3.3* vorhanden.

Produkt	Status	Produkt	Status
Geschäftsmodell	fast fertig	Architektur-beschreibung	fertig
Anwendungsfalldiagramm	fortgeschritten	Risiken- und Maßnahmenliste	entsprechend Informationsstand
Anwendungsfallbeschreibung	begonnen		

Tabelle 3.3: Produkte nach der Konzeptionsphase

Entwurfsphase

In dieser Phase wird der Großteil der Analyse durchgeführt. Zusätzlich wird eine erste Implementierung des Architekturkerns realisiert, um eine Einschätzung der daraus entstehenden Komplexität zu erhalten. Da das Projektteam im Normalfall noch klein ist, werden in dieser Phase viele Iterationen durchgeführt werden, um die Anforderungen so präzise wie möglich zu formulieren.

Der Großteil der Anforderungen wird ausreichend analysiert und dokumentiert. Zu diesem Zweck muss der Analytiker einen intensiven Kundenkontakt pflegen. In mehreren Treffen werden alle Anforderungen erkundet und alle Fragen ausreichend abgeklärt. Es werden den bereits bestehenden Anforderungen weitere Akteure und sowohl funktionale als auch nichtfunktionale Anforderungen hinzugefügt. Bei nichtfunktionalen Anforderungen ist auf deren möglicherweise unmittelbaren Auswirkungen auf die Architektur zu achten.

Alle Anwendungsfälle werden in das Anwendungsfalldiagramm übernommen. Mit zunehmender Größe wird es notwendig, dieses vermehrt zu strukturieren (z.B. durch das Hinzufügen von neuen Subsystemen). Bereits identifizierte Anwendungsfälle werden im Detail beschrieben. Nach dieser Phase sollten alle Anforderungen verstanden worden sein (auch wenn deren vollständige Beschreibung noch nicht abgeschlossen ist). Mit dem Kunden müssen nur mehr unwesentliche Details besprochen werden.

Begleitend dazu wird das Analysemodell erstellt und ausgebaut. Es werden Klassen gefunden, welche alleine oder in Zusammenarbeit mit anderen Klassen die Umsetzung der Anwendungsfälle übernehmen können. Diese Klassen werden in einem Analysemodelldiagramm dargestellt. Anschließend erfolgt eine detaillierte Darstellung aller Klassen.

Ein Programmierer erstellt unter Anleitung des Analytikers bzw. basierend auf den bereits vorhandenen Anforderungen einen Analyseprototypen, der dem Kunden anhand der Anwenderschnittstellen Aussehen und Funktion des zukünftigen Systems näher bringen soll. In Treffen mit dem Kunden können Unklarheiten bezüglich der Gestaltung der Anwenderschnittstelle und auch der Anforderungen geklärt werden, da der Prototyp eine erste Umsetzung der Anforderungen darstellt.

Weiters wird die Architektur fertig entworfen und zumindest teilweise implementiert. Dazu müssen die Schichten und Subsysteme des Systems endgültig festgelegt und ihre Schnittstellen untereinander definiert werden.

Der Gruppenleiter kann für fertig gestellte Anforderungen, für welche auch die Umsetzung im Analysemodell abgeschlossen ist, mit dem Entwurf der Klassen beginnen. Auch hierbei sollte mit Klassen begonnen werden, welche für das System von besonderer Bedeutung sind (korrespondierend zu den Anforderungen, welche in der Konzeptionsphase als erste beschrieben worden sind).

Der Projektleiter erstellt aufgrund der ersten vollständigen Version der Anforderungen (diese sollte spätestens als Ergebnis der vorletzten Iteration dieser Phase veröffentlicht werden, um in der letzten Iteration noch genügend Zeit für Änderungen zu haben) einen Projektplan für den Entwurf und die Implementierung. Weiters überprüft der Projektleiter in Reviews die erstellten Produkte.

Mit zunehmender Zahl der Anwendungsfälle kann der Tester bereits erste Testfälle erzeugen und dokumentieren. Nach der Erstellung des Projektplans ist der Tester auch in der Lage, einen Testplan zu erstellen. Die Implementierung der Architektur kann bereits in dieser Phase getestet werden. Aus diesem Grund sollten die Testfälle für die Architektur als Erstes erstellt werden.

Parallel zu diesen Tätigkeiten kann auch bereits eine erste Rohfassung des Anwenderhandbuchs und der Online-Hilfe erstellt werden, da zu deren Erstellung hauptsächlich die Anwenderschnittstellen (des Prototypen) und die Funktionalität des Systems (aus den Anwendungsfällen) notwendig sind.

Nach der Entwurfsphase sind die Artefakte in *Tabelle 3.4* vorhanden.

Produkt	Status	Produkt	Status
Geschäftsmodell	fertig	Benutzerdokumentation	begonnen
Anwendungsfalldiagramm	fertig	Entwurf der Klassen	begonnen
Anwendungsfallbeschreibung	fast fertig	Analyseprototyp	fertig
Analysemodelldiagramm	fertig	Beschreibung der Komponenten des Analysemodelldiagramms	fast fertig
Architekturbeschreibung	fertig	Testplan	fertig
Implementierung der Architektur	begonnen	Testfälle	fortgeschritten
Projektplan für Realisierung	fertig	Risiken- und Maßnahmenliste	entsprechend Informationsstand verändert

Tabelle 3.4: Produkte nach der Entwurfsphase

Konstruktionsphase

Alle Anforderungen und die daraus resultierende Analyse ergeben zu Beginn dieser Phase zusammen mit der Architektur eine konsistente und realisierbare Gesamtheit. Der Entwurf wird fortgeführt und abgeschlossen. Darauf aufbauend kann mit der Entwicklung und den Tests der entwickelten Komponenten begonnen werden. Diese werden abschließend zu dem zu entwickelnden Produkt zusammengesetzt. Begleitend zum System sollte die notwendige Dokumentation fertig gestellt werden. Änderungen des Systems, welche während der Implementierung notwendig werden, sollten sofort wieder in diese einfließen. Am Ende dieser Phase sind das fertige System und alle an den Kunden zu liefernden Produkte in einem auslieferbaren Zustand.

Die Implementierung der Architektur wird fertig gestellt und entsprechend den auftretenden Änderungen adaptiert.

Nachdem die Anforderung und das Analysemodell weitgehend abgeschlossen sind oder zu Beginn dieser Phase fertig gestellt werden, können die Programmierer mit dem Entwurf und der Implementierung der Klassen beginnen.

Mit wachsender Zahl der implementierten Klassen wird das System integriert und getestet. Dabei werden zuerst Klassentests, anschließend Integrationstests und schließlich Systemtests durchgeführt. Nach jedem Testdurchlauf passen die Tester die Testfälle entsprechend ihrer neuen Erkenntnisse an und übergeben den Programmierern Fehlerberichte, damit alle gefundenen Fehler ausgebessert werden können. Der Projektleiter erhält nach jedem Testdurchlauf einen Bericht über den Fortschritt der Implementierung.

Zusätzlich überprüft der Projektleiter in Reviews den Projektfortschritt selbst. Weicht dieser vom Projektplan ab, so müssen geeignete Maßnahmen zur Einhaltung des Projektplans beschlossen und durchgeführt werden.

Anwenderhandbuch und Online-Hilfe werden fertig gestellt und in das System eingebunden.

Der Architekt bereitet während dieser Phase die Zielumgebung für die Installation und Datenübernahme vor. Er entwickelte einen genauen Plan für das Vorgehen zur Integration des Systems auf der Zielumgebung. Weiters erstellt er Schulungsunterlagen für die Anwenderschulung nach erfolgter Inbetriebnahme. Dabei sollte er auf möglicherweise unterschiedliche Zielgruppen (Anfänger, Fortgeschrittene bzw. Benutzer der einen oder anderen Abteilung) besonders achten und wenn notwendig auch unterschiedliche Schulungen gestalten.

Nach der Konstruktionsphase sollten alle Produkte, welche im Zuge des konkreten Softwareprojekts erstellt werden sollten, vorhanden und getestet worden sein. Alle verbleibenden Fehler und Inkonsistenzen werden vor der Auslieferung an den Kunden in der nächsten Phase noch beseitigt.

Übergangsphase

Das Produkt wird fertig gestellt und nach abschließenden Tests und Qualitätskontrollen beim Kunden auf der Zielplattform integriert. Nach der erfolgten Integration werden die abschließenden Integrationstests durchgeführt. Nach der Schulung der Anwender und

einer formellen Abnahme durch den Kunden kann das Projekt abgeschlossen werden. Der Architekt bleibt für den Kunden Ansprechpartner für auftretende Probleme oder Fehler. Für eine notwendige Weiterentwicklung wird auf Initiative des Kunden durch die Geschäftsleitung ein neues Projekt gestartet.

Die Programmierer korrigieren letzte Fehler in den Klassen. Die Implementierung wird mit der letzten Integration aller Systemteile fertig gestellt. Die letzten Tests sollen die Fehlerfreiheit des Systems überprüfen.

Nach der endgültigen Fertigstellung des Systems, der internen Abnahme durch den Projektleiter und der Freigabe durch die Geschäftsleitung wird das System beim Kunden integriert. Weiters wird die Datenübernahme laut der dafür erstellten Planung durchgeführt. Integrationstest vor Ort sollen das Funktionieren auf der Zielplattform sicherstellen.

Die Anwender werden am System selbst geschult. Weiters werden die verantwortlichen Personen und Kommunikationswege (Telefon, E-Mail usw.) für Probleme während des Betriebs bestimmt.

Nach Abschluss aller Projekttätigkeiten kommt dem Qualitätssicherer die Aufgabe der Projektanalyse zu. Er wertet alle Daten aus, welche über den Projektablauf verfügbar sind. Aufgrund der dadurch gewonnenen Fakten lässt sich der Entwicklungsprozess anpassen oder es werden Maßnahmen für kommende Projekte beschlossen.

Alle Projektdaten werden archiviert.

In dieser Phase entstehen keine neuen Produkte. Alle bisher angefertigten Produkte werden verbessert und abgeschlossen. Alle für den Kunden notwendigen Produkte werden in ein Installationspaket zusammengefasst und an diesen ausgeliefert. Sämtliche Projektdaten werden im Unternehmen selbst archiviert.

3.4 Der Rational Unified Process (RUP)

Der Rational Unified Process ist eine konkrete Implementierung des Unified Process. Der RUP ist, wie der Name schon sagt, von der Firma Rational entworfen worden und wird von dieser auch als Produkt vertrieben. Zur Unterstützung des RUP bietet Rational auch eine Reihe von darauf abgestimmten Werkzeugen an.

Der RUP definiert neun Hauptarbeitsschritte, alle repräsentieren eine logische Einteilung und Gruppierung von Arbeitern und Aktivitäten nach Aufgabenbereich und Disziplinen. Es gibt sechs Ingenieurs Arbeitsschritte (*Engineering Workflows*): Geschäftsmodellierung (*Business Modeling*), Anforderungen (*Requirements*), Analyse und Entwurf (*Analysis and Design*), Implementierung (*Implementation*), Test (*Test*), Auslieferung (*Deployment*). Weiters existieren drei unterstützende Arbeitsschritte: Projektmanagement (*Project Management*), Konfigurations- und Änderungsmanagement (*Configuration and Change Management*), Infrastruktur (*Environment*).

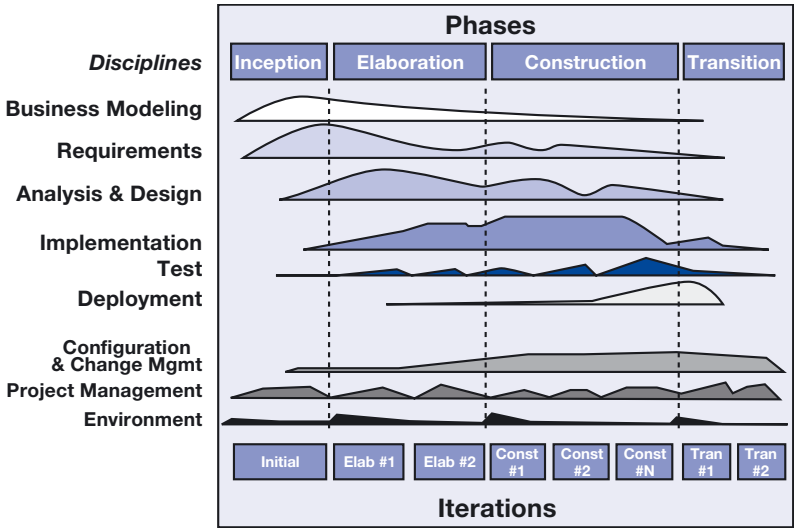

Abbildung 3.10: Phasen und Arbeitsschritte im RUP (aus [Kruc00] S. 23)

Obwohl diese sechs Ingenieurs Arbeitsschritte auf den ersten Blick mit den Phasen des traditionellen Wasserfallmodells übereinstimmen können, unterscheiden sich diese vor allem dadurch, dass die Arbeitsschritte im Laufe des Prozesses iterativ immer wieder durchlaufen werden. Ein kompletter Arbeitsschritt des RUP durchläuft diese neun Hauptarbeitsschritte und wiederholt diesen Durchlauf in unterschiedlichen Phasen des Projekts, wobei die Intensität und Dauer des Durchlaufs verändert wird.

3.4.1 Die Arbeitsschritte im Detail

Arbeitsschritt Projektmanagement

Dieser Arbeitsschritt beschäftigt sich mit dem Ausbalancieren gegensätzlicher Ziele, dem Verwalten von Risiken und damit jene Hürden zu überwinden, die bei der Auslieferung an den Kunden und den Endbenutzer entstehen können. Vor allem für das Management intensiver Softwareprojekte ist dieser Arbeitsschritt die Grundlage für die Planung, Ausführung und Überwachung eines Projekts sowie die Verwaltung der Projektrisiken. Einige wichtige Aspekte des Projektmanagements werden jedoch in diesem Arbeitsschritt nicht abgedeckt:

- Personalmanagement,
- Budgetmanagement,
- Vertragsmanagement (Zulieferer und Kunden).

In diesem Arbeitsschritt findet die Planung der Iteration statt, dazu werden zwei Detailstufen an Plänen unterschieden. Zum einen findet sich der Phasenplan, zum anderen der Iterationsplan. Der Phasenplan existiert nur einmal im Projekt und beschreibt den kom-

pletten Projektzyklus, und kann bei entsprechender Ausarbeitung auch für weitere, ähnliche Projekte herangezogen werden. Darin werden alle wesentlichen Meilensteine berücksichtigt. Der Phasenplan wird bereits frühzeitig in der Anfangsphase erstellt, und wird so oft als nötig überarbeitet.

Der Iterationsplan ist sozusagen die „Feinplanung“ und existiert einmal pro Iteration. In einem Projekt sind im Normalfall zwei Iterationspläne gleichzeitig aktiv: Der momentane Iterationsplan, und der nächste Plan der folgenden Iteration, der ab der Halbzeit der ersten Iteration erstellt wird. Der Iterationsplan wird mit herkömmlichen Methoden der Planung (beispielsweise GANTT- Diagramme) erstellt und definiert die einzelnen Aufgaben und ihre Zuordnung zu den Teams. Der Iterationsplan kann als eine Art „Lupe“ verstanden werden, die sich über den Projektplan fortbewegt und eine detaillierte Vergrößerung zeigt.

Arbeitsschritt Geschäftsmodellierung

Ziel der Geschäftsmodellierung ist das Verstehen der Struktur und Dynamik einer Organisation, in der das System letztendlich zum Einsatz kommen soll. Weitere zentrale Punkte sind: das Verstehen der Probleme der Zielorganisation, die Garantie, dass Kunden und Entwickler dieselben Vorstellungen über die Zielorganisation haben, und die Systemanforderungen so anzupassen, um die Zielorganisation tatsächlich wirkungsvoll zu unterstützen. Dazu wird in diesem Arbeitsschritt die Zielorganisation abgebildet und darauf basierend Prozesse, Rollen und Verantwortlichkeiten in einem Geschäftsmodell dargestellt. Ein Geschäftsprozess-Modell und ein Geschäftsobjekt-Modell werden für dieses Ziel herangezogen.

Arbeitsschritt Anforderungen

Die Ziele dieses Arbeitsschritts sind das Etablieren und Beibehalten einer gemeinsamen Sicht, was das System leisten soll. Damit wird den Systementwicklern ein besseres Verständnis für die Anforderungen geboten, die Grenzen und Limits des Systems werden gezogen und eine Basis für die technische Planung und die Zeit und Kostenschätzung geliefert.

Um diese Zielsetzungen zu erreichen, sieht der Arbeitsschritt die Erstellung einer Vision des Systems vor und definiert, wie diese Vision in ein Anwendungsfallmodell überführt wird. Der Arbeitsschritt definiert ebenfalls die Verwendung von Anforderungsattributen und wie diese genutzt werden können, um den Umfang der Anforderungen zu reduzieren und die Änderungen selbiger zu erleichtern. Neben dem Anwendungsfallmodell selbst werden in diesem Arbeitsschritt die Benutzerschnittstellen definiert und modelliert. Der Arbeitsschritt legt großes Augenmerk auf die Zusammenarbeit von Systementwicklern und den Stakeholdern, um eine einheitliche Sichtweise des System zu erlangen. Die Anforderungen werden dazu in unterschiedliche Kategorien wie funktionale und nichtfunktionale sowie „High-Level“-Anforderungen unterteilt.

Arbeitsschritt Analyse und Entwurf

Der Arbeitsschritt beschäftigt sich mit dem Überleiten der Anforderungen in spezifische Implementierungsrichtlinien (also den Entwurf) für das System. Zudem soll bereits frühzeitig im Projekt eine robuste Architektur für das System festgelegt werden, wobei die Eigenheiten der Implementierungsumgebung berücksichtigt werden müssen. Die Analyse beschäftigt sich dabei mit der Umwandlung der Anforderungen in eine für den Entwickler verständliche Sprache, zumeist Klassen und Subsysteme. Die Analyse verzichtet größten Teils auf die nichtfunktionalen Anforderungen und stellt ein ideales Bild des Systems dar. Der Entwurf adaptiert dann die Resultate der Analyse in Hinblick auf die nichtfunktionalen Anforderungen und die Bedingungen der Entwicklungsumgebung. Eine Optimierung des Systems wird dabei ebenfalls durchgeführt. Als Resultat des Arbeitsschritts entstehen das Analyse- und Entwurfsmodell, eine Beschreibung der Schnittstellen, die Systemarchitektur sowie Datenbankentwürfe. Dieser Arbeitsschritt schließt somit die Lücke zwischen den Anforderungen und der Implementierung.

Arbeitsschritt Implementierung

In der Implementierung entstehen das ausführbare System, die einzelnen Komponenten und Klassen sowie die Tests der Subsysteme, nicht jedoch die Tests des kompletten Systems. Dies wird in einem eigenen Arbeitsschritt durchgeführt, dem Arbeitsschritt Test. Im RUP wird zwischen drei Konzepten in der Implementierung unterschieden:

1. Builds: Ein Build beschreibt eine operationale Version des Systems. Durch die iterative Softwareentwicklung im RUP kommt es natürlich zu zahlreichen Builds. Diese dienen der frühzeitigen Fehlererkennung und als Review-Zeitpunkte. In Normalfällen wird versucht, in regelmäßigen Abständen, täglich oder zumindest einmal pro Woche, ein Build zu veröffentlichen.

2. Integration: Die Integration beschreibt das Zusammenführen von unterschiedlichen Komponenten und Subsystemen zu einem Ganzen. Das kann von der Zusammenführung der Ergebnisse einzelner Teams bis hin zur Integration des kompletten Systems reichen. Im RUP soll die Integration inkrementell stattfinden, das bedeutet, dass kleine Codesegmente geschrieben und getestet werden sollen, die dann zu einem größeren Teil zusammengefügt werden. Die Integration muss zumindest einmal pro Iteration durchgeführt werden.

3. Prototypen: Prototypen werden zur Verminderung der Risiken genutzt. Dazu sind im RUP unterschiedliche Arten von Prototypen definiert, welche für die Struktur oder das Verhalten des Systems herangezogen werden können. Prototypen werden entwickelt, um im Vorfeld gewisse technische oder anforderungsspezifische Probleme auszuloten, und dem Benutzer einen Einblick in die Art des EndProdukts zu gewährleisten. In vielen Projekten wird jedoch der Fehler gemacht, „gute“ Prototypen in das finale System zu integrieren, was oftmals zu Qualitätseinbußen führen kann. Daher wird im RUP eine strikte Trennung zwischen Prototypen und tatsächlichem System vorgenommen.

Arbeitsschritt Test

Der Arbeitsschritt Test im RUP definiert eine Vielzahl an Tests und Testvorgängen sowie das Konzept der Qualitätsverantwortung für die einzelnen Teams. Dem Testen wir daher im RUP ein hoher Stellenwert beigemessen, wobei die Tests im RUP nicht dazu gedacht sind, Qualität zu bestätigen, sondern Qualität zu erreichen. Jedes Team und jeder Entwickler ist im RUP dafür verantwortlich, dass die von ihm erstellten Artefakte nicht nur den Anforderungen sondern auch den erwarteten Qualitätsansprüchen genügen. Das Testmodell im Arbeitsschritt besteht aus den Teilen Testfälle, Testprozesse, Testskripts, Testklassen und Testabläufe.

Arbeitsschritt Konfigurations- und Änderungsmanagement

Wie der Arbeitsschritt Test hat auch dieser Arbeitsschritt einen großen Einfluss auf die Qualitätssicherung. Ziel ist es, die erstellten Artefakte zu verfolgen, aufzubewahren und deren Integrität sicherzustellen. Konfigurationsmanagement beschäftigt sich mit der Identifizierung der Artefakte, deren Abhängigkeiten und Versionsnummern. Zudem teilt es die „Arbeitsräume“ den einzelnen Arbeitern zu, damit sich diese während ihrer Arbeit nicht „auf die Füße steigen“.

Das Änderungsmanagement ist für die Verwaltung der einzelnen Änderungsanfragen durch interne oder externe Stakeholders zuständig. Zudem werden hier auch die Auswirkungen der einzelnen Änderungen erfasst und analysiert. Zu guter Letzt beschäftigt sich die Statusprüfung und Messung, mit dem Extrahieren von Informationen für das Projektmanagement, durch Tools welche die Funktionen des Konfigurations- und Änderungsmanagement anbieten.

Arbeitsschritt Infrastruktur

Das Ziel und die Aufgabe dieses Arbeitsschritts ist die Unterstützung der Entwicklungsorganisation sowohl mit Prozessen als auch mit Tools. Dadurch erhofft man sich die Reduzierung der menschlichen Fehler auf ein Minimum. Diese Unterstützung inkludiert Folgendes:

- Auswahl von Tools und deren Erwerb,
- das Installieren und Anpassen der Tools in Hinblick auf die Organisation,
- Prozesskonfiguration,
- Prozessverbesserung und Weiterentwicklung,
- technische Dienstleistungen wie IT-Infrastruktur, Zugriffsadministration und das Erstellen von Backups.

Arbeitsschritt Auslieferung

Softwareentwickler finden sich des Öfteren zu früh im Glauben, bereits „gewonnen“ zu haben, und vergessen dabei immer wieder, dass nicht das fehlerfreie Kompilieren des Systems das Ende eines Softwareentwicklungsprozesses darstellt, sondern der Wille des Kunden dieses System auch zu benutzen. Das Ziel und der Zweck dieses Arbeitsschritts

ist es, die fertig gestellte Software dem Kunden zu übergeben. Dabei werden die folgenden Kernelemente herausgehoben:

- Beta-Tests,
- Software-Übergabe vorbereiten (Installationspakete),
- Installation und Verteilung der Software,
- Einschulung des Endbenutzers bzw. des Verkaufspersonals,
- Migration existierender Software und Datenbanken.

Die Wege, wie diese Kernelemente durchgeführt werden, unterscheiden sich hinsichtlich der Größe des Projekts und der Geschäftsbedingungen beträchtlich. Es ist daher für jedes Unternehmen und jedes Projekt, welches den RUP verwendet, ein eigener Spezialisierungsprozess nötig, wie dieser Arbeitsschritt letzten Endes umgesetzt wird. Einige grundlegende Artefakte wie Installationspakete, Supportmaterial (Benutzerhandbücher oder Wartungshinweise) oder Testergebnisse sind jedoch in allen Fällen ein Muss.

3.5 Das Microsoft Solution Framework (MSF)

Das MSF konzentriert sich auf die Planung, das Erstellen und die Auslieferungsphasen des Projektlebenszyklus. Dazu stellt das MSF neben Artikeln, Fallstudien und Kursmaterial für die Bereiche Unternehmensarchitektur, Applikationsentwicklung, Komponentendesign und Auslieferung auch detaillierte Pläne und Lösungsansätze zur Verfügung.

Das MSF besteht aus drei Kernmodellen, dem Risikomanagement-, dem Team- und dem Prozessmodell. Das MSF Prozessmodell kombiniert Konzepte des Wasserfall- und Spiralmodells, orientiert sich dabei stark an den Meilensteinen des Wasserfallmodells und dem Feedback aus dem Spiralmodell.

Die Struktur des MSF besitzt vier Phasen, die sich an unterschiedliche Projekttypen anpassen. Jede Phase endet mit einem nach außen sichtbaren Meilenstein (Abbildung 3.11). Sowohl die Benennung der Meilensteine als auch der Phasen sind von Projekt zu Projekt unterschiedlich.

3.5.1 Einsatz des Prozessmodells

Wie schon zuvor erwähnt kann das Prozessmodell für unterschiedliche Projekttypen eingesetzt werden und ändert sich dementsprechend in der Benennung der Phasen und Meilensteine. Als Beispiele für Projekttypen können nicht nur „normale“ Applikationsentwicklungen sondern auch der Infrastrukturaufbau oder Unternehmensgestaltungsprozesse (in Bezug auf deren Softwaresysteme) angeführt werden. In der Applikationsentwicklung und Unternehmensgestaltung würden die vier Phasen „Visionsfindung“, „Planung“, „Entwicklung“ und „Stabilisierung“ zum Einsatz kommen, in Infrastrukturprojekten „Visionsfindung“, „Planung“, „Entwicklung“ und „Auslieferung“. Während einer Unternehmensgestaltung kann das Projekt in der Entwicklungsphase in kleinere Projekte übergehen bzw. diese starten. Diese Projekte benutzen dann wiederum das MSF-Prozessmodell (s. Abbildung 3.12).

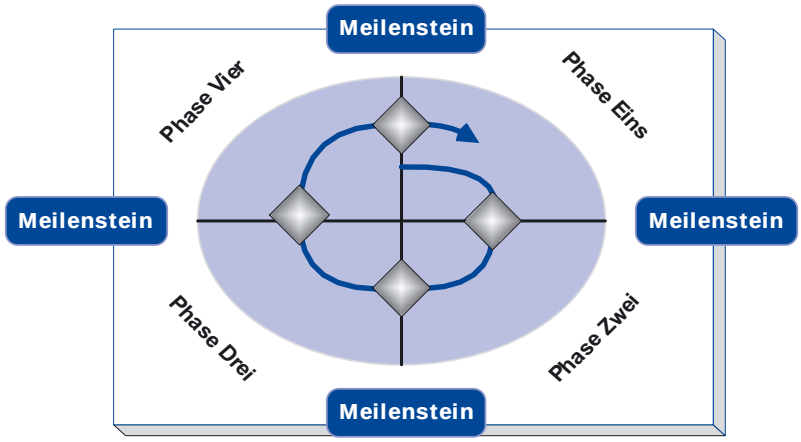

Abbildung 3.11: MSF-Prozessmodell

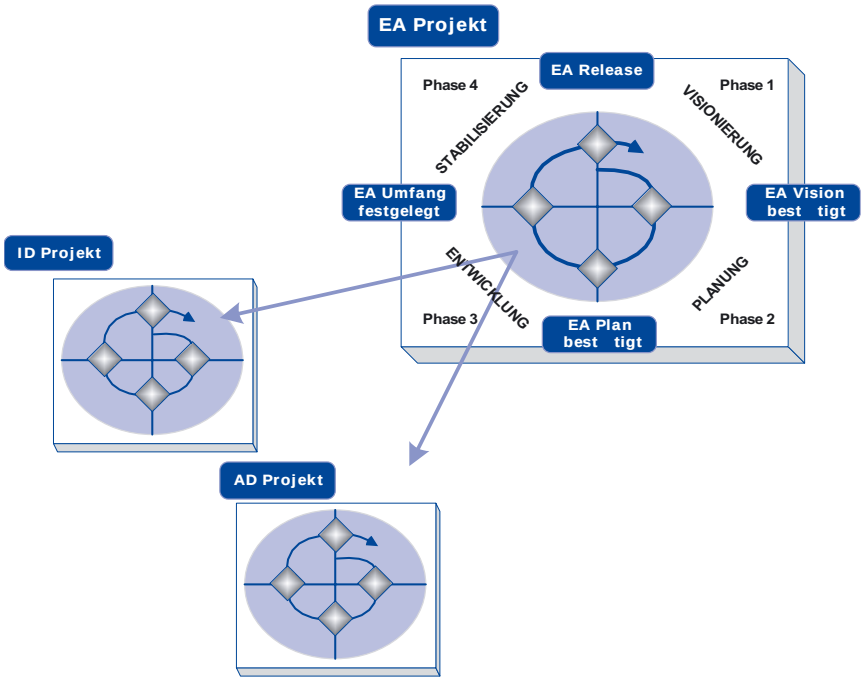

Abbildung 3.12: MSF-Prozess im Einsatz

3.5.2 Einsatz in der Applikationsentwicklung

Die vier Phasen in der Applikationsentwicklung sind zwar vom Namen her gleich wie jene in der Unternehmensgestaltung, doch variieren die Aktivitäten und Aufgaben. *Abbildung 3.13* zeigt die vier Phasen und ihre Meilensteine.

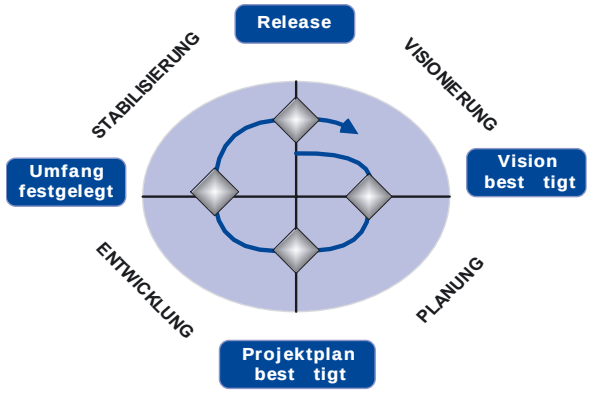

Abbildung 3.13: Applikationsentwicklung

Phase eins: Visionsfindung

Das übergeordnete Ziel der Phase ist eine gemeinsame High-Level-Sicht auf die Ziele des Projekts sowie die Festlegung von Einschränkungen. Das Team und der Kunde analysieren das Geschäftsproblem, die Ziele des Produkts, eine Übersicht über die Problemlösung, Profile der Benutzer und grundlegende Entwurfsziele. All diese Ergebnisse werden in einem gemeinsamen Dokument, dem Vision/Scope-Dokument, gesammelt.

Phase zwei: Planung

Im Zuge der Planungsphase werden vom Team nicht nur ein Hauptprojektplan und ein Zeitplan erstellt, sondern auch eine funktionale Spezifikation des Produkts. Diese beschreibt, was implementiert werden wird, und listet dabei die Produktziele, Anforderungen, Features und Abhängigkeiten der Funktionalitäten auf. Wird diese Phase mit dem entsprechenden Meilenstein abgeschlossen, ist dies der Startschuss zur Implementierung des Produkts.

Phase drei: Entwicklung

Während dieser Phase liegt der Fokus auf der Programmierung und dem Testen des Produkts. Die Phase involviert eine Vielzahl von internen Veröffentlichungen des Produkts, das parallel und segmentiert entwickelt wird. Die Fortschritte werden dabei laufend gemessen und ständig synchronisiert.

Der Testprozess gehört zwar zur Stabilisierungsphase, ist aber bereits Bestandteil der Entwicklung. Da die Rolle des Testers nicht nur auf das Auffinden von Bugs beschränkt ist,

sondern auch auf das Sichern der Qualität, ist der Tester bereits in der Entwicklung gefordert, zu überprüfen, ob das Produkt die Problemstellung qualitativ hochwertig löst. Gegen Ende der Phase beginnt der Tester mit den Abdeckungstests, abzielend auf die Features und den Code des Produkts, und den Benutzertests. Die Entwicklungsphase endet mit dem entsprechenden Meilenstein, wenn alle Features im Programm integriert sind und das Produkt zur Stabilisierung bereit ist. Benutzerhilfen müssen zu diesem Zeitpunkt ebenfalls bereits vorhanden sein, und das Team und die Stakeholder übereinstimmen, dass wirklich alle Feature, die inkludiert sein sollen, auch inkludiert sind.

Phase vier: Stabilisierung

Die Phase beginnt mit den Beta-Tests des Produkts und endet, wenn der Kunde das Produkt als fertig ansieht. Das Testen während dieser Phase fokussiert sich auf Benutzer und „Real-World“ Tests. Das Augenmerk des Teams liegt auf dem Auffinden und Ausbessern von Bugs, um das Produkt auf jenen Stand zu bringen, dass es für eine Auslieferung bereit ist. Wenn das Team den letzten Meilenstein erreicht, wird das Produkt dem Operationsmanagement übergeben, das Team beginnt von Neuem mit dem MSF-Prozessund bereitet sich auf das nächste Release vor.

3.5.3 Prinzipien des MSF Prozessmodells

Die vier grundlegende Prinzipien, auf denen das MSF Prozessmodell basiert, werden in diesem Abschnitt kurz vorgestellt. Es sind dies der Projekt-Tradeoff, lebende Dokumente, Meilensteine und „*versioned releases*“.

Projekt-Tradeoff

Die Variablen, die in jedem Projekt eine besondere Beachtung benötigen, sind Ressourcen in Form von Personal und Geld, Termine (Zeit) und Features (das Produkt und seine Qualität). Während der Entwicklung des Produkts ist es unvermeidlich, zwischen diesen drei Variablen einen Tradeoff zu finden. Die richtige Balance zwischen diesen drei Größen ist essenziell für den Erfolg eines Projekts. Diese Variablen können in Form eines Dreiecks zueinander in Beziehung gesetzt werden. Die Änderung einer Größe führt zwangsläufig auch zur Änderung zumindest einer der beiden anderen Größen.

Ein Projekt kann nur dann erfolgreich sein, wenn der Kunde auch davon überzeugt ist, dass das Team die richtige Balance zwischen diesen Faktoren gefunden hat. Daher ist es von großer Bedeutung, dass sich das Team über die Prioritäten des Kunden im Klaren ist.

Lebende Dokumente

Das MSF versucht, die Lücke zwischen der Projektdurchführung und dem „Denken hinter dem Projekt“ mit so genannten lebenden Dokumenten zu überbrücken. Es kann durchaus vorkommen, dass Projekte in einer „Analyse/Paralyse“ hängen bleiben, endlose Planung, ohne dass irgendwelche Aktionen durchgeführt werden. Die lebenden Dokumente sollen die Richtung zeigen und spezifizieren, wann mit dem Projekt begonnen wird. Sie

können sich jedoch im Laufe des Projekts aufgrund neuer Informationen immer wieder ändern. Solche Dokumente zu erstellen, erlaubt es dem Team eine Balance zwischen zu wenig und zu viel Planung zu erreichen. Die zwei grundlegenden Eigenschaften dafür sind:

- Frühe Grundstruktur: Die Grundstruktur des Dokumentes sollte so früh als nur möglich erstellt werden, und als Basis für das fertige Dokument dienen. Das Team sollte mit der Entwicklung von Lösungen fortfahren, wenn auch noch nicht alle Fragen in dem Dokument beantwortet sind.
- Spät „einfrieren“: Dieser Grundsatz bedeutet ganz einfach, das Dokument dynamisch zu halten, und Antworten und neue Details im Zuge des Projekts einzubinden.

Lebende Dokumente erlauben es dem Team also, schon mit der Entwicklung zu beginnen, auch wenn noch nicht alle offenen Fragen geklärt worden sind. Das Projektteam kann jedes Mal, wenn genug neue Details vorhanden sind, weiter voranschreiten.

Meilensteine

Meilensteine sind formale Checkpunkte, um den Fortschritt des Projekts zu messen und sich über die Projektrichtung einig zu sein. Das MSF Modell unterscheidet zwischen Intermims- und Hauptmeilensteinen. Während die Hauptmeilensteine den Übergang von einer Phase zur nächsten markieren, gliedern Interimsmeilensteine große Arbeitspakete in kleinere überschaubare Segmente.

Bei einem Hauptmeilenstein synchronisieren alle Teammitglieder ihre Arbeit. Der Kunde und das Team einigen sich auf den Übergang in die nächste Phase und bestimmen damit den Projektfortschritt. Diese Meilensteine sind im Prozessmodell bereits vordefiniert, während die Interimsmeilensteine vom Team selbst erstellt werden. Die wichtigsten Funktionen der Hauptmeilensteine im MSF kurz zusammengefasst:

- Review und Synchronisationspunkte: Eine Möglichkeit für das Team, den Projektfortschritt zu messen und Änderungen im Kurs vorzunehmen. Diskussionspunkte, die erörtert werden können: was ging gut, was nicht? Was hätte besser funktionieren können, was können wir dokumentieren, um zukünftigen Projekten zu helfen?
- Gemeinsame Einigung von Kunde und Team über den Projektfortschritt.

Versionierte Releases

Dieses Prinzip ist eine fundamentale Technik im MSF-Prozessmodell. Sie teilt große Projekte in kleinere, so genannte „versioned releases“ ein, beginnend mit dem ersten Release als dem Kernprodukt, während nachfolgende Releases inkrementell so lange neue Features hinzufügen, bis das Produkt der Projektvision entspricht.

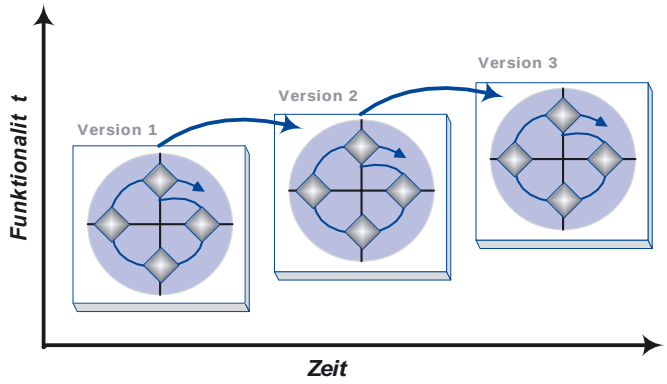

Abbildung 3.14: Versionierte Releases

Mit Hilfe dieses Konzepts ist es für Teams möglich, innerhalb kürzerer Zeit die wichtigsten Teile des Produkts fertig zu stellen, da nicht alle Features in der ersten Version vorhanden sein müssen. Zudem erleichtert und erweitert es die Möglichkeiten des Teams auf Änderungen bezüglich des Umfangs, der Risiken und der Zeitplanung. Die Vorteile dieses Ansatzes laut dem MSF sind daher:

- kürzere Auslieferungszeit für das Kernprodukt,
- klare Zielsetzungen,
- inkrementelle Featureentwicklung.

3.6 Extreme Programming (XP)

Extreme Programming ist ein neun Jahre alter Ansatz für Softwareentwicklung, der bereits bei so namhaften Unternehmen wie der Bayerischen Landesbank, der Credit Swiss Life, Daimler Chrysler oder der Ford Motor Company zum Einsatz gekommen ist. Der Fokus von Extreme Programming liegt darauf, dem Kunden eine Software vorzulegen, die seinen Wünschen und Vorstellungen zu dem Zeitpunkt entspricht, zu dem er auch diese Wünsche und Vorstellung hat. Zu diesem Zweck ist es bei Extreme Programming vorgesehen, selbst in einem fortgeschrittenen Projekt auf Änderungen der Kundenwünsche und Anforderung zu reagieren.

Extreme Programming ist zwar nicht das alleinige Werk von Kent Beck, ist aber vor allem wegen seiner zahlreichen Publikationen zu dem Thema stark mit diesem Namen verbunden.

Kent Beck arbeitet als selbstständiger Berater und weist zahlreiche Publikationen im Bereich des objektorientierten Software Engineering auf. Kent Beck ist Co-Autor an der Publikation „A Laboratory for Teaching Object-Oriented Thinking“, in der CRC-Karten erstmals vorgestellt wurden. Er ist Mitinitiator der „Hillside Group“, welche als die Triebfeder hinter der gesamten *Software Muster Community* gilt (und die auch die meisten Software-Muster-Konferenzen organisiert oder unterstützt).

Extreme Programming ist ein Modell, das die Kommunikation zwischen den Entwicklern untereinander und zwischen den Entwicklern und Kunden einen hohen Stellenwert beimisst. Zu diesem Zweck soll der Entwurf selbst so einfach als möglich gestaltet und am ersten Tag des Projekts mit dem Testen des Produkts begonnen werden, um das Produkt und die Ergebnisse so schnell als möglich dem Kunden vorlegen zu können. *Abbildung 3.15* zeigt den Ablauf eines Projekts, das mit Extreme Programming durchgeführt wird.

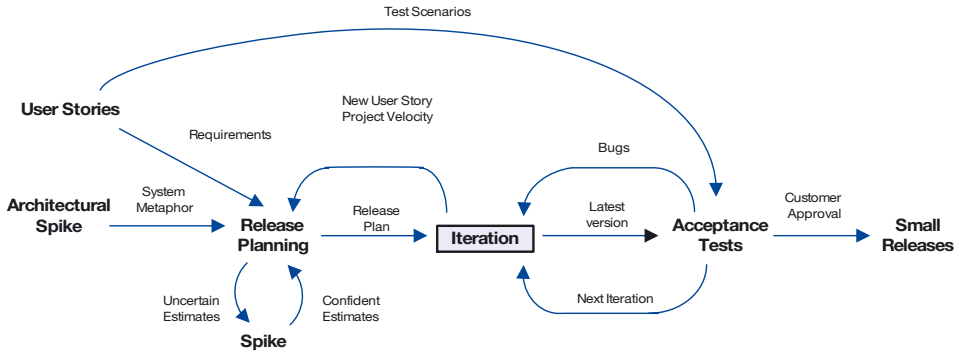

Abbildung 3.15: Ein XP-Projekt

Extreme Programming eignet sich nicht für jedes Softwareprojekt, sondern wurde besonders für Software entwickelt, deren Anforderungen sich in regelmäßigen Abständen ändern, oder für Softwareprojekte, in denen die genauen Anforderungen vom Kunden erst im späteren Verlauf festgelegt werden. Zudem wurde Extreme Programming zu Beginn nur für kleinere Gruppen von Entwicklern, am besten zwischen zwei und zwölf, entworfen, doch durch den Erfolg wurde der Ansatz auch bereits für Projekte mit mehr als 30 Beteiligten angewendet. Dennoch gibt es bei größeren Projekten Umsetzungsprobleme, da ein Kernpunkt von Extreme Programming das aktive und direkte Zusammenarbeiten von Managern, Kunden und Entwicklern darstellt, die zusammen Termine, Tests und Umfang der Arbeiten festlegen, was gerade bei großen Projekten zu organisatorischen Schwierigkeiten führen kann.

3.6.1 Werte, Prinzipien, Techniken

XP definiert Werte, Prinzipien und Grundtechniken. Die Werte, auf denen Extreme Programming als „Philosophie“ beruht, sind:

- Einfache Lösungen: Einfache Lösungen sind schnell und kostengünstig zu erstellen, lassen sich einfach erklären, warten und weiterentwickeln.
- Feedback: Feedback wird als wesentlicher Mechanismus zur Qualitätssicherung betrachtet. Feedback gibt es auf unterschiedlichen Ebenen (Anwender an Entwickler, Entwickler an Entwickler).
- Kommunikation: Kommunikation zum effektiven Informationsaustausch und der damit verbundenen Vermeidung von Missverständnissen wird hervorgehoben. Damit einher geht auch der sparsame Umgang mit Dokumentation, die durch gute Kommunikation überflüssig wird.
- Mut: Die bisher genannten Werte verlangen viel Mut, da sie oft im Gegensatz zu althergebrachten Arbeitsweisen, Gewohnheiten und der allgemein anerkannten Wertekultur stehen (z.B. wird Feedback meist mit negativem Feedback assoziiert und daher oft eher vermieden als gefördert).

Die Prinzipien, die auf den Werten von Extreme Programming aufbauen, gliedern sich in:

1. Fünf zentrale Prinzipien: Unmittelbares Feedback (*Rapid Feedback*), Einfachheit anstreben (*Assume Simplicity*) Inkrementelle Veränderung (*Incremental Change*), Veränderung wollen (*Embracing Change*) und Qualitätsarbeit (*Quality Work*).
2. Weitere Prinzipien: Lernen Lehren (*Teach Learning*), Geringe Anfangsinvestition (*Small Initial Investment*), Auf Sieg spielen (*Play To Win*), Gezielte Experimente (*Concrete Experiments*), Offene, Aufrichtige Kommunikation (*Open, Honest Communication*), Die Instinkte des Teams nutzen, nicht dagegen arbeiten (*Work With Peoples Instincts, Not Against Them*), Verantwortung übernehmen (*Accepted Responsibility*), An örtliche Gegebenheiten anpassen (*Local Adaptations*), Mit leichtem Gepäck reisen (*Travel Light*) und Ehrlich Messen (*Honest Measurement*).

Die in XP definierten zwölf Grundpraktiken (*core practices*) sind: Kunde vor Ort (*On-Site-Customer*), Planungsspiel (*Planning Game*), Methaper (*Metaphor*), Einfaches Design (*Simple Design*), 40-Stunden-Woche (*40-Hour` Week*), Refactoring (*Refactoring*), Programmierung in Paaren (*Pair Programming*), Testen (*Testing*), Programmierstandards (*Coding Standards*), Gemeinsame Verantwortlichkeit (*Collective Ownership*), Fortlaufende Integration (*Continous Integration*), Kurze Releasezyklen (*Short Releases*).

3.6.2 Prozess

Die vier grundlegenden Arbeitsschritte der Softwareentwicklung, Planung, Entwurf, Implementierung und Test, finden sich auch im Modell des Extreme Programming wieder. Für jeden dieser vier Teile sind Regeln und vorgeschlagene Durchführungsschritte definiert, die eine erfolgreiche Durchführung eines Extreme Programming-Projekts garantie-

ren sollen. Im Folgenden sind nur jene Konzepte von Extreme Programming beschrieben, die unmittelbar mit dem Prozess von Extreme Programming in Zusammenhang stehen.

Release Planning

Das „Release Planning Meeting“ wird dazu genützt, einen sog. release plan zu erstellen, der einen Überblick über den weiteren Verlauf des Projekts gibt. Dieser Plan wird dann weiter dazu verwendet, um Iterationspläne für jede weitere Iteration zu generieren. Die Hauptaufgabe bei diesem Meeting besteht darin, den Aufwand für die vom Kunden erstellten User Stories abzuschätzen. Der Kunde setzt dann die Prioritäten fest, in welcher Reihenfolge die Stories umgesetzt werden sollen.

Iterationen

Um die Agilität des Projekts zu bewahren, wird das gesamte Vorhaben in Iterationen aufgeteilt, deren Dauer zwischen einer und drei Wochen betragen sollte. Die Länge der Iterationen soll über die gesamte Dauer des Projekts konstant gehalten werden, und ist sozusagen der Herzschlag des Projekts. In speziellen Meetings werden die Iteration geplant, „just-in-time“-Planung wird beim Extreme Programming groß geschrieben.

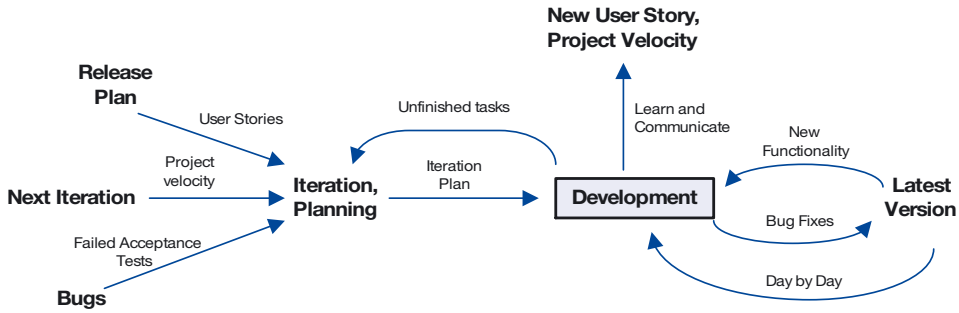

Abbildung 3.16: XP-Iteration

Iteration Planing Meeting

Zu Beginn jeder Iteration wird ein Meeting einberufen, um die Iteration zu planen. Dabei werden vom Kunden User Stories ausgewählt sowie durch Tests aufgedeckte Fehler zur Korrektur eingeplant. Dabei werden die User Stories und fehlerhaften Tests in „Program Tasks“ eingeteilt. Diese Tasks werden nun in der Sprache der Entwickler geschrieben, wobei es vorkommen kann, dass unterschiedliche User Stories zu gleichen Tasks führen, die dann zu einem Task zusammengefasst werden. Jeder dieser Tasks wird dann einem Entwickler zugeteilt, der dessen Aufwand abschätzt, wobei die Dauer jedes Tasks zwischen einem und drei Entwicklungstagen betragen sollte. Mit Hilfe der Projektgeschwindigkeit kann nun abgeschätzt werden, ob die Iteration überbucht ist oder nicht. Sollte eine Überbuchung auftreten, so liegt es am Kunden, User Stories zu entfernen, die in einer späteren Iteration durchgeführt werden.

Zusammenfassung

- Software Engineering-Prozesse gliedern sich in deren abstrakten Betrachtung in strategische Vorgehensmodelle, Arbeitschritte und konkrete Aktivitäten.
- Ein Vorgehensmodell gibt dem gesamten Modell eine Prozessarchitektur.
- Arbeitsschritte gliedern das konkrete Vorgehen in leicht zu überblickende Einheiten und definieren auch ein bestimmtes Abstraktionsniveau der darin erstellten Produkte.
- Neben der Auflistung der konkreten Aktivitäten, die schließlich von einzelnen Entwicklern ausgeführt werden, wird in einem Software Engineering-Prozessmodell auch die Abfolge der Aktivitäten festgelegt.
- Der Unified Process ist ein Metamodell, d.h. er muss für eine konkrete Anwendung noch entsprechend erweitert und adaptiert werden (z.B. durch Produktvorlagen). Der Rational Unified Process ist eine konkrete Implementierung des UP.
- Sowohl der Rational Unified Process als auch das Microsoft Solution Framework werden von namhaften Softwareunternehmen entwickelt und propagiert, welche einschlägige Werkzeuge für die Softwareherstellung erzeugen und damit den Vertrieb ihrer Werkzeuge unterstützen wollen.
- Extreme Programming ist ein Vertreter der so genannten „lightweigt“ bzw. agilen Prozesse. Der Prozess gibt nur ein Mindestmaß an Regeln vor und wird laufend, je nach Projektbedingungen, adaptiert.
- Extreme Programming wurde von einer relativ unabhängigen Gruppe von Projektmanagern entworfen und publiziert.

Übungen und Fragen

1. Vergleichen Sie Vor- und Nachteile der beschriebenen Vorgehensmodelle.
2. Welches Vorgehensmodell würden Sie für die Durchführung Ihrer Projekte wählen? Begründen Sie Ihre Antwort.
3. Welche Erweiterung nimmt das V-Modell im Vergleich zu dem ihm zugrunde liegenden Wasserfallmodell vor?
4. Was ist die Besonderheit des Spiralmodells?
5. Beschreiben Sie wesentlichen Neuerungen der iterativen und inkrementellen Vorgehensweise im Vergleich zu anderen Vorgehensweisen.
6. Worin unterschiedet sich der RUP vom Unified Process?
7. Worin unterscheiden sich RUP, MSF und XP bezüglich der zu erstellenden Produkte?
8. Worin unterscheiden sich RUP, MSF und XP bezüglich der auszuführenden Aktivitäten?

9. Worin unterscheiden sich RUP, MSF und XP bezüglich der vorgesehenen Rollen im Projekt?
10. Für welche Projekttypen halten Sie RUP, MSF und XP besonders geeignet? Für welche gar nicht?

Weiterführende Literatur

[Beck00] Beck, Kent: *Extreme Programming*; München: Addison-Wesley, 2000 (dt. Ausgabe).

[Jaco99] Jacobson, Ivar; Booch, Grady; Rumbaugh, James: *The Unified Software Development Process*; Reading, Mass. [u.a.]: Addison-Wesley, 1999.

[Kruc00] Kruchten, Philippe: *The Rational Unified Process: An Introduction*; Reading, Mass. [u.a.]: Addison-Wesley, 2. Aufl., 2000.

[Rumb99] Rumbaugh, James; Jacobson, Ivar; Booch, Grady: *The Unified Modeling Language Reference Manual*; Reading, Mass. [u.a.]: Addison-Wesley, 1999.

[Scha99]Schach, Stephen R.: *Classical and Object-Oriented Software Engineering*; Boston: McGraw-Hill, 4. Aufl., 1999.

[Stev01] Stevens, Perdita; Pooley, Rob; *UML. Software Engineering mit Objekten und Komponenten*; München: Pearson Studium, 2001.