Prof. Dr. Andreas Henrich, Birgit Feldmann, Dr. Eberhard Heuel

Kurs 01895

Management von Softwareprojekten

LESEPROBE

Fakultät für
Mathematik und
Informatik

FernUniversität in Hagen

Das Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere das Recht der Vervielfältigung und Verbreitung sowie der Übersetzung und des Nachdrucks bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder ein anderes Verfahren) ohne schriftliche Genehmigung der FernUniversität reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Inhaltsverzeichnis

1 Einleitung und Grundlagen 3

1.1 Motivation 3

1.2 Überblick über den Aufbau des Kurses 4

1.3 Zum Begriff des Softwareprojekts 8
1.3.1 Was ist ein Projekt? 8
1.3.2 Was ist ein Softwareprojekt? 9
1.3.3 Was sind die Unterschiede zu anderen Projekten? 13

1.4 Ziele des Managements von Softwareprojekten 16

1.5 Teilaufgaben des Managements von Softwareprojekten 20

1.6 Vorgehensmodelle für Softwareprojekte 26
1.6.1 Grundidee der Vorgehensmodelle 28
1.6.2 Einordnung von Vorgehensmodellen 31
1.6.3 Das klassische Wasserfallmodell 35
1.6.4 Das Wasserfallmodell mit Rücksprüngen 39
1.6.5 Projektbegleitende Qualitätssicherung 40
1.6.6 Prototyping 43
1.6.7 Das Spiralmodell 45
1.6.8 Das V-Modell 48
1.6.9 Rational Unified Process 53
1.6.10 eXtreme Programming 60
1.6.11 Scrum 66

1.7 Zusammenfassung 70

Lehrziele 73

Literaturverzeichnis 75

Index zu Kurseinheit 1 78

1 Einleitung und Grundlagen

1.1 Motivation

niederschmetternde Statistiken

Die Literatur über Softwareprojekte ist voll von niederschmetternden Statistiken über deren Erfolgsaussichten. Wohl am häufigsten zitiert werden dabei die CHAOS-Reports der Standish Group (www.standishgroup.com), die regelmäßig der Frage nachgehen, was aus Softwareprojekten wurde und ob sie erfolgreich waren. Die Bilanz, die diese Reports ziehen, hat sich zwar seit der ersten Untersuchung 1994 verbessert, ist aber immer noch erschreckend. Demnach wurden Softwareprojekte im Jahre 2012 mit folgendem Ergebnis abgeschlossen:

- Erfolgreich (*successful*) 39 %
- Teilweise erfolgreich (*challenged*) 43 %
- Misserfolg/Abbruch (*failed*) 18 %

Indikatoren für den Projekterfolg sind die Zielerreichung bzw. die Zielabweichung bei Zeit, Kosten, Funktionsumfang und Qualität.

hohe Kostenabweichungen

Insgesamt hat sich wenig an dem negativen Bild geändert, das bereits eine Untersuchung von A.W. Feyhl und E. Feyhl aus den 90er-Jahren zeichnet /Feyhl, Feyhl 1996/. Hier wurde bei einer Analyse von 162 Projekten in 50 Organisationen deren Kostenabweichung ermittelt. Tabelle 1 zeigt, dass bei dieser Analyse über 70% der Projekte eine Kostenabweichung von mindestens 50% gegenüber der ursprünglichen Planung aufwiesen.

	Kostenabweichung				
	keine	< 50 %	50 bis 100 %	> 100 %	nicht bekannt
Projekte	12	14	59	55	22

Tabelle 1: Angaben zu Kostenabweichungen von 162 Projekten (nach /Feyhl, Feyhl 1996/, S. 1)

Ursache: schlechtes Management

Solche negativen Statistiken lassen sich durch eine Vielzahl von Fallbeispielen ergänzen. In einem sehr plastischen Beitrag hat beispielsweise Robert N. Charette /Charette 2005/ eine lange Liste von Softwareprojekten amerikanischer Unternehmen zusammengestellt, die teilweise spektakulär gescheitert sind. Als Gründe für das Scheitern ermittelt er unter anderem folgende Faktoren:

- Unrealistische oder nicht ausformulierte Projektziele
- Ungenaue Aufwandsabschätzung
- Schlecht spezifizierte Systemanforderungen
- Unzulängliches Berichtswesen
- Nicht beherrschte Risiken
- Mangelnde Kommunikation zwischen den Projektbeteiligten

- Nicht beherrschte Komplexität des Projekts
- Mangelndes Projektmanagement

Alle aufgeführten Faktoren machen letztlich deutlich: Die meisten Projekte scheitern nicht in erster Linie an technischen Problemen, sondern an dem, was man am ehesten als „*schlechtes Management*“ bezeichnen könnte. Dies deckt sich auch mit den Aussagen von Boehm /Boehm 1981/, der in seinem grundlegenden Buch *Software Engineering Economics* feststellt, dass nichts die Kosten eines Softwareprojekts so nachhaltig negativ beeinflussen kann wie ein schlechtes, planloses Management und ständig wechselnde Anforderungen.

Ziel des Kurses

Wenn ein schlechtes Management von Softwareprojekten so fatale Folgen hat, dann stellt sich natürlich umso dringlicher die Frage, was *gutes* Projektmanagement auszeichnet. Der vorliegende Kurs versucht, (im Rahmen seiner Möglichkeiten) eine Antwort auf diese Frage zu geben. Dazu werden die erforderlichen Grundlagen aus dem Bereich der Betriebswirtschaftslehre ebenso betrachtet wie spezifische Aspekte des Projektmanagements und insbesondere des Managements von Softwareprojekten. Ziel des Kurses ist es, dem Projektmitarbeiter und dem Projektverantwortlichen in kompakter Form die für das Projektmanagement erforderlichen und sinnvollen Grundlagen, Techniken und Methoden zu vermitteln. Dabei steht nicht etwa ein ganz konkretes Vorgehensmodell im Mittelpunkt – wie etwa das *V-Modell* /V-Modell XT 1.4 2012/, der *Rational Unified Process* /Rational 2001/, das *eXtreme Programming* /Beck 1999/ oder *Scrum* /Schwaber, Beedle 2001/ –, auf die wir durchaus eingehen werden. Vielmehr behandelt der Kurs den gesamten Bereich des Managements von Softwareprojekten.

1.2 Überblick über den Aufbau des Kurses

Um den Inhalt des vorliegenden Kurses einordnen zu können, sollte zunächst deutlich werden, dass bei der Durchführung eines Projekts drei Bereiche zusammenwirken müssen:

- *Softwaretechnik*

  Der Bereich der Softwaretechnik umfasst die Prinzipien, Methoden und Techniken zur Entwicklung großer Softwaresysteme. Hierzu gehören Analysemethoden, die sich mit der Bestimmung der fachlichen Anforderungen an das System beschäftigen, ebenso wie Designmethoden, die die Entwicklung einer technischen Systemarchitektur aus modularen Bausteinen zum Ziel haben (vgl. hierzu die Ausführungen in Abschnitt 1.6.3). Ferner werden in der Softwaretechnik insbesondere auch Implementierungstechniken und Testverfahren betrachtet.

- *Software-Qualitätssicherung*

  Software-Qualitätssicherung beschäftigt sich mit den Tätigkeiten, die dazu dienen, den Nachweis zu erbringen, dass die Qualitätsanforderungen an die Software erfüllt sind. Dazu dienen insbesondere Techniken zum Testen von Software. Fragen der Testfallermittlung werden hier ebenso behandelt wie Fragen des Testumfangs. Die *Testfallermittlung* bezieht sich dabei auf die Vorgabe der in den Tests zu betrachtenden Testfälle für die Software. Bei der Festlegung und Bestimmung des *Testumfangs* geht es um die kritische Frage, wann „genug“ getestet wurde.

- *Software-Projektmanagement*

  Das Software-Projektmanagement beschäftigt sich primär mit der Wahrnehmung der klassischen Managementaufgaben (vgl. z.B. /Wöhe, Döring 2010/) *Ziele setzen*, *Planen*, *Entscheiden*, *Realisieren* und *Kontrollieren* im Rahmen der Abwicklung eines Softwareprojekts. Abbildung 1 stellt diese Aufgaben in der üblichen Darstellungsweise als „Management-Kreis“ dar.

klassische Managementaufgaben

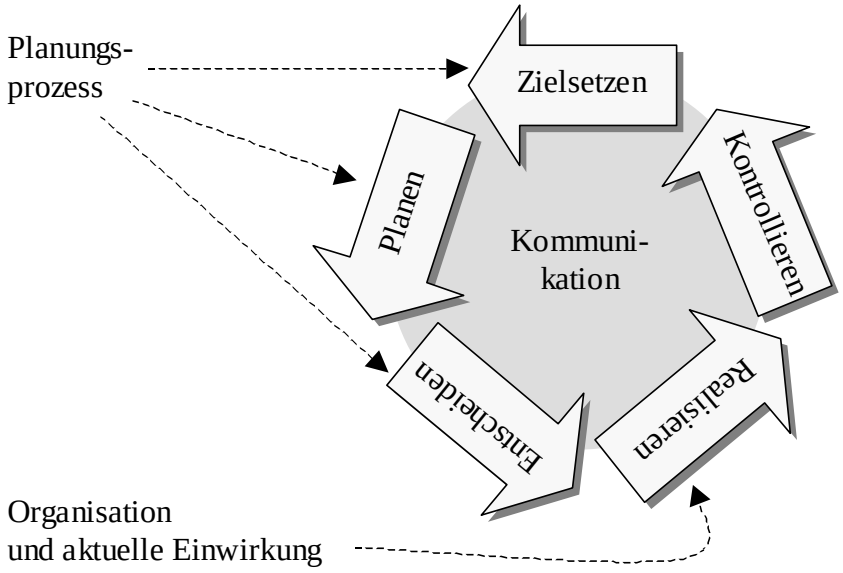


Abbildung 1: Management-Kreis (nach /Wöhe, Döring 2010/, S.89)

Damit bildet das Software-Projektmanagement den Rahmen, in dem die Prinzipien und Methoden der Softwaretechnik und der Software-Qualitätssicherung angewendet werden können und müssen.

Die beschriebenen Bereiche sind dabei nicht überlappungsfrei. Abbildung 2 verdeutlicht die Abgrenzung und die sich ergebenden Schnittbereiche.

Der vorliegende Kurs beschäftigt sich nun primär mit dem Management von Softwareprojekten (in Abbildung 2 grau hinterlegt). Aspekte der Softwaretechnik und der Software-Qualitätssicherung außerhalb der Schnittbereiche werden wir nur dann in die Betrachtung mit einbeziehen, wenn dies zum Verständnis erfor-

derlich ist[1]. Diese Abgrenzung hat zwei Ursachen. Zum einen würde eine umfassende Betrachtung aller drei Bereiche den Rahmen eines Kurses bei Weitem übersteigen, und zum anderen sind die meisten Prinzipien des Managements von Softwareprojekten unabhängig von den konkreten in einem Projekt eingesetzten Verfahren der Softwaretechnik und der Software-Qualitätssicherung anwendbar.

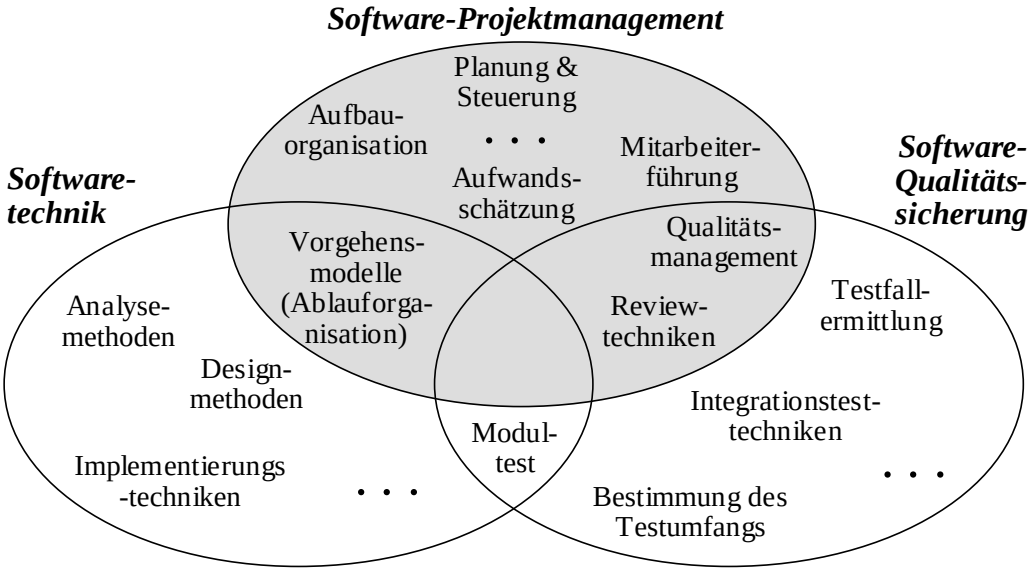


Abbildung 2: Abgrenzung des Managements von Softwareprojekten

Bei der Behandlung der Teilaspekte des Managements von Softwareprojekten werden wir wie folgt vorgehen:

Einleitung

In den Abschnitten 1.3, 1.4 und 1.5 werden wir zunächst einführend unser Verständnis eines Softwareprojekts definieren und einen Überblick über die Aufgaben beim Management von Softwareprojekten geben.

Organisation

Danach werden wir uns in diesem und im folgenden Kapitel mit Fragen der Projektorganisation beschäftigen. Dabei geht es zum einen um die *Ablauforganisation*. Die Ablauforganisation wird im Rahmen von Softwareprojekten in der Regel durch ein sogenanntes *Vorgehensmodell* beschrieben. Wir werden hierzu einige typische Vertreter diskutieren. In Kapitel 2 werden wir uns den Fragen der *Aufbauorganisation* widmen. Dabei geht es sowohl um die Integration des Projekts in die umgebende Aufbauorganisation des Unternehmens als auch um die projektinterne Aufbauorganisation.

Kosten/Nutzen-Analysen

Kapitel 3 betrachtet *Kosten/Nutzen-Analysen* für Softwareprojekte. Den Ausgangspunkt bildet eine Klassifikation der Kosten- und der Nutzenarten. Ein besonderes Augenmerk liegt hierbei auf der Differenzierung in quantifizierbare und nicht quantifizierbare Kosten- und insbesondere Nutzenarten sowie auf dem As-

1 Für eine detailliertere Betrachtung der Softwaretechnik und der Software-Qualitätssicherung sei auf entsprechende Lehrbücher verwiesen (z.B. /Pagel, Six 1994/; /Balzert 2008/; /Liggesmeyer 2009/; /Ludewig 2010/).

pekt der Unsicherheit der Plandaten. Anschließend werden zunächst aus der Betriebswirtschaftslehre bekannte Verfahren der Investitionsrechnung betrachtet, die auch auf Investitionen in Software angewendet werden können. Den Abschluss des Kapitels bildet ein Überblick über Entscheidungstechniken. Als Beispiel sei an dieser Stelle lediglich die Nutzwertanalyse genannt.

Aufwandsschätzung

In Ergänzung zu den in Kapitel 3 betrachteten Kosten/Nutzen-Analysen werden in Kapitel 4 Verfahren zur *Aufwandsschätzung* für Software-Entwicklungsprojekte betrachtet. Ziel dieser Schätzverfahren ist es, zum einen eine Basis für Kosten/Nutzen-Analysen zu bieten und zum anderen eine fundierte Ressourcenplanung für das Projekt zu ermöglichen.

Projektplanung und Projektcontrolling

Kapitel 5 beschäftigt sich mit der *Projektplanung* und dem *Projektcontrolling* für Softwareprojekte. Dabei werden zunächst im Rahmen der *Projektplanung* Fragen nach dem richtigen Umfang der Planung, nach der Differenzierung in Teilpläne sowie nach dem Vorgehen bei der Planung betrachtet. In diesem Zusammenhang werden u.a. die Verfahren der Netzplantechnik und ihre Anwendung in einem Software-Entwicklungsprojekt behandelt. Im Hinblick auf die *Steuerung* und das *Controlling* von Softwareprojekten werden wir uns mit dem Einsatz von Metriken – d.h. Kennzahlen –, mit dem Berichtswesen sowie mit der Frage, was getan werden kann, wenn Planabweichungen festgestellt werden, befassen.

Techniken des Projektmanagements

Kapitel 6 widmet sich ergänzenden *Techniken* zu folgenden Bereichen *des Projektmanagements*: Kreativitätstechniken, Risikomanagement, Besprechungen, Projektpräsentationen, Dokumentation sowie Versions- und Konfigurationsmanagement.

„die menschliche Komponente“

Gegenstand des Kapitels 7 sind Fragen der Motivation der Mitarbeiter, der Zusammenarbeit in der Gruppe sowie der Mitarbeiterführung im Rahmen von Softwareprojekten. Daneben werden auch Aspekte wie die Einstellung neuer Projektmitarbeiter und Projektmitarbeiterinnen sowie die Personalentwicklung betrachtet.

Qualitätssicherung und Qualitätsmanagement

Kapitel 8 schließlich widmet sich der *Qualitätssicherung* und dem *Qualitätsmanagement* in Softwareprojekten. Hierzu werden zunächst die Begriffe Qualität, Qualitätssicherung und Qualitätsmanagement gegeneinander abgegrenzt. Im Anschluss werden als Teilaspekte der Qualitätssicherung die Qualitätsplanung, die Qualitätslenkung und die Qualitätsprüfung näher betrachtet. Als Beispiel für eine mögliche Ausprägung der Qualitätssicherung werden die entsprechenden Teile des V-Modells dargestellt. Im Anschluss wird als weiterführender Ansatz das Qualitätsmanagement und insbesondere das Total Quality Management (TQM) vorgestellt. Den Abschluss des achten Kapitels bildet eine Betrachtung des Qualitätsmanagements nach der Normenreihe ISO 9000.

1.3 Zum Begriff des Softwareprojekts

Bevor wir uns mit den einzelnen Aspekten des *Managements von Softwareprojekten* beschäftigen, muss zunächst genauer definiert werden, was wir unter dem *Management eines Softwareprojekts* verstehen wollen. Dazu werden wir in den folgenden Abschnitten einleitend folgende Fragen klären:

- Was ist ein Projekt?
- Was ist ein *Software*projekt?
- Welche *Ziele* werden mit dem Management von Softwareprojekten verfolgt?
- Welche *Teilaufgaben* gehören zum Management von Softwareprojekten?

Nach Klärung dieser grundlegenden Fragen werden wir dann detaillierter auf die einzelnen Teilaufgaben des Projektmanagements eingehen.

1.3.1 Was ist ein Projekt?

Wenn wir von einem Softwareprojekt sprechen, dann ist dieser Begriff in zweierlei Hinsicht erklärungsbedürftig. Zum einen ist zu klären, was wir unter einem Projekt im Allgemeinen verstehen, und zum anderen ist zu klären, welche speziellen Projekte wir mit Softwareprojekten ansprechen wollen. Den erstgenannten Aspekt werden wir in diesem Abschnitt und den zweitgenannten in Abschnitt 1.3.2 ansprechen.

Zum allgemeinen Projektbegriff hat sich in der Literatur ein relativ klares Verständnis entwickelt (vgl. hierzu z.B. /Lehner et al. 1991/; /Grupp 1998/; /Kellner 2001/; /Corsten et al. 2008/; /Kraus, Westermann 2010/). Danach ist ein Projekt wie folgt charakterisiert:

Charakteristika eines Projekts

- *zeitliche Begrenzung*

  Ein Projekt ist zeitlich begrenzt. Es handelt sich *nicht* um eine permanente Aufgabe – wie z.B. den Einkauf, den Verkauf oder die Buchhaltung –, sondern um ein Vorhaben, das durch einen Anfangs- und einen Endzeitpunkt charakterisiert werden kann.

- *klare Aufgabendefinition*

  Daneben ist auch das Ziel bzw. die zu erstellende Leistung vorgegeben. In der Regel handelt es sich dabei um eine Entwicklungstätigkeit, die klar definiert und abgegrenzt ist.

- *hohe Komplexität*

  Ein weiteres wichtiges Kriterium ist die Komplexität. Das Vorhaben ist keine sich wiederholende Routineaufgabe, sondern durch eine gewisse

Einzigartigkeit oder Neuartigkeit gekennzeichnet. Es handelt sich eben nicht um die bloße Wiederholung früherer Arbeiten.

Aus dieser Einzigartigkeit folgt unmittelbar auch eine relativ hohe Unsicherheit in der Planung des Projekts. Projekte sind wegen ihrer Neuartigkeit zwangsweise immer mit einem gewissen Risiko verbunden, das in der Hauptsache zunächst die Einhaltung des vorgegebenen Zeit- und Kostenrahmens betrifft.

- *Konkurrenz um begrenzte Mittel*

  Die zur Durchführung des Vorhabens verfügbaren Mittel sind begrenzt. Ein Projekt steht dabei von Natur aus in einer gewissen Konkurrenzsituation zu anderen Projekten und zu andauernden Aufgaben um personelle, finanzielle und andere Mittel.

- *mehrere beteiligte Stellen*

  An der Durchführung des Vorhabens sind mehrere Stellen innerhalb und gegebenenfalls auch außerhalb des Unternehmens beteiligt.

  Hieraus und aus der Komplexität des Vorhabens kann auch die Anforderung abgeleitet werden, dass sich die Aufgabe nicht durch die vorhandenen Betriebsinstanzen lösen lässt. Vielmehr ist eine zusätzliche temporäre Organisationseinheit erforderlich, in der die Projektdurchführung ihren Rahmen findet.

- *vorgegebener Kostenrahmen*

  Im Hinblick auf die Kosten wird häufig gefordert, dass diese im Voraus geplant oder festgelegt werden können. Offensichtlich kann dies aber wegen der aus der Neuartigkeit der Aufgabe erwachsenden Unsicherheit nur grob geschehen. Trotzdem bleibt es typisch für Projekte, dass diese in einem vorgegebenen finanziellen Rahmen abgewickelt werden müssen [2].

- *Lösbarkeit der Aufgabe*

  Des Weiteren wird häufig die relativ naheliegende Anforderung gestellt, dass die Aufgabe, die mit dem Projekt verfolgt wird, auch lösbar sein muss.

1.3.2 Was ist ein Softwareprojekt?

Nachdem der allgemeine Projektbegriff umrissen ist, stellt sich die Frage, was mit einem *Software*projekt gemeint ist. Zunächst kann man sich die Sache leicht machen und definieren, dass ein Softwareprojekt ein Projekt ist, in dem hauptsäch-

---

2 Der Aspekt der Schätzung der mit einem Softwareprojekt verbundenen Kosten und Aufwände wird uns noch ausführlich in den Kapiteln 3 und 4 beschäftigen.

lich oder ausschließlich Software entwickelt wird. Dabei ist aber zu bedenken, dass sich sehr unterschiedliche Arten von Projekten mit der Erstellung von Software beschäftigen. Abbildung 3, auf die wir im Folgenden noch ausführlicher eingehen wollen, zeigt exemplarisch die mögliche Vielfalt von Softwareprojekten auf, wobei in der Praxis sowohl hinsichtlich der Projektart als auch hinsichtlich der Art der Software häufig Mischformen anzutreffen sind.

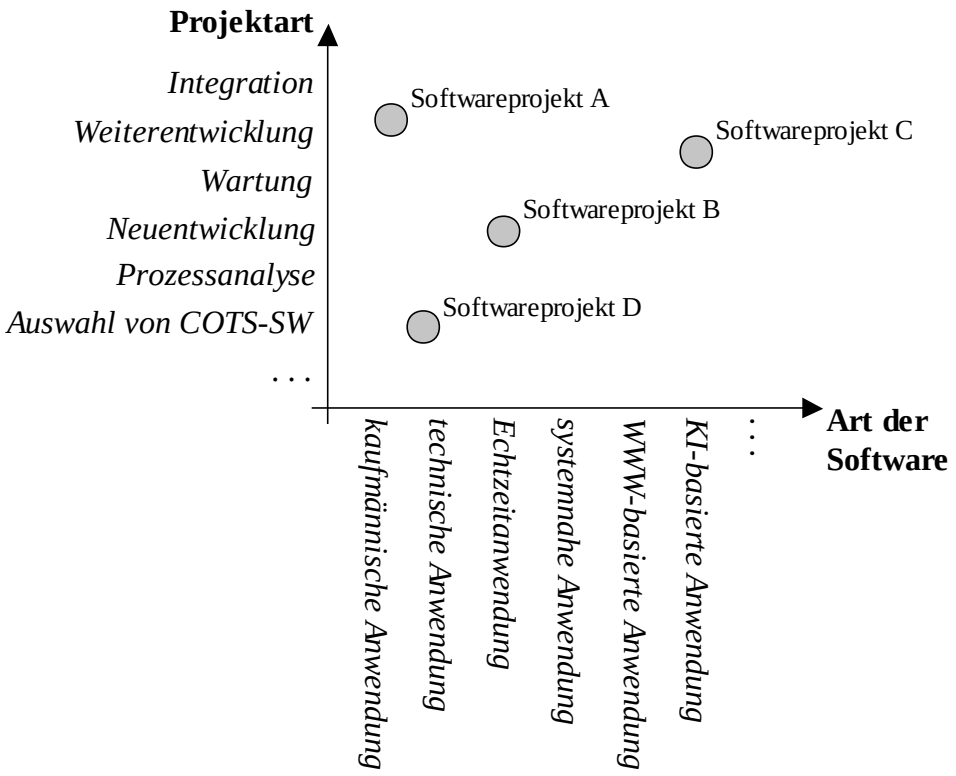


Abbildung 3: Arten von Softwareprojekten

Wir wollen zunächst die Unterscheidung nach der *Projektart* genauer betrachten.

Neuentwicklung

Als Erstes ist hier die *Neuentwicklung* eines Systems zu nennen. Diese wird dadurch charakterisiert, dass die Software – zumindest zum überwiegenden Teil – vollständig neu entwickelt wird. Es handelt sich also nicht um die Weiterentwicklung einer bereits bestehenden Software, sondern um ein neues System. Ein solches System kann sowohl eine Individualentwicklung sein, die speziell für einen Anwender entwickelt wird, als auch eine Standardlösung, die für einen breiteren Markt von Anwendern gedacht ist. Wir können hier folgende Unterarten differenzieren, die allerdings fließend ineinander übergehen:

- *Einzelentwicklung*

  Das Charakteristische für eine Einzel- oder Individualentwicklung ist der direkte Kontakt zum späteren Anwender. Der Anwender kann und soll hier eine Lösung bekommen, die möglichst genau seinen Anforderungen entspricht. Dafür muss er aber auch die gesamten Kosten der Entwicklung

tragen, die hier nicht über eine Gruppe von Anwendern verteilt werden können.

Daraus folgt, dass bei derartigen Projekten die Einbindung der Anwender in das Projekt und insbesondere in die Analyseaktivitäten, in denen der fachliche Funktionsumfang des Systems erarbeitet werden soll, sehr wichtig ist. Die hohen Kosten einer Individualentwicklung sind nämlich in der Regel nur dann zu rechtfertigen, wenn der Anwender sich durch die auf ihn zugeschnittene Funktionalität einen strategischen Vorteil gegenüber der Konkurrenz erhoffen kann.

In der Praxis können heute im Hinblick auf solche Individualentwicklungen aufgrund der hohen Kosten zwei Tendenzen beobachtet werden.

Verwendung mächtiger Basissysteme

Zum einen versucht man die Entwicklungskosten durch die Verwendung mächtiger Basissysteme zu reduzieren. Dies beginnt bei Datenbanksystemen und Kommunikationsplattformen und endet bei der individuellen Systementwicklung auf der Basis leistungsfähiger betrieblicher Standardsoftware. Dabei ist der Übergang von der Individualentwicklung zur reinen Anpassung einer Standardsoftware letztlich fließend.

Verkauf an Dritte

Die zweite zu beobachtende Tendenz besteht darin, dass man die Kosten einer Individualentwicklung durch den späteren Verkauf an Dritte wieder erwirtschaften möchte. Das System wird dabei zunächst speziell für den Erstanwender entwickelt. Später wird dieses System dann aber ggf. mit leicht modifiziertem Funktionsumfang auch anderen „befreundeten“ Unternehmen angeboten.

- *Entwicklung für eine kleine Anwendergruppe*

  Eine Möglichkeit, die Kosten einer Individualentwicklung nicht einem einzigen Anwender aufzubürden, ist die Bildung von Gruppen. So können sich z.B. Unternehmen, die ohnehin in anderer Weise verbunden sind (evtl. durch einen Einkaufsverbund), die Kosten teilen. In diesem Fall muss in dem Softwareprojekt allerdings mit einem ggf. heterogenen Anforderungsspektrum gerechnet werden. Hier tritt häufig die Frage in den Vordergrund, wie mit divergierenden Anforderungen der Anwender umzugehen ist.

- *Entwicklung für einen anonymen Markt*

  Bei einer Entwicklung für einen anonymen Markt sind die konkreten Anwender des Systems zum Zeitpunkt der Entwicklung noch nicht bekannt. Dadurch ergibt sich ein gewisses Problem hinsichtlich der Analyse der fachlichen Anforderungen. Dieses Problem kann unter anderem dadurch

adressiert werden, dass man einige möglichst repräsentative Anwender im Voraus auswählt, die dann am Entwicklungsprozess beteiligt werden.

Auswahl und Anpassung von COTS-SW

Die zweite idealtypische Projektart neben der Neuentwicklung ist die Auswahl und ggf. Anpassung sogenannter *COTS-Software* (COTS = commercial off-the-shelf). Der Begriff COTS-Software steht dabei für Software, die in einer gegebenen Grundversion „aus dem Regal“ des Softwareanbieters gekauft werden kann. Beispiele für COTS-Software reichen von den sogenannten Office-Paketen bis hin zu ERP-Systemen (ERP = Enterprise Resource Planning) wie SAP ERP 6.0 oder Sage ERP X3.

Der Schwerpunkt der Projekte liegt dabei zunächst auf der Auswahl der richtigen COTS-Software. Anschließend sind noch ggf. umfangreichere Anpassungen an das jeweilige Unternehmen erforderlich, die sich im Extremfall vom Umfang her an eine Neuentwicklung anlehnen können, sodass die Übergänge fließend sind.

Prozessanalyse

Andere Projekte beginnen nicht direkt mit der eigentlichen Softwareentwicklung oder Auswahl, sondern mit einer Analyse der betrieblichen Prozesse, die ggf. mit einem IT-System unterstützt werden könnten. Hier wird häufig eine vorgeschaltete Geschäftsprozessoptimierung mit einer anschließenden Automatisierung der Prozesse durch Workflow-orientierte Anwendungssoftware kombiniert. Man kann derartige Projekte als eine oft überlappende Folge von Teilprojekten sehen, bei der zunächst in einem initialen Projekt Geschäftsprozesse analysiert und ggf. zumindest im Plan restrukturiert werden. Anschließend wird entschieden, welche Softwareprojekte sich hieraus ergeben. Von einem Softwareprojekt im engeren Sinne ist dann erst bei diesen „Folgeprojekten“ zu sprechen.

Wartung, Weiterentwicklung, Integration

Einen letzten wesentlichen Bereich der „Softwareprojekte“ bilden Projekte, die auf bereits bestehenden Anwendungssystemen aufsetzen. Das Spektrum reicht dabei von der Beseitigung entdeckter Fehler (Wartung) bis hin zu einem vollständigen Re-Design des Systems mit stark erweiterter Funktionalität. Ferner ist auch häufig die Integration eines Systems in ein anderes, bereits bestehendes System notwendig. Eine entsprechende Situation könnte z.B. dadurch entstehen, dass in einem Unternehmen eine betriebswirtschaftliche Standardsoftware eingeführt werden soll, in die mehrere im Unternehmen im Einsatz befindliche „Altsysteme“ integriert werden müssen.

Einen anderen typischen Bereich für Integrationsprojekte bilden Firmenzusammenschlüsse oder Kooperationen. Wenn z.B. zwei Handelshäuser fusionieren, entsteht der Bedarf, die jeweiligen Anwendungssysteme zumindest teilweise zu integrieren. Ein typisches Beispiel hierfür wäre der Wunsch nach übergreifenden Auswertungen von in den Unternehmen vorhandenen Datenbeständen.

Welche Projekte betrachten wir?

Nachdem wir nun die unterschiedlichen Projektarten aufgezeigt haben, stellt sich die Frage, welche Projekte wir in diesem Kurs adressieren. Viele der behandelten Themen – wie Projektorganisation, Projektkalkulation im Sinne einer Investiti-

onsbetrachtung oder der Umgang mit Projektmitarbeitern – sind von allgemeinem Charakter. Wo eine Konkretisierung erforderlich ist – z.B. bei den Vorgehensmodellen oder bei der Aufwandsschätzung – werden wir uns allerdings primär auf die Softwareentwicklung im engeren Sinne beziehen, d.h. auf Projekte, in deren Rahmen eine umfangreichere Software neu erstellt wird, sei es als Neuentwicklung oder als umfangreiche Anpassung einer COTS-Software. Wir werden dabei aber immer wieder auch Aspekte der Softwarewartung ansprechen.

Im Hinblick auf die in dem Projekt erstellte Software werden keine Einschränkungen gemacht. Die Prinzipien, nach denen das Management eines Softwareprojekts erfolgen sollte, sind nicht grundsätzlich verschieden für kaufmännische und technische Anwendungen oder für Echtzeitsysteme und Systeme ohne Echtzeitanforderungen.

1.3.3 Was sind die Unterschiede zu anderen Projekten?

Auf der Basis unseres Verständnisses von einem Softwareprojekt ist nun zu klären, welche Unterschiede zwischen einem Softwareprojekt und einem anderen Projekt – wie z.B. dem Bau einer Lagerhalle – bestehen. Die Unterschiede haben ihre Ursache dabei zum einen in der Tatsache, dass Software ein immaterielles Produkt ist, dessen Fertigstellungsgrad und dessen Qualität nur sehr schwer visuell wahrgenommen werden können. Zum anderen handelt es sich um einen Entwicklungsprozess in einem sehr dynamischen Umfeld seitens der Technologie und der Anforderungen, der hohe Ansprüche an die Qualifikation der Beteiligten stellt.

- *Aufwands- und Kostenschätzung sind schwierig.*

  Die Ursachen hierfür sind vielschichtig. Sie reichen von der Unsicherheit, mit der alle ambitionierten Entwicklungsprojekte zu kämpfen haben, über die relativ geringe Erfahrung mit Softwareprojekten bis zu den sich schnell wandelnden technologischen und anwendungsseitigen Rahmenbedingungen. In der Folge benötigt man spezielle Verfahren zur Aufwands- und Kostenschätzung, die auf die spezifischen Erfordernisse und Rahmenbedingungen von Softwareprojekten zugeschnitten sind[3].

- *Der Entwicklungsfortschritt ist schwer abzuschätzen.*

  Zu einem beliebigen Zeitpunkt während der Projektabwicklung kann nur sehr schwer abgeschätzt werden, wie weit die zu entwickelnde Software fortgeschritten ist. Dies hat zum einen damit zu tun, dass zunächst einige Vorarbeiten zu leisten sind, die nur indirekt mit der zu erstellenden Software verbunden sind. Genannt seien hier nur Analyseaktivitäten zur Auf-

3 Derartige Verfahren werden wir in Kapitel 4 ausführlich betrachten.

nahme des Ist-Zustands oder die Einarbeitung der Projektmitarbeiter. Zum anderen besteht das Problem aber auch darin, dass die Qualität der erarbeiteten Zwischenergebnisse nur schwer bestimmt werden kann. So kann z.B. eine sehr schnell, aber mit mangelhafter Qualität erstellte Systemarchitektur zunächst einen sehr zügigen Projektfortschritt suggerieren. Dass später aber für Nacharbeiten ein erheblicher Mehraufwand erforderlich sein wird, ist – zumindest auf den ersten Blick – nicht zu erkennen.

- *Eine projektbegleitende Qualitätssicherung ist schwierig.*

  Im Gegensatz z.B. zu Bauvorhaben ist die Qualität der Zwischenergebnisse bei Softwareprojekten einer Prüfung wesentlich schlechter zugänglich. Die primäre Ursache hierfür ist wiederum, dass Software immateriell ist.

  Obwohl eine sinnvolle Qualitätssicherung hierdurch einerseits erschwert wird, ist sie andererseits umso dringlicher, um den Mangel an intuitiver Qualitätssicherung durch visuelle Kontrolle auszugleichen. Während nämlich bei Projekten, bei denen die Zwischenergebnisse materiell verfügbar sind, zahlreiche Fehlentwicklungen aufgrund des Augenscheins kaum übersehen werden können, ist hierfür bei Softwareprojekten bereits eine fundierte Qualitätssicherung erforderlich.

- *Ergebnisse und insbesondere Zwischenergebnisse sind für IT-Laien oft nicht nachvollziehbar.*

  Dieser Punkt, der eng mit dem vorgenannten Aspekt verbunden ist, führt häufig zu Problemen zwischen der Unternehmensleitung bzw. dem Auftraggeber und/oder den Anwendern des zu erstellenden Systems auf der einen und dem Projektteam auf der anderen Seite.

  Ein Aspekt dieses Problems ist, dass innerhalb des Projektteams zur Vermeidung von Missverständnissen eine möglichst eindeutige formale Notation verwendet werden sollte. Diese Notation ist aber für Außenstehende oft nur schwer verständlich. Ein anderer Aspekt liegt darin, dass häufig kein übergreifendes Begriffsverständnis existiert. Durch diese Kommunikationsprobleme entsteht häufig ein Misstrauen, das ein gemeinsames Arbeiten und ein Verständnis für die gegenseitigen Arbeitsleistungen erschwert.

- *Der Zusammenhang zwischen Anforderungen und Kosten ist Anwendern schwer zu vermitteln.*

  Eine sehr wichtige Facette des im vorherigen Punkt skizzierten Problems ist, dass sowohl den Anwendern als auch der Unternehmensleitung bzw. dem Auftraggeber oft nicht klar ist, welche Aufwände sich hinter Zwischenergebnissen einerseits und der Berücksichtigung neuer Anforderun-

gen andererseits verbergen. So kann es sein, dass für eine Änderung, die dem Anwender als sehr schwerwiegend erscheint, nur ein relativ geringer technischer Aufwand erforderlich ist, während eine andere neue Anforderung, die dem Anwender sehr einfach erscheint, technisch einen hohen Aufwand bedingt.

- *Unteilbarkeit der Arbeit bei der Softwareentwicklung.*

  Bei verschiedenen Projektarten kann man einen Verzug in der Bearbeitung durch den Einsatz zusätzlichen Personals ausgleichen. Die Arbeit, die bisher zwei Personen gemacht haben, machen nun 6 Personen in einem Drittel der Zeit. Bei Softwareprojekten ist genau dies nicht möglich. Der Einarbeitungsaufwand für die zusätzlichen Arbeitskräfte in das konkrete Projekt ist so hoch, dass kurzfristig sogar eine weitere Verzögerung des Projekts eintritt, weil die bereits eingearbeiteten Mitarbeiter einen Teil ihrer Arbeitszeit für die Einarbeitung der neuen Mitarbeiter opfern müssen. Vorteile sind so nur mittel- oder langfristig zu erwarten.

  Die Arbeit, die ein Einzelner im Projekt erledigt, ist damit zumindest kurzfristig unteilbar. Dies führt dazu, dass der massive Einsatz zusätzlichen Personals zur „Rettung“ eines in Verzug geratenen Softwareprojekts genau das Gegenteil des erhofften Effekts bewirkt.

- *Softwareprojekte sind durch einen hohen Grad der Abstraktion bei geringer Normierung gekennzeichnet.*

  Während bei anderen technischen Entwicklungsaufgaben – z.B. im Maschinenbau – auf eine Vielzahl normierter Komponenten zurückgegriffen werden kann, sind diesbezüglich im Bereich der Softwareentwicklung allenfalls Ansätze erkennbar. Die Objektorientierung stellt hier zwar einen Schritt in die richtige Richtung dar. Erforderlich sind aber wohldefinierte Komponenten mit einer einfachen Schnittstelle und einem genau definierten Verhalten. Genau daran fehlt es aber leider häufig bei großen Klassenbibliotheken. Dadurch führt der Komponenteneinsatz nicht zu der erhofften Komplexitätsreduktion – manchmal ist gar das Gegenteil der Fall.

- *Die Anforderungen ändern sich fortlaufend.*

  Während man es bei anderen Projekten oft mit im Wesentlichen über die Projektlaufzeit konstanten Anforderungen zu tun hat, ist bei Softwareprojekten ein sich während der Projektlaufzeit veränderndes Anforderungsprofil die Regel. Die Ursachen hierfür liegen zum einen in der schnellen technischen Entwicklung, die dazu führt, dass insbesondere bei Projekten mit mehrjähriger Laufzeit die technologischen Möglichkeiten während der Laufzeit zum Teil deutliche Fortschritte machen. Ein wesentlich wichtigerer Aspekt ist aber, dass sich durch die Beschäftigung mit dem Projekt so-

wohl beim Projektteam als auch bei den Anwendern ein ständiger Lernprozess einstellt. So erkennen die Anwender vielleicht erst im Laufe der Zeit, dass diese oder jene Funktionalität, an die man am Anfang – als die Gedanken noch im Ist-Zustand „gefangen“ waren – noch nicht gedacht hat, nun doch möglich und wünschenswert erscheint.

Gemeinsamkeiten mit anderen Projektarten

Obwohl es somit zahlreiche spezifische Aspekte bei Softwareprojekten gibt, existieren aber auch zahlreiche Gemeinsamkeiten mit anderen Projektarten. So sind ein enges Budget und knappe Termine keineswegs eine spezifische Eigenschaft von Softwareprojekten. Auch die Notwendigkeit einer effizienten Qualitätssicherung ist allgemein gegeben. Man spricht dabei von einem magischen Dreieck, in dem Projekte abgewickelt werden müssen. Dieses Dreieck wird in Abbildung 4 verdeutlicht.

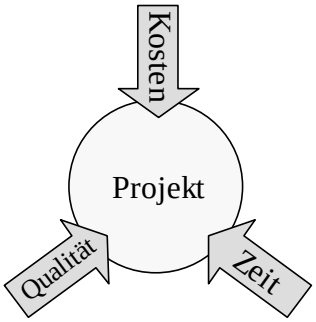


Abbildung 4: Spannungsfeld eines Projekts

Feyhl und Feyhl /Feyhl, Feyhl 1996/ identifizieren als weitere Gemeinsamkeiten mit anderen Projektarten die Notwendigkeit, Ziele zu formulieren, sowie die Notwendigkeit der Planung, der Kontrolle und des Reporting.

Ein Kurs, der sich mit dem Management von Softwareprojekten beschäftigt, muss daher zwangsweise allgemeine Aspekte des Projektmanagements (Planungsaktivitäten, Reportingtechniken, …) und spezifische Aspekte des Managements von Softwareprojekten (Verfahren zur Aufwandsschätzung, Vorgehensmodelle, …) behandeln.

1.4 Ziele des Managements von Softwareprojekten

grundlegende Ziele

Die grundlegenden Ziele beim Management von Softwareprojekten lassen sich aus dem in Abbildung 4 dargestellten Spannungsfeld eines Projekts ableiten:

- Termine einhalten
- Kostenrahmen einhalten
- erforderliche Qualität sicherstellen

realistische Vorgaben

Damit diese Ziele erreichbar sind, muss erfolgreiches Projektmanagement aber schon früher ansetzen, nämlich bei der Erarbeitung realistischer Termin- und Kostenvorgaben und bei der Festsetzung sinnvoller Qualitätsziele. Denn die Termin-

und Kostenvorgaben können offensichtlich auch bei einem „optimalen“ Projektmanagement nur dann eingehalten werden, wenn sie in Anbetracht des zu erstellenden Funktionsumfangs und der Rahmenbedingungen realistisch sind.

Kraus und Westermann (/Kraus, Westermann 2010/, S.23) geben daher die folgenden konkreteren Ziele für das Projektmanagement an:

- *„Verbesserung der fachbereichsübergreifenden Zusammenarbeit – bessere Nutzung der Energien für die Projektbearbeitung, die sonst durch Reibungsverluste in der Kommunikation verloren gehen;*
- *Ausrichtung der Fachbereichsressourcen auf die Unternehmensziele;*
- *Verbesserung der Effizienz, Transparenz und Koordination aller Projekte (Multiprojektmanagement);*
- *Einhaltung der Ziele in den einzelnen Projekten bezüglich Ergebnis, Termine, Aufwand und Produktivität;*
- *Durchsetzung/Umsetzung der Planung.“*

Transparenz

Besonders wichtig erscheint der Aspekt der Transparenz[4]. In Abschnitt 1.3.3 hatten wir ausgeführt, dass bei Softwareprojekten aufgrund des immateriellen Charakters des Produkts der Projektfortschritt nur schwer zu ermitteln ist und Qualitätssicherungsmaßnahmen erschwert werden. Da zu den Aufgaben des Projektmanagements die Planung und Kontrolle des Projektfortschritts zählen (vgl. hierzu Abbildung 1 auf Seite 5), muss eine der wichtigsten Zielsetzungen in der Transparenz des Projekts liegen. Dazu muss klar sein, welche Aktivitäten gerade in Arbeit sind und wie ihr jeweiliger Status einzuschätzen ist.

Zur Erreichung einer hohen Transparenz können u.a. folgende Maßnahmen dienen: (1) der Einsatz eines wohldefinierten Vorgehensmodells für die Softwareentwicklung, (2) der Einsatz von Metriken zur Messung des Fortschritts und der Qualität in den einzelnen Teilaktivitäten und (3) die Einführung eines sinnvollen Berichtswesens. Auf diese Maßnahmen werden wir im weiteren Verlauf des Kurses noch ausführlich eingehen.

Steigerung der Produktivität

Neben der Gewährleistung eines transparenten Projektablaufs stellt die Erhöhung der Produktivität (Effektivität) des Projektteams – bzw. die Gewährleistung einer hohen Produktivität – ein weiteres wesentliches Ziel des Projektmanagements dar. Die Produktivität ist dabei ein sinnvolles Ziel, weil sie die erzielte Leistung zum Aufwand in Beziehung setzt:

4 Der Begriff der Transparenz wird hier in dem Sinne verwendet, dass die „Hülle“ des Projekts transparent (≈ durchsichtig) wird und somit die inneren Vorgänge des Projekts sichtbar und verständlich werden.

$$\text{Produktivität} = \frac{\text{produzierte Ergebnisse}}{\text{eingesetzter Aufwand}}$$

Um die Produktivität im Rahmen der Softwareentwicklung fassbar zu machen, sind entsprechende Kennzahlen erforderlich. Sehr beliebt ist hier die Produktivität in *Lines of Code*. Dabei wird die Anzahl der erstellten Programmzeilen zum eingesetzten Aufwand in Mitarbeiter- oder Personenmonaten ins Verhältnis gesetzt. Die Aussagekraft dieser Kennzahl sollte aber auf keinen Fall überbewertet werden, weil der Programmumfang in Programmzeilen nur ein Teilaspekt bei der Betrachtung des produzierten Ergebnisses ist. Ferner ist auch der Personaleinsatz nur ein Teil der Kostenseite des Projekts. Abbildung 5 gibt einen Überblick über die Faktoren, die den Wert des erstellten Produkts und die entstehenden Kosten beeinflussen.

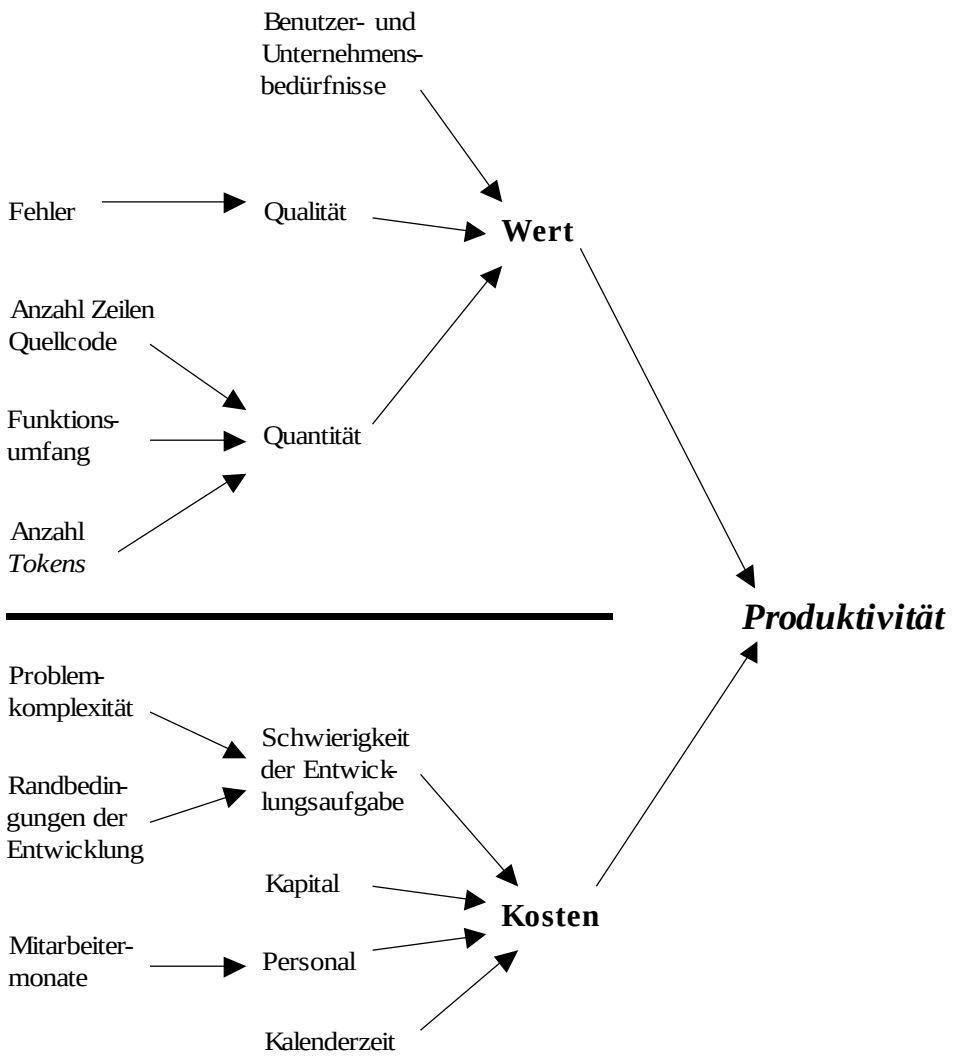


Abbildung 5: Produktivität und ihre Einflussfaktoren nach Basili (zitiert nach /Balzert 1998/, S. 11)

Wenn man also die Produktivität durch ein gutes Projektmanagement steigern möchte, dann muss dieses Ziel in konkretere Teilziele umgesetzt werden.

Von besonderer Bedeutung ist dabei die Mitarbeitermotivation. Zufriedene Mitarbeiter sollten ebenso ein Ziel des Projektmanagements sein wie eine zufriedene Unternehmensleitung/Auftraggeber und ein zufriedener Anwender/Benutzer. Hierzu ist im Hinblick auf die Mitarbeiter schon bei der Zusammenstellung des Projektteams auf die richtige Qualifikation und auf die „Chemie im Team“ zu achten. Ferner sind ein sinnvoller Mitarbeitereinsatz und ein offener Kommunikationsfluss im Team und darüber hinaus wichtig. Wir werden diese Aspekte in Kapitel 7 ausführlich behandeln.

Ziel: zufriedene, motivierte Mitarbeiter

Abschließend sei zum Bereich der Produktivität noch auf die in Abbildung 6 angegebenen Maßnahmen zur Steigerung der Produktivität verwiesen. Aufgabe des Projektmanagements ist es, diese und andere Maßnahmen den konkreten Verhältnissen im Projekt entsprechend gezielt einzusetzen. Wir werden die meisten der in der Abbildung nur stichwortartig angegebenen Aspekte im weiteren Verlauf des Kurses im Detail diskutieren.

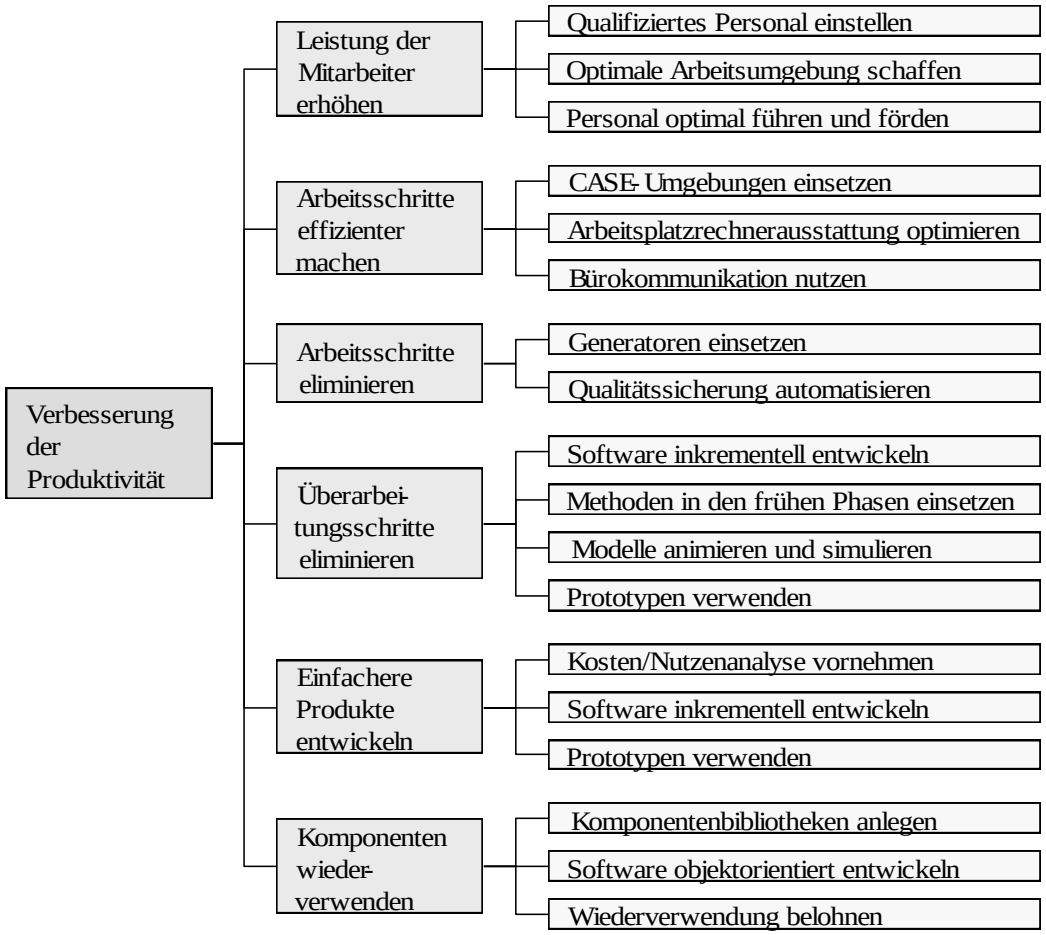


Abbildung 6: Maßnahmen zur Verbesserung der Produktivität (nach /Balzert 1998/, S. 23)

Es zeigt sich damit, dass sowohl die Ziele des Projektmanagements als auch die Maßnahmen zu ihrer Erreichung sehr vielgestaltig sind. Hieraus ergibt sich ein breites Spektrum von Teilaufgaben für das Management von Softwareprojekten.

1.5 Teilaufgaben des Managements von Softwareprojekten

Nachdem wir in Abschnitt 1.4 die Ziele des Projektmanagements und teilweise auch schon Ansätze zu ihrer Erreichung angesprochen haben, wollen wir nun versuchen, die sich aus den Zielen und dem Umfeld ergebenden Aufgaben für das Projektmanagement zu konkretisieren.

Umfeld eines Projekts

Zunächst ist es zum Verständnis der Aufgaben wichtig, sich die wesentlichen Projektbeteiligten in Erinnerung zu rufen. Abbildung 7 verdeutlicht dieses Umfeld.

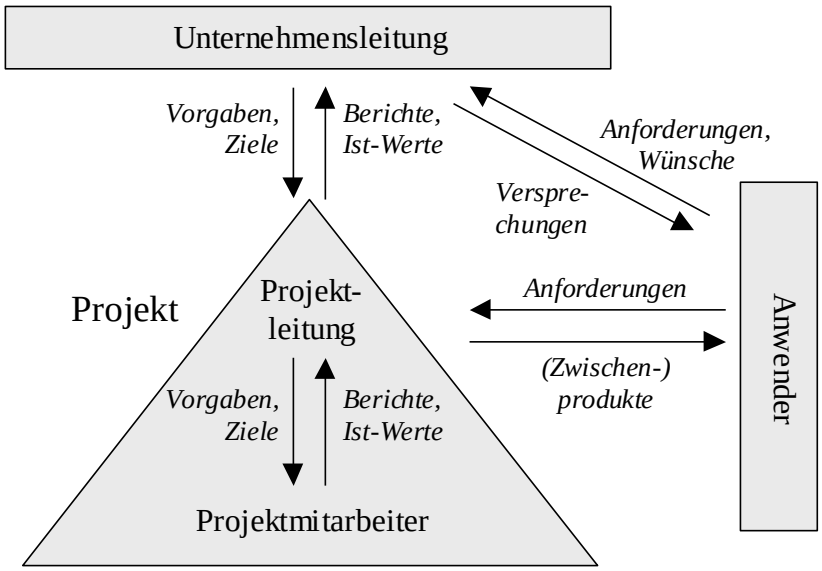


Abbildung 7: Umfeld eines Projekts

Information und Kommunikation

Die Projektleitung regelt das Zusammenspiel der beteiligten Interessengruppen. Aus der Sicht der Projektleitung ist hinsichtlich der Unternehmensleitung und der Anwender die *Information* und die *Kommunikation* in beiden Richtungen besonders vordringlich. Im Rahmen der Projektleitung sind hierfür die organisatorischen Voraussetzungen zu schaffen. Dazu kann ein abgestimmtes Berichtswesen ebenso dienen wie regelmäßige Besprechungen und Projektpräsentationen.

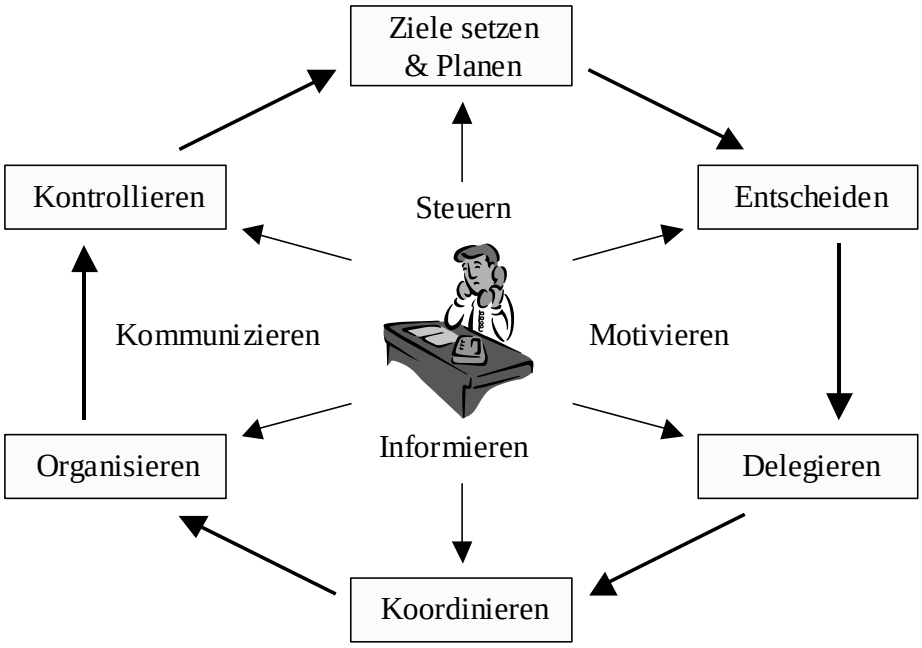


Abbildung 8: Ständige Managementaufgaben eines Projektleiters (nach /Feyhl, Feyhl, 1996/, S. 4)

Motivation und Steuerung

Gegenüber den Projektmitarbeitern tritt neben die Information und die Kommunikation in stärkerem Maße auch die *Motivation* und die *Steuerung*. Abbildung 8 versucht, die sich hieraus ergebenden Aufgaben zusammenzufassen. Die sechs im äußeren „Ring" angeordneten Aufgaben können dabei gewissermaßen als natürliche Abfolge angesehen werden, während die Aufgaben *Steuern*, *Motivieren*, *Informieren* und *Kommunizieren* stärker übergreifenden Charakter haben.

Wir wollen nun kurz auf die einzelnen in Abbildung 8 angeführten Aufgabenbereiche eingehen.

- *Ziele setzen & Planen*

  „Wo kein Ziel ist, ist auch kein Weg!"

  Zielgerichtet arbeiten kann nur der, der seine Ziele kennt. *„Wo kein Ziel ist, ist auch kein Weg!"* /Feyhl, Feyhl 1996/. Wenn wir also zur Steigerung oder Erhaltung der Produktivität in einem Projekt die verfügbaren Ressourcen zielgerichtet einsetzen wollen, dann ist die Voraussetzung, dass die Projektziele klar definiert sind. Die Ziele sollten insbesondere eindeutig, realistisch, widerspruchsfrei und verständlich sein. Auf der Basis dieser Ziele kann dann eine konkrete Planung erfolgen.

  Im vorliegenden Kurs werden wir uns deshalb mit den Aspekten der Zieldefinition und der Ablauf- und Terminplanung – u.a. mit Hilfe der Netzplantechnik – beschäftigen.

- *Entscheiden*

  Projekte sind im Allgemeinen durch eine Vielzahl von Entscheidungssituationen gekennzeichnet. Dies beginnt bei der initialen Entscheidung über

die Projektdurchführung und setzt sich in zahlreichen Detailentscheidungen während der Analyse-, Design- und Implementierungstätigkeiten fort.

typische Fehler

Natürlich ist es dabei nicht sinnvoll, durch übereilte Entscheidungen Spielräume unnötig einzuschränken. Die typischen Fehler, die in Projekten bei Entscheidungen gemacht werden, sind aber eher andere:

1. Entscheidungen werden nicht getroffen. Stattdessen wird ohne eine konkrete Entscheidung weitergearbeitet, was Mehrfacharbeiten und unnötige Unsicherheiten bedingt.

2. Entscheidungen werden unstrukturiert und nicht nachvollziehbar getroffen. Dies führt zu einer mangelnden Akzeptanz und zu ständig wieder aufkommenden Diskussionen über Entscheidungen, die eigentlich längst getroffen worden sind.

sicher und strukturiert entscheiden

Um diesen Problemen entgegenzuwirken, muss sicher und strukturiert entschieden werden. Dazu sollten zumindest in den wichtigeren Entscheidungssituationen Entscheidungstechniken angewendet werden, die den Entscheidungsvorgang strukturieren und die Entscheidung nachvollziehbar machen. Wir werden diese Techniken in Kapitel 3 beschreiben.

- *Delegieren*

Gerade für Projektleiter mit wenig Erfahrung stellt das Delegieren häufig ein großes Problem dar. Sie wollen zu viel selbst machen und demotivieren damit letztlich diejenigen im Projekt, an die die Aufgabe delegiert wurde (oder werden sollte).

Kongruenz zwischen Aufgabenerfüllung und Entscheidungsverantwortung

Wichtig ist dabei, dass das Ziel der Delegation sein sollte, „eine Kongruenz zwischen Aufgabenerfüllung und Entscheidungsverantwortung zu erlangen“ /Corsten, Gössinger 2008/. Oder wie Kraus und Westermann /Kraus, Westermann 2010/ es formulieren: Es gilt „der Grundsatz des Gleichgewichts zwischen:

- Funktion/Aufgaben/Pflichten,
- Verantwortung,
- Macht/Befugnissen/Rechten.“

In der Praxis wird oft der Fehler gemacht, dass zwar Funktionen, Aufgaben oder Pflichten delegiert werden; mit der Delegation von Verantwortung und erst recht mit der Delegation von Macht, Befugnissen und Rechten tut man sich aber wesentlich schwerer.

Delegieren ist damit sowohl eine Frage der Organisation (auf die wir in Kapitel 2 eingehen werden) als auch eine Frage der Mitarbeiterführung (der wir uns in Kapitel 7 widmen werden).

- *Organisieren*

Organisieren bedeutet in erster Linie die Festlegung der Abläufe und der Zuständigkeiten. Dazu wird das Vorgehen bei der Erfüllung bestimmter Funktionen in der *Ablauforganisation* festgelegt, während die Zuständigkeiten und Gliederungen in der *Aufbauorganisation* definiert werden. Daneben beinhaltet das Organisieren in einem Softwareprojekt aber auch eine Vielzahl von kleineren Tätigkeiten, wie das Organisieren von Besprechungen, des Projektarchivs und Ähnliches. Während wir die Aspekte der Ablauforganisation in der zweiten Hälfte dieser Kurseinheit und die Aspekte der Aufbauorganisation in der nächsten Kurseinheit diskutieren, begegnen uns die kleineren Organisationsaktivitäten immer wieder in den verschiedenen Teilen des Kurses.

Ablauforganisation und Aufbauorganisation

- *Koordination*

Das Koordinieren hat stärker als das Organisieren den Aspekt des Abstimmens verschiedener Tätigkeiten aufeinander. Dies kann zum Teil durch Vorgaben im Rahmen der Organisation geschehen. Darüber hinaus bedarf es aber gerade in einem Softwareprojekt immer wieder auch informeller Koordinationsbemühungen, um auf konkrete Entwicklungen zu reagieren. Koordinieren muss daher letztlich durch eine Mischung von formaler Organisation (vgl. hierzu Kapitel 2) und situativer Mitarbeiterführung (vgl. hierzu Kapitel 7) erfolgen.

Abstimmen verschiedener Tätigkeiten

- *Steuern*

Analog zur *Koordination* kann auch die *Steuerung* auf verschiedene Weise erfolgen. So kann es in der einen Situation sinnvoll sein, nur das Ziel vorzugeben und dem Aufgabenträger die konkreten Details zur Erreichung des Ziels selbst zu überlassen, während es in einer anderen Situation sinnvoll sein mag, die konkreten Arbeitsschritte vorzugeben. Die verschiedenen Arten des Steuerns (oder Führens) bezeichnet man dabei auch als Managementstile, auf die wir in Kapitel 7 eingehen werden.

- *Kontrollieren*

Das Kontrollieren ist wohl eine der schwierigsten Aufgaben beim Projektmanagement. Dennoch gilt, dass Planung ohne Kontrolle sinnlos ist. Durch die Kontrolle entsteht erst der in Abbildung 1 auf Seite 5 dargestellte Kreislauf, bei dem die Ergebnisse der Kontrolle bei der Überprüfung und Fortschreibung der Pläne berücksichtigt werden.

Planung ohne Kontrolle ist sinnlos

Das negative Image der Kontrolle rührt wahrscheinlich von dem inhärenten Prüfungscharakter her, der jeder Kontrolle innewohnt. Dabei hat eine konstruktiv verstandene Kontrolle eine Vielzahl positiver Auswirkungen:

- Kontrolle ist für die Beteiligten in jedem Falle eine Möglichkeit der Weiterentwicklung. Ergibt die Kontrolle keine Probleme, so bekommen die für die Erstellung des Kontrollgegenstands Zuständigen eine zusätzliche Absicherung ihrer Arbeitsweise. Werden Defizite erkannt, so ist dies die Möglichkeit, aus diesen Defiziten für die Zukunft zu lernen. Dabei gilt das Sprichwort: „Nur wer nicht arbeitet, macht keine Fehler!“

- Werden durch rechtzeitige Kontrollen Fehlentwicklungen frühzeitig erkannt, so ist der entstandene Schaden im Allgemeinen wesentlich geringer als ohne Kontrolle. Kontrollen sind damit das primäre Mittel der Risikosteuerung, denn Kontrollieren schafft (außer vielleicht auf der psychologischen Ebene) keine Probleme, sondern macht sie nur sichtbar.

- Kontrolle eines Arbeitsergebnisses bedeutet immer auch eine Wertschätzung für das Geleistete. Viel demotivierender als eine Kontrolle, die evtl. ein paar Schwachstellen zu Tage fördert, ist es, wenn Arbeitsergebnisse nicht beachtet werden. Keine Rückmeldung ist – zumindest für einen Mitarbeiter, dessen Grundeinstellung stimmt – wesentlich schlimmer als eine konstruktiv vorgebrachte kritische Würdigung des Arbeitsergebnisses.

Ist man sich dieser positiven Aspekte der Kontrolle bewusst, kann man wesentlich leichter ein konstruktives Kontrollwesen einsetzen. Den damit verbundenen Bereich der Qualitätssicherung und des Qualitätsmanagements werden wir in Kapitel 8 behandeln. Daneben werden wir in Kapitel 7 auf die mit Kontrolltätigkeiten verbundenen Aspekte der Mitarbeiterführung eingehen.

- *Motivieren*

Vielschichtigkeit der Bedürfnisse

Kaum ein anderer Einflussfaktor ist so bedeutend für den Erfolg eines Projekts wie die Motivation der Mitarbeiter. In der Betriebswirtschaftslehre werden daher eine Vielzahl von Motivationstheorien behandelt. Dabei geht man im Wesentlichen davon aus, dass der Mensch durch sein Streben nach der Befriedigung der menschlichen Grundbedürfnisse motiviert wird. Diese Bedürfnisse reichen nach Maslow von den *physiologischen Bedürfnissen* über die *Sicherheitsbedürfnisse*, die *sozialen Bedürfnisse* und das *Bedürfnis nach Wertschätzung* bis zum *Bedürfnis nach Selbstverwirklichung*. Wie die Vielschichtigkeit der Bedürfnisse schon nahe legt, ist eine gute Motivation der Mitarbeiter nicht mit einigen wenigen Maßnahmen zu erreichen. Vielmehr gilt es, die verschiedensten Einflussfaktoren von der Entlohnung über die Arbeitszeit, das Arbeitsumfeld, die Arbeitsaufgabe etc. mit Umsicht und Einfühlungsvermögen zu gestalten. Wir werden uns dem Aspekt der Motivation in Kapitel 7 ausführlich widmen.

- *Informieren und Kommunizieren*

  Viele Projekte scheitern an einem Defizit von Information und Kommunikation. Fehlt die Information z.B. darüber, welche Gründe zu einer Entscheidung geführt haben, so fördert dies das Misstrauen, und in dem daraus resultierenden Arbeitsklima ist dann oft kein produktives, gemeinsames Arbeiten mehr möglich.

  zielgerichtetes Berichtswesen

  Eine notwendige Voraussetzung für eine gute Informations- und Kommunikationsstrategie ist ein zielgerichtetes Berichtswesen. Dabei ist zu berücksichtigen, dass die Kunst eines guten Berichtswesens darin besteht, die notwendigen Informationen in kompakter und übersichtlicher Form bereitzustellen. Es hilft nichts, alle Projektmitarbeiter mit allen Details zu versorgen. Dann würden die einzelnen Mitarbeiter zu viel Zeit darauf verwenden müssen, aus der Informationsflut das für sie Wesentliche zu extrahieren.

  informelle Kommunikation

  Neben dem Berichtswesen sind aber im Hinblick auf die Information und Kommunikation im Projekt und zwischen dem Projekt und anderen Stellen auch die informellen Kontakte von großer Bedeutung. Dies kann sich sowohl positiv als auch negativ auswirken. Positiv kann dabei sein, dass eine informelle Kommunikation auf kurzen Wegen oft wesentlich schneller und flexibler ist als die Kommunikation auf dem „Dienstweg“. Informelle Beziehungen sind deshalb eine unverzichtbare Ergänzung zu den von der Organisation vorgegebenen Wegen. Negativ kann sich dies aber auch in mehrerer Hinsicht auswirken. So kann der Eindruck, dass der Projektleiter mit Mitarbeiter X bereits vorab immer einige Dinge informell bespricht, bevor die anderen davon erfahren, zu Neid und Streit im Projekt führen. Ein anderes Problem stellt häufig die mangelnde Nachvollziehbarkeit informeller Kontakte dar. Wenn so die Transparenz des Informationsflusses leidet, können ggf. später die Gründe z.B. für Entscheidungen nicht mehr nachvollzogen werden.

  Wir werden deshalb Strategien zur Information und Kommunikation noch ausführlich betrachten.

Die genannten Teilaufgaben des Projektmanagements machen deutlich, dass die Durchführung von Softwareprojekten nur zum Teil eine technische Herausforderung darstellt. Mindestens ebenso groß sind beim Management von Softwareprojekten die Herausforderungen im Bereich der Menschenführung. Ein erfolgreiches Projektmanagement setzt damit neben dem Verständnis für den Problembereich und „handwerklichen“ Fertigkeiten im Bereich der Softwareentwicklung auch organisatorische und psychologische Fertigkeiten voraus. Dies verdeutlicht das in Abbildung 9 gegebene Schaubild.

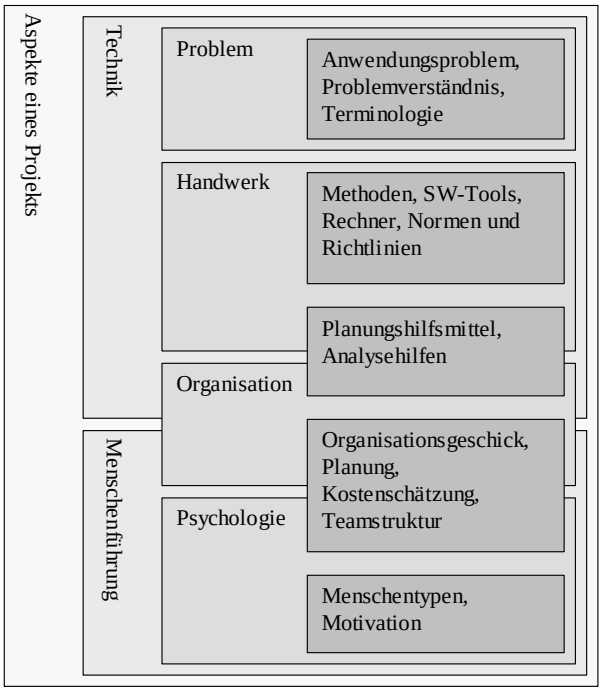


Abbildung 9: Überblick über wesentliche Aspekte eines Projekts (nach /Elzer 1994/, S. 5)

Nach diesem Überblick über die Gesamtheit der Aufgaben des Projektmanagements wollen wir uns nun dem ersten großen Themenbereich im Detail widmen, den Vorgehensmodellen für die Softwareentwicklung.

1.6 Vorgehensmodelle für Softwareprojekte

historische Entwicklung

Historisch gesehen begann die Entwicklung von Software in den 50er Jahren des 20. Jhs. in einem sehr überschaubaren Rahmen. Typischerweise wurde ein Programm von einer Person entwickelt, die gleichzeitig auch Anwender des Programms war. Aufgrund der beschränkten Leistungsfähigkeit der Rechner waren die Programme nicht sehr umfangreich und konnten in überschaubarer Zeit entwickelt werden. Ein explizites Projektmanagement war daher nicht erforderlich.

Mit zunehmender Leistungsfähigkeit der Rechner wurden in den 60er Jahren immer umfangreichere Programme entwickelt. Dabei kam es immer öfter zu Fehlschlägen. Dies hatte in der Hauptsache folgende Ursachen:

Entwicklung im Team

1. Da Software aufgrund des gewachsenen Umfangs nun in Teams entwickelt werden musste, war eine Aufteilung der Arbeiten auf mehrere Personen erforderlich. Weil Softwareentwicklung bis dahin aber immer monolithisch betrachtet worden war, fehlten die sauber strukturierten Teilaufgaben, die man auf einzelne Teammitglieder hätte übertragen können.

Projektfortschritt unklar

2. Es war – ebenfalls bedingt durch die monolithische Betrachtungsweise – nicht möglich, während des Projektablaufs fundierte Aussagen über den

Projektfortschritt zu machen. Viele Projekte überschritten daher den geschätzten Zeit- und Aufwandsrahmen bei Weitem.

Um Softwareprojekte plan- und kontrollierbar zu machen, strebte man nach einer Strukturierung von Softwareprojekten. Das Ziel war ein „ingenieurmäßiges“ Vorgehen bei der Softwareentwicklung, worunter man die Aufteilung des Projekts in wohldefinierte Teilaktivitäten verstand, die mit vorgegebenen Methoden bearbeitet werden sollten. In diesem Zusammenhang kam auch der Begriff des „Software Engineering“ auf.

Softwaretechnik

Software Engineering oder – um den deutschen Begriff zu verwenden – die Softwaretechnik beschäftigt sich somit sowohl mit der Strukturierung des Entwicklungsprozesses als auch mit den in den einzelnen Phasen vorkommenden Aktivitäten und den dabei einzusetzenden Methoden. Für das Management von Softwareprojekten ist vor allem die Strukturierung wichtig, weil sie die Ablauforganisation[5] eines Softwareprojekts regelt. Hinsichtlich der konkreten Methoden, die innerhalb der einzelnen Teilaktivitäten eingesetzt werden können, sei hier auf die Literatur zum Software Engineering verwiesen (vgl. z.B. /Balzert 1996/; /Ludewig 2010/; /Sommerville 2012/). Die Techniken des Projektmanagements, die wir in diesem Kurs behandeln, sind weitestgehend methodenneutral, d.h. sie können im Wesentlichen unabhängig von den in den einzelnen Teilaktivitäten eingesetzten Methoden angewendet werden.

Vorgehensmodelle

Vorgehensmodelle dienen somit der Strukturierung des Ablaufs von Softwareprojekten. Ein einfaches Beispiel für ein Vorgehensmodell gibt Abbildung 10. Die Softwareentwicklung ist hier in vier Phasen aufgeteilt, die schließlich im Einsatz der Software münden. Dabei wird idealtypisch davon ausgegangen, dass in jeder Phase ein Ergebnis erarbeitet wird, das als Ausgangspunkt für die folgende Phase dient. So wird in der Phase „Analyse und Definition“ die fachliche Produktdefinition erarbeitet, die den Ausgangspunkt für den technischen Entwurf des Systems bildet. Dieser ist wiederum die Basis für die Implementierung, die schließlich getestet wird.

5 Unter *Ablauforganisation* versteht Wöhe /Wöhe, Döring 2010/ „die Ordnung von Handlungsvorgängen (Arbeitsprozessen)“. In der Betriebswirtschaftslehre wird die *Ablauforganisation* von der *Aufbauorganisation* unterschieden, die sich „auf die Verknüpfung der organisatorischen Grundelemente (Stelle, Instanz und Abteilung) zu einer organisatorischen Struktur“ bezieht. Wir werden auf die betriebswirtschaftliche Definition dieser Begriffe in Kapitel 2 bei der Betrachtung der aufbauorganisatorischen Einordnung und Gestaltung von Projekten ausführlicher eingehen.

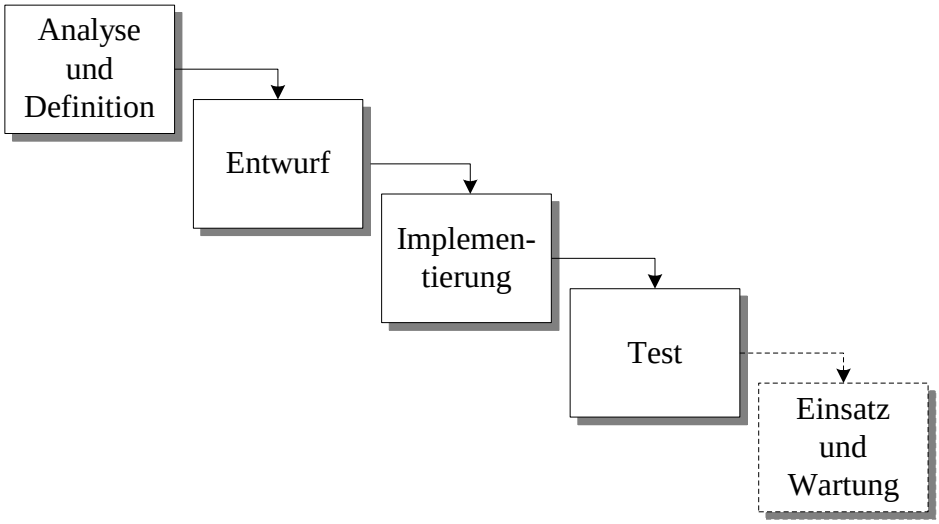


Abbildung 10: Ein einfaches Vorgehensmodell (ablauforientierte Darstellung)

Bevor wir die gebräuchlichsten Vorgehensmodelle vorstellen, erscheint es sinnvoll, zunächst einige generelle Aspekte von Vorgehensmodellen anzusprechen.

1.6.1 Grundidee der Vorgehensmodelle

hierarchische Gliederung der Gesamtaufgabe

Vorgehensmodelle verfolgen das Ziel einer hierarchischen Gliederung der Gesamtaufgabe, die ein- oder mehrstufig sein kann. Die Darstellung dieser Gliederung kann sowohl zerlegungsorientiert – als Baum oder Hierarchie – als auch ablauforientiert – als Graph – erfolgen. Während Abbildung 10 eine ablauforientierte Darstellung zeigt, ist das gleiche Vorgehensmodell in Abbildung 11 zerlegungsorientiert dargestellt.

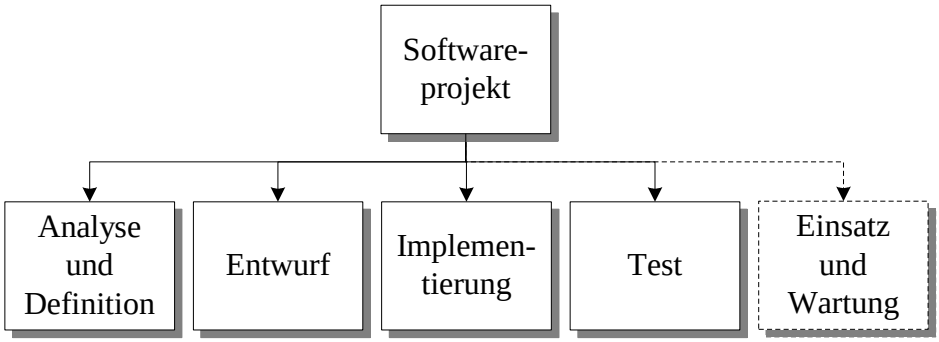


Abbildung 11: Ein einfaches Vorgehensmodell (zerlegungsorientierte Darstellung)

Die Knoten stellen jeweils die Teilphasen des Vorgehensmodells dar. In der ablauforientierten Sicht sind die Kanten als Abhängigkeiten zu interpretieren. Wichtig ist, dass die einzelnen Phasen und ihre Abhängigkeiten möglichst genau beschrieben werden sollten.

Begriffe *Phase*, *Aktivität* und *Aufgabe*

Zunächst ist an dieser Stelle aber unser Begriffsverständnis für die Begriffe *Phase*, *Aktivität* und *Aufgabe* zu klären. Die Begriffe *Aktivität* und *Aufgabe* betrachten in gewisser Weise zwei Aspekte der gleichen Sache. Während unter dem Begriff der *Aufgabe* verstanden wird, „Was zu tun“ ist, verbindet man mit dem Begriff der *Aktivität* die Erfüllung oder Ausführung einer Aufgabe. Ein Softwareprojekt

selbst kann damit als Aufgabe und seine Durchführung als Aktivität betrachtet werden, die in Teilaufgaben bzw. Teilaktivitäten zerlegt werden können. Von einer *Phase* sprechen wir typischerweise auf der obersten Zerlegungsebene, wenn von einer sequenziellen Abfolge der Aktivitäten ausgegangen wird. Phasen sind in diesem Sinne spezielle Aktivitäten, die durch eine sequenzielle Abfolge gekennzeichnet sind.

genaue Definition einer Teilaufgabe

Abbildung 12 zeigt nun, welche Aspekte bei der genauen Definition einer (Teil-)Aufgabe in einem Vorgehensmodell nach dem Standardwerk von Humphrey /Humphrey 1990/ berücksichtigt werden sollten.

Bemerkenswert sind dabei folgende Punkte:

Standardisierung

1. Die Beschreibung der Teilaufgabe sollte auch Aussagen darüber machen, *wie* die Teilaufgabe bearbeitet werden soll. Dies umfasst insbesondere Vorgaben für die einzusetzenden Methoden. Dadurch wird unter anderem die spätere Nachvollziehbarkeit gefördert. Wird in einem Unternehmen in allen Projekten nach den gleichen Standards und Methoden gearbeitet, können sich Mitarbeiter, die später zum Projektteam hinzukommen oder die in der Wartungsphase Arbeitsergebnisse nachvollziehen müssen, um Anpassungen vorzunehmen, wesentlich leichter einarbeiten.

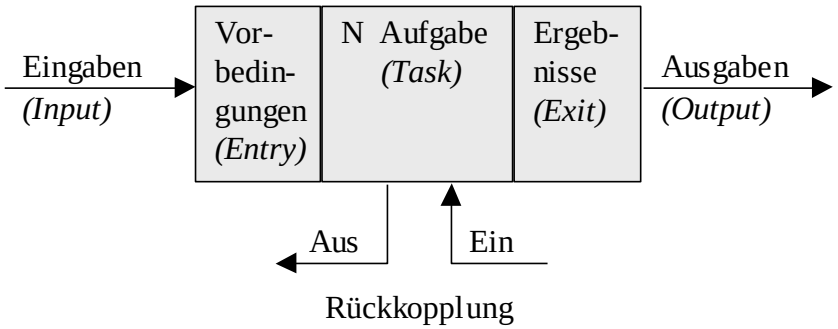
Zeit- und Ressourcenbedarf

2. Bezüglich der Aufgaben sollten Vorgaben für den Zeit- und den Ressourcenbedarf (insbesondere benötigte Mitarbeiter) sowie z.B. für den Umfang und die Qualität der Ergebnisse gemacht werden.

   Dabei muss zwischen dem abstrakten Vorgehensmodell, das auf alle Projekte angewendet werden soll, und der konkreten Instanz, d.h. dem für ein konkretes Projekt spezialisierten Vorgehensmodell, unterschieden werden. Während im abstrakten Vorgehensmodell nur Hinweise zur Bedarfsermittlung und ggf. Schätzverfahren angegeben werden können, sind auf der Instanzebene konkrete Zahlen erforderlich, die sich ggf. aus der Anwendung der Schätzverfahren ergeben.

Rückkopplung

3. Die vorgesehene Möglichkeit der Rückkopplung ermöglicht auch Abläufe, die nicht streng „wasserfallartig“ verlaufen, sondern Schleifen enthalten. Derartige Rückkopplungen erschweren natürlich die Planung eines Projekts. Durch mögliche Rückkopplungen können insbesondere wesentlich schwerer Aussagen über den Projektfortschritt gemacht werden, weil auch nach Ausführung einer Aufgabe durch Rückkopplungen für diese Aufgabe erneut Aufwände anfallen können. Auf der anderen Seite sind Rückkopplungen aber in der Praxis oft nicht zu vermeiden, und es wäre deshalb unrealistisch, sie in den Modellen nicht zu berücksichtigen.



Spezifikationen:

Vorbedingungen:	Die Bedingungen, die vor Durchführung der Aufgabe erfüllt sein müssen.
Ergebnisse:	Die Resultate, die erzeugt werden, und wie sie aussehen.
Rückkopplung:	
Ein:	Jede Rückkopplung von einer anderen Aufgabe.
Aus:	Jede Rückkopplung zu anderen Aufgaben.
Aufgabe:	Was ist zu tun, durch wen, wie und wann, einschließlich entsprechender Standards, Verfahren und Verantwortlichkeiten.
Maße: *(measurements)*	Die geforderten Aufgabenmaße (Aktivitäten, Ressourcen, Zeit), Ergebnisse (Anzahl, Größe, Qualität) und Rückkopplungen (Anzahl, Größe, Qualität).

Abbildung 12: Vorlage zur Spezifikation einer (Teil-)Aufgabe (nach /Humphrey 1990/, S. 257ff.)

genaue Definition einer Teilaufgabe

Zusammenfassend ist bei der Definition einer Aufgabe innerhalb eines Vorgehensmodells darauf zu achten, dass zumindest folgende Fragen geklärt sind:

- Worauf baut die Aufgabe auf?
- Was soll bei der Bearbeitung der Aufgabe untersucht und/oder entwickelt werden?
- Wie soll dies geschehen (Methoden)?
- Welche Ergebnisse sind zu erarbeiten?

Anwendung eines Vorgehensmodells

Bei der Anwendung eines Vorgehensmodells in einem Projekt sind dann zu den jeweiligen Aufgaben konkrete Vorgaben hinsichtlich des Aufwands, der einzusetzenden Verfahren etc. zu machen. Feyhl und Feyhl schlagen hierzu eine Reihe sogenannter W-Fragen vor, die in Tabelle 2 aufgelistet sind.

Wer ?	Verantwortung
Was ?	Aufgabe
Warum ?	Begründung und Ziel
Wann ?	Zeitpunkt
Wo ?	Ort, Funktion oder Produkt
Wie ?	Art und Weise, Ablauf
Womit ?	Arbeitsmittel
Wonach ?	Methoden, Normen und Standards
Wofür ?	Zielgruppe

Tabelle 2: Fragen, die vor der Bearbeitung einer Aufgabe aus dem Vorgehensmodell geklärt sein sollten (nach /Feyhl, Feyhl 1996/, S.25)

Bevor wir in den nachfolgenden Abschnitten einige der wichtigsten Vorgehensmodelle betrachten, wollen wir noch darauf hinweisen, dass Vorgehensmodelle innerhalb des Managements von Softwareprojekten nicht isoliert betrachtet werden können. Es gibt vielmehr eine Reihe von Zusammenhängen zu anderen Aufgaben, die nicht direkt zum Bereich der Ablauforganisation gehören.

1.6.2 Einordnung von Vorgehensmodellen

Vorgehensmodelle dienen zunächst der allgemeinen Ablaufmodellierung. Man möchte durch sie einen genaueren Überblick über das Gesamtprojekt gewinnen und die Plan- und Kontrollierbarkeit fördern. Letztlich dienen Vorgehensmodelle damit dem Zweck, Projekte einem sinnvollen Management zugänglich zu machen.

Prozessmodelle / Workflow-Managementsysteme

Wenn der Ablauf eines Projekts durch das Vorgehensmodell recht genau vorgegeben ist, dann liegt der Gedanke nahe, die Softwareentwicklung mit einem Workflow-Managementsystem zu unterstützen. Dazu werden den Projektmitarbeitern Qualifikationsprofile zugeordnet. Ferner wird definiert, welche Qualifikationen zur Bearbeitung welcher Aufgabe erforderlich sind. Das Workflow-System prüft, für welche Aufgaben aus dem Vorgehensmodell die nötigen Vorarbeiten abgeschlossen sind. Anschließend wird für diese Aufgaben nach den Qualifikationsprofilen ermittelt, wer sie ausführen könnte. Das System nimmt somit eine Zuordnung der anstehenden Aufgaben zu den Projektmitarbeitern vor, die z.B. so realisiert werden kann, dass jeder Miterbeiter eine eigene „To-Do-List“ hat, in der ihm die Aufgaben angezeigt werden, die er jetzt bearbeiten könnte, weil ihre Voraussetzungen erfüllt sind und seine Qualifikation passt.

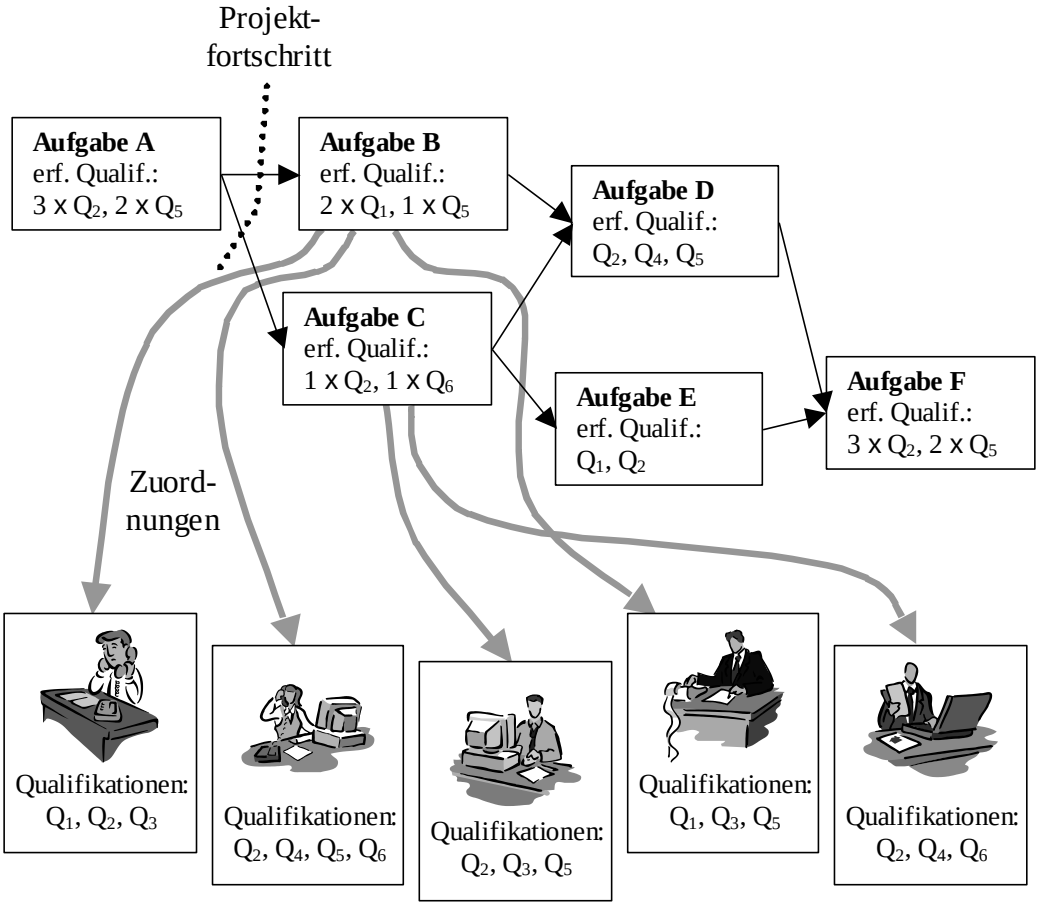


Abbildung 13: Zuordnung der Aufgaben zu Mitarbeitern aufgrund von Qualifikationsprofilen

In das Vorgehensmodell können hier auch automatisch ausführbare Aufgaben wie Übersetzungsvorgänge oder Ähnliches aufgenommen werden. Das Ziel ist eine rechnergestützte Abwicklung des Entwicklungsprozesses. In diesem Kontext wird für die hierzu erforderlichen feinschrittigen Vorgehensmodelle der Begriff *Prozessmodelle* verwendet /Derniame et al. 1999/.

Netzplantechnik

Die Netzplantechnik ist ein weiterer Bereich, der im Zusammenhang mit Vorgehensmodellen zu sehen ist. Diese dient, wie es Müller-Merbach formuliert, „der Terminplanung solcher Projekte, die aus vielen zeitlich voneinander abhängenden Einzelvorgängen bestehen. Der große Wert der Netzplantechnik liegt darin, dass man auch bei sehr umfangreichen Projekten stets einen Überblick über das gesamte Projekt behalten kann“ (/Müller-Merbach 1998/, S. 181).

Die Netzplantechnik basiert auf der Annahme, dass das Projekt in einzelne Aufgaben (in diesem Kontext meist als *Aktivitäten* bezeichnet) aufgeteilt werden kann, für die Inhalt, Dauer, benötigte Ressourcen und gegenseitige Abhängigkeiten sehr genau vorgegeben werden können. Auf der Basis dieser Aktivitäten und ihrer Abhängigkeiten wird dann eine Planung für das Projekt erstellt, die für die einzelnen Aktivitäten die Anfangs- und die Endzeitpunkte sowie die einzusetzenden Ressourcen bestimmt. In diesem Zusammenhang wird auch ermittelt, welche Aktivitäten für die Erreichung des frühesten möglichen Endtermins kritisch sind und welche durchaus einen gewissen Spielraum hinsichtlich des Anfangstermins lassen.

Von der Idee her liegen Vorgehensmodelle für die Softwareentwicklung und Netzpläne somit nah beieinander. Der Hauptunterschied liegt in zwei Bereichen:

- *in der Granularität der Aktivitäten*

  Das Vorgehensmodell beschreibt den Ablauf eines Softwareprojekts auf einer höheren Ebene als ein Netzplan, der zur feinschrittigen Ablaufplanung eingesetzt wird. Man kann – und sollte – ein Vorgehensmodell im Verhältnis zum Netzplan daher als Vorstrukturierung verstehen, die für jedes konkrete Projekt spezifisch weiter strukturiert werden muss. Wir erhalten damit einen hierarchischen Netzplan, dessen oberste Ebene durch das Vorgehensmodell definiert wird.

- *in der Bestimmtheit*

  Während man bei der Anwendung der Netzplantechnik davon ausgeht, dass die einzelnen Aktivitäten recht genau bekannt sind, kann dies bei der Anwendung eines Vorgehensmodells allenfalls für die nächsten bevorstehenden Aufgaben der Fall sein. Wenn wir z.B. an das Vorgehensmodell aus Abbildung 10 auf Seite 28 denken, dann können wir sicher am Anfang der Phase *Analyse und Definition* noch nicht konkret sagen, welche Aufgaben in der *Testphase* im Einzelnen auf uns zukommen werden; wir kennen ja zu diesem Zeitpunkt weder die genaue Funktionalität des Systems noch seine Architektur.

In der Praxis wird damit beides benötigt, ein Vorgehensmodell, um den Ablauf eines Projekts auf hoher Ebene zu regeln, und die Netzplantechnik, um die Detailplanung für die aktuell bevorstehenden Aufgaben durchzuführen. Die Kombination kann dabei in einer rollierend[6] verfeinerten Planung bestehen. Auf der Basis des Vorgehensmodells wird jeweils für die unmittelbar bevorstehende Phase eine Feinplanung im Sinne der Netzplantechnik durchgeführt. Abbildung 14 verdeutlicht diesen Zusammenhang. Hier wird davon ausgegangen, dass die Phase *Analyse und Definition* gerade abgeschlossen wurde. Auf dieser Basis kann für die *Entwurfsphase* eine Feinplanung mit Hilfe der Netzplantechnik erfolgen.

6 Unter einer *rollierenden* Planung versteht man allgemein eine Planung, die nur für die direkt bevorstehenden Aufgaben detailliert ausgearbeitet ist. Für die in der weiteren Zukunft liegenden Aufgaben enthält die Planung dagegen nur grobe Vorgaben. Diese Planung wird dann fortlaufend („rollierend“) aktualisiert.

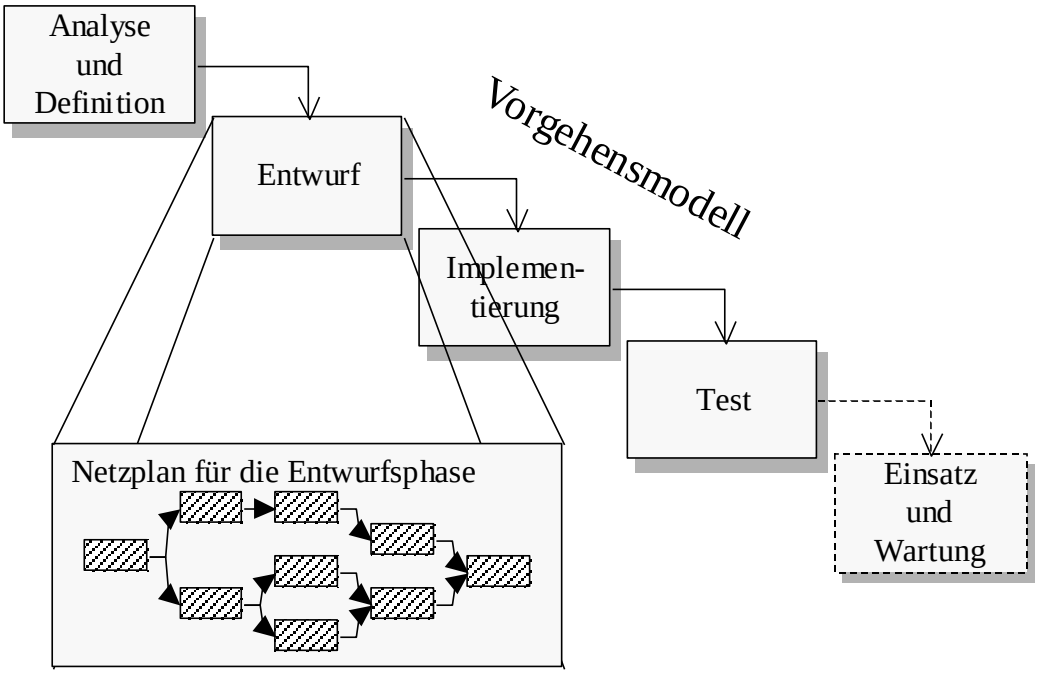


Abbildung 14: Zusammenhang zwischen Vorgehensmodell und Netzplan

Wegen der Bedeutung der Netzplantechnik für die konkrete Planung und Steuerung eines Projekts werden wir in Kapitel 5 ausführlich auf die Verfahren der Netzplantechnik eingehen.

Aufwandsschätzung

Ein wichtiger Bereich der Projektplanung ist die Aufwandsschätzung, auf die wir in Kapitel 4 noch ausführlich eingehen werden. Wir werden dabei sehen, dass zwei einfache Verfahren zur Aufwandsschätzung unmittelbar am Vorgehensmodell ansetzen. Da ist zunächst die *Prozentsatzmethode* zu nennen, bei der man unterstellt, dass die Verteilung des Aufwands auf die verschiedenen Phasen des Vorgehensmodells über alle Projekte hinweg relativ ähnlich ist. Man kann dann aus dem Aufwand für die erste Phase den Aufwand für die weiteren Phasen hochrechnen. Ein anderes Verfahren ist die *Arbeitsschrittschätzung*, die im Sinne der Netzplantechnik das gesamte Projekt hierarchisch recht fein strukturiert und für die einzelnen Teilaktivitäten die Aufwände schätzt.

weitere Zusammenhänge

Neben den geschilderten Zusammenhängen zwischen dem Vorgehensmodell und Prozessmodellen, Netzplänen und einfachen Verfahren der Aufwandsschätzung sollte man auch beachten, dass die einzelnen Phasen des Vorgehensmodells natürliche Gegenstände der *Delegation* und der *Qualitätssicherung* sind. Auch im Hinblick auf eine mögliche *externe Vergabe von Teilaufgaben* kann die Strukturierung durch das Vorgehensmodell als Vorgabe betrachtet werden.

Es ist daher im Folgenden bei der Vorstellung der Vorgehensmodelle wichtig, diese Zusammenhänge im Hinterkopf zu behalten und die Qualität und Anwendbarkeit der Vorgehensmodelle ggf. auch vor diesem Hintergrund zu sehen.

Abschließend sei noch angemerkt, dass die Vorgehensmodelle den Projektablauf in der Regel erst ab dem Zeitpunkt betrachten, wenn die Entscheidung für die Durchführung des Projekts getroffen worden ist. Davor liegen aber in der Praxis noch die Entstehung der Idee zu dem entsprechenden Projekt und der Entscheidungsprozess über die Projektdurchführung. Wir werden diese beiden Aspekte in Kapitel 6 und in Kapitel 3 ausführlich betrachten.

1.6.3 Das klassische Wasserfallmodell

Im klassischen Wasserfallmodell werden die Phasen in einer linearen Abfolge durchlaufen. Rücksprünge und Iterationen sind nicht vorgesehen. Ein Beispiel für ein einfaches Wasserfallmodell haben wir bereits in Abbildung 10 auf Seite 28 betrachtet. In der Literatur existieren zahlreiche Varianten dieses Modells mit im Detail durchaus unterschiedlichen Aufteilungen und Untergliederungen der Aktivitäten.

Im Vorgehensmodell aus Abbildung 10 sind die einzelnen Phasen wie folgt zu interpretieren:

- *Analyse und Definition*

  In der Analyse und Definition werden die spezifischen Produktanforderungen ermittelt. Dazu ist im Allgemeinen zunächst eine Ist-Analyse zu erstellen, auf deren Basis dann eine Schwachstellenermittlung durchgeführt werden kann. Daran schließt sich eine Zieldefinition an, die schließlich zu einer Definition der Produktanforderungen führt. Auf Basis dieser Produktanforderungen sollte eine Durchführbarkeitsstudie erstellt und eine Kosten-/Nutzen-Analyse vorgenommen werden.

  Produktdefinition

  Das Ergebnis der Phase ist die *Produktdefinition*, die auch als Anforderungsdefinition oder Pflichtenheft bezeichnet wird. Diese Produktdefinition kann im Falle einer Fremdvergabe der Erstellung als Vertragsbasis zwischen Auftraggeber und Auftragnehmer dienen. Sie schreibt den Leistungsumfang des Produkts fest und ist damit auch die Basis der Endabnahme. Die Produktdefinition enthält i.A. Angaben im Hinblick auf den Funktionsumfang, die Benutzungsoberfläche, die Schnittstellen zur Systemumgebung, das Leistungsverhalten des Systems, die vorgesehene Hardware und ggf. Systemsoftware, die Dokumentationsrichtlinien und die Terminplanung.

- *Entwurf*

  Software-Architektur

  Im Entwurf wird auf der Basis der Produktdefinition die innere Struktur des Softwaresystems entwickelt. Dazu wird das Gesamtsystem schrittweise in Komponenten zerlegt. Das erste Zwischenergebnis des Entwurfs ist damit die Software-Architektur (auch als Systemdesign bezeichnet).

Entwurfsspezifikation

In einem zweiten Schritt erfolgt eine genauere Spezifikation der Aufgaben der Komponenten und ihres Zusammenwirkens. Es geht dabei allerdings noch nicht um programmtechnische Details, sondern um eine klare Definition der Komponenten. Das Leitthema ist die Bewältigung von Komplexität durch Zerlegung. Dies führt zur Entwurfsspezifikation, die die Software-Architektur verfeinert und konkretisiert. Die Qualität der Entwurfsspezifikation ist dabei von kaum zu überschätzender Bedeutung für die Qualität und Wartbarkeit des Gesamtsystems.

- *Implementierung*

Methoden des Programmierens im Kleinen

Auf Basis der Softwarespezifikation erfolgt nun die Implementierung der Komponenten in einer Programmiersprache. Dabei sind jeweils die Methoden des Programmierens im Kleinen zu beachten. Zu nennen sind hier insbesondere Wohlstrukturiertheit, ein sauberer übersichtlicher Programmierstil und der solide Test der Komponenten. Das Ergebnis der Implementierung sollte damit eine Menge dokumentierter und getesteter Komponenten sein.

- *Test*

Integration der Komponenten

Da in der Implementierung die einzelnen Komponenten bereits getestet wurden, richtet sich das Augenmerk nun auf die Integration der Komponenten und damit auf den Integrationstest, der im Allgemeinen sukzessive durchgeführt wird.

Systemtest

An den Integrationstest, der das Zusammenspiel der einzelnen Komponenten adressiert, schließt sich der Systemtest an, bei dem das System gegen die Produktdefinition getestet wird. Auf den Systemtest folgt die Installation des Systems im Produktionsumfeld und der Abnahmetest. Die Grenzen zwischen System- und Abnahmetest sind dabei durchaus fließend.

- *Einsatz und Wartung*

Nun folgen Einsatz und Wartung des Systems, die natürlich nicht mehr zum Entwicklungsprozess, aber doch zum Lebenszyklus einer Software zu rechnen sind. Im Hinblick auf die Wartung ist dabei ebenso an die Korrektur von Fehlern wie an die Anpassung an andere Systemumgebungen oder Änderungen und Erweiterungen der Funktionalität zu denken.

Eine recht verbreitete Variante der obigen Einteilung fasst Teilaspekte aus den Phasen *Analyse und Definition* zu einer neuen Phase *fachlicher Systementwurf* zusammen. An diese Phase schließt sich dann der *DV-technische Systementwurf* an. Ziel dieser Trennung ist es, zunächst unabhängig von der technischen Realisierung die fachlichen Funktionalitäten des Systems zu definieren und erst im Anschluss daran die technischen Aspekte – insbesondere die System- und Soft-

warearchitektur – zu betrachten. Das in Tabelle 3 dargestellte wasserfallartige Vorgehensmodell verwendet zum Beispiel eine solche Trennung.

Standard-Phasenkonzept für eine Individualentwicklung	
Phase 1:	**Voruntersuchung (Eventualphase)** Grobe Problem- und Bedarfsanalyse Anforderungen an die Lösung
Phase 2:	**Fachliche Grobkonzeption** Problem- und Bedarfsanalyse (beziehungsweise Ergänzung) Erarbeitung eines fachlichen Grobentwurfs (beziehungsweise Verfeinerung)
Phase 3:	**Fachliche Feinkonzeption** Fachlicher Feinentwurf Logische Datenorganisation Planung der Benutzerorganisation
Phase 4:	**Programmvorgabe** Erstellung der Programmstruktur Aufbau der physischen Datenbank
Phase 5:	**DV-technische Realisierung** Erstellung Programmentwurf Codierung und Umwandlung Testarbeiten
Phase 6:	**Benutzerorganisation und Systemeinführung** Zusammenstellung des Benutzerhandbuchs Benutzerschulung und -einweisung Datenbereinigung und -aufbereitung Anpassung der Benutzerorganisation Systemumstellung

Tabelle 3: Standard-Phasenkonzept für eine Individualentwicklung nach /Grupp 1998/, S. 117

Problembereiche des klassischen Wasserfallmodells

Die Verwendung eines Vorgehensmodells, das sich an der skizzierten Aufteilung des Entwicklungsprozesses orientiert, ist unbedingt sinnvoll und notwendig. Wir werden später sehen, dass auch neuere, in der Praxis verbreitete Vorgehensmodelle eine derartige Aufteilung verwenden. Trotzdem – oder auch gerade deswegen – erscheint es notwendig, sich über die damit verbundenen Probleme und Gefahren im Klaren zu sein. Vier wesentliche Problembereiche sollen deshalb hier herausgestellt werden.

1. *weiterhin problematische Kontrolle des Projektfortschritts*

   Durch die Zerlegung werden Softwareprojekte scheinbar plan- und kontrollierbar, weil man jetzt zu jedem Zeitpunkt genau sagen kann, in welcher Phase man sich gerade befindet. Das Problem ist aber, dass diese Kontrollmöglichkeit beliebig unterlaufen werden kann!

   Wenn in den frühen Phasen oberflächlich und ungenau gearbeitet wird, dann erreicht man sehr früh weiterführende Phasen und es entsteht der Eindruck eines erfolgreichen Projektfortschritts. Dass sich durch die ober-

flächliche Arbeit aber im Weiteren ein großer Mehraufwand ergeben wird, ist dabei nicht transparent.

verhängnisvoller Dominoeffekt

In der Praxis ist an dieser Stelle bisweilen sogar ein verhängnisvoller Dominoeffekt zu beobachten: Die erste Phase des Vorgehensmodells wird vom Projektteam und vom Management schnell abgehakt. In der zweiten Phase bekommt dann das Projektteam allmählich ein ungutes Gefühl, gleichzeitig wächst aber auch die Erwartungshaltung des übergeordneten Managements, das erwartet, dass es nach der schnell abgeschlossenen ersten Phase zügig weitergeht. Also lässt man sich im Projektteam dazu hinreißen, auch die zweite Phase recht oberflächlich zu bearbeiten. Dies setzt sich dann so fort und kommt bisweilen erst in der Implementierungsphase zu einem dann recht folgenschweren Projektzusammenbruch.

2. *Zusammenhänge zwischen Phasen werden unzureichend abgebildet*

Dadurch, dass jetzt einzelne relativ gut beschriebene Phasen existieren, scheint es möglich zu sein, die Phasen einzeln zu planen und zu kontrollieren. Dabei wird leicht übersehen, dass die Phasen aufeinander aufbauen. Die Ergebnisse der Phasen müssen zueinander passen. Wenn man z.B. das Ergebnis des technischen Systementwurfs betrachtet, dann muss man sich nicht nur die Frage stellen, ob es sich hier um einen technisch eleganten Entwurf handelt, sondern auch, ob dieser Entwurf mit den fachlichen Anforderungen harmoniert. Wenn dies nicht der Fall ist, ist zu klären, ob die Ursache im technischen Systementwurf liegt oder ob vielleicht im fachlichen Konzept Dinge übersehen wurden. Ist Letzteres der Fall, so müssen in einer eigentlich bereits abgeschlossenen Phase Verbesserungen vorgenommen werden, was einem strengen wasserfallartigen Vorgehen widerspricht.

3. *Testaktivitäten werden zu sehr als Phase gedacht*

Dadurch, dass z.B. in dem Vorgehensmodell in Abbildung 10 eigene Phasen für Test und Wartung vorgesehen sind, scheinen bei oberflächlicher Betrachtung Qualitätskriterien wie Korrektheit und Wartbarkeit in das Modell integriert zu sein. Das Gegenteil ist aber der Fall.

Sehr gut kann man dies mit einem plastischen Vergleich darstellen: Die Testaktivitäten und die Qualitätssicherungsaktivitäten in einem Projekt können mit einem Wachhund verglichen werden. Sperrt man diesen Wachhund in einen Zwinger – eine eigene Phase – ein, dann darf man sich nicht wundern, wenn er außerhalb des Zwingers nicht für Ordnung und Sicherheit sorgen kann. Deshalb sollte man dem „Wachhund“ im Projekt freien Auslauf lassen, d.h. man sollte Qualitätssicherung als phasenübergreifende Grundaufgabe verstehen – die natürlich nicht von bissigen Hunden, sondern von einfühlsamen Fachleuten wahrgenommen werden muss.

4. *bei großen Projekten vergeht viel Zeit, bis das erste System erstellt wird*

   Bei umfangreichen Projekten dauert es bei Anwendung der beschriebenen wasserfallartigen Vorgehensmodelle zum Teil weit mehr als ein Jahr, bevor ein erstes System begutachtet werden kann. Dies kann zwar von der Sache her logisch begründet werden, es erscheint aber dennoch aus mehreren Gründen problematisch: So erhöht sich die Mitarbeitermotivation im Allgemeinen dadurch, dass man ein lauffähiges System vorführen kann. Ob es sich dabei um das endgültige System oder um eine Zwischenstufe handelt, ist weniger wichtig. Auch auf Seiten des übergeordneten Managements ergibt sich eine höhere Sicherheit, wenn man ein erstes lauffähiges System vorweisen kann. Schließlich kann ein solches System im Hinblick auf die Abstimmung mit dem Auftraggeber hilfreich sein, weil eine Produktdefinition „auf Papier“ für Auftraggeber oft missverständlich ist.

   Hier erscheint also ein Vorgehensmodell, das durch die Integration von Prototypen frühzeitig ein lauffähiges Teilsystem liefert, zumindest erwägenswert.

Integration von Prototypen

Im Folgenden werden wir nun verschiedene Variationen und Weiterentwicklungen des klassischen Wasserfallmodells betrachten, die den beschriebenen Kritikpunkten zumindest teilweise Rechnung tragen.

1.6.4 Das Wasserfallmodell mit Rücksprüngen

Gründe für Rücksprünge

In der Praxis kann es aus zwei Gründen zu Situationen kommen, die es sinnvoll erscheinen lassen, erneut in eine Phase einzutreten, die eigentlich bereits abgeschlossen war. Zum einen können in einer Phase Fehler in den Ergebnissen einer vorherigen Phase entdeckt werden. In diesem Fall ist eine Überarbeitung des entsprechenden Ergebnisses einer früheren Phase erforderlich. Man sollte dabei aber sehr restriktiv sein und sich auf die Behebung „wirklicher“ Fehler beschränken. Wegen „Geschmacksfragen“ sollte man nicht erneut in eine bereits abgeschlossene Phase eintreten.

neue Anforderungen

Die zweite Motivation zu einem erneuten Eintritt in eine bereits abgeschlossene Phase können neue Anforderungen an das System sein. Auch hier sollte man sehr restriktiv sein. Sich fortlaufend wandelnde Anforderungen sind eine der Hauptursachen für das Scheitern von Softwareprojekten! Man sollte immer zunächst die Möglichkeit diskutieren, die neuen Anforderungen als Aufgaben für eine Folgeversion des Systems zu sehen. Nur bei unter keinen Umständen verschiebbaren neuen Anforderungen (oder bei solchen, die fast keinen Aufwand verursachen) sollte man eine Anpassung der Ergebnisse bereits abgeschlossener Phasen durchführen.

Abbildung 15 zeigt das aus Abbildung 10 bekannte Vorgehensmodell in einer um die Möglichkeit von Rücksprüngen erweiterten Variante.

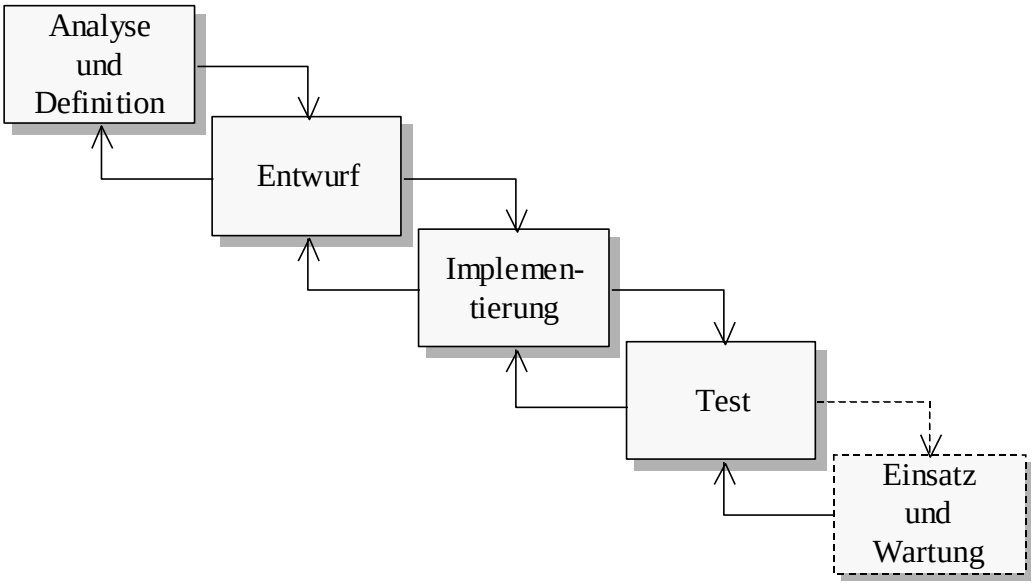


Abbildung 15: Ein einfaches Vorgehensmodell, das Rücksprünge berücksichtigt

1.6.5 Projektbegleitende Qualitätssicherung

Ein anderer Kritikpunkt am klassischen Wasserfallmodell besteht in der „Reduzierung“ der Testaktivitäten auf eine einzige sehr spät im Projektablauf angesiedelte Phase. Zweifellos ist nach der Implementierung des Systems ein ausgiebiger Test erforderlich. Dies darf aber nicht die einzige Qualitätssicherungsmaßnahme im Projektablauf sein.

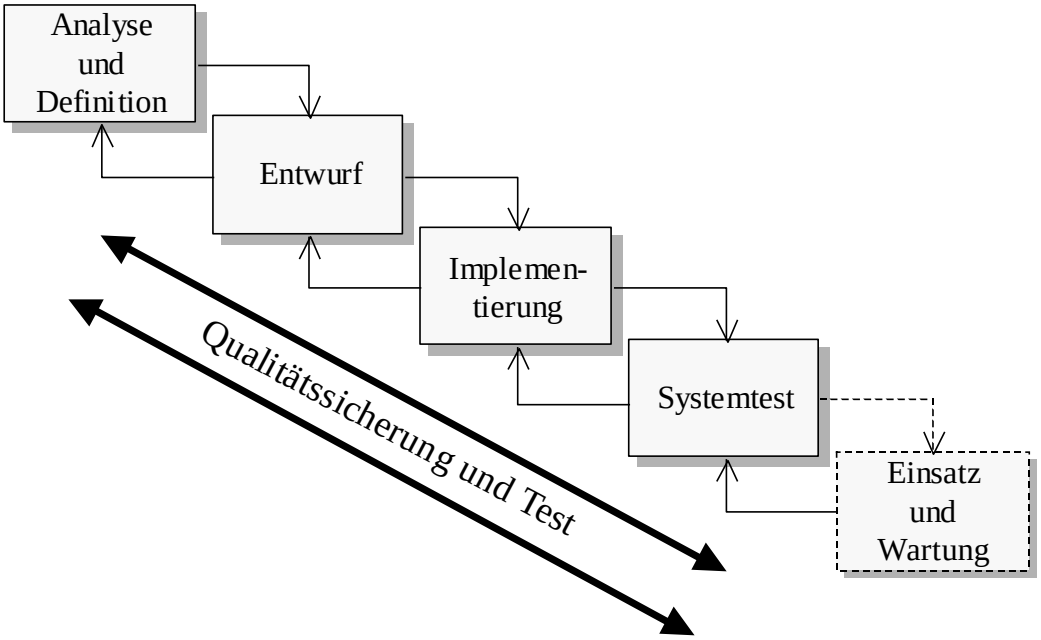


Abbildung 16: Qualitätssicherung als phasenübergreifende Aufgabe

Systemtest und begleitende Qualitätssicherung

In Abbildung 16 wird daher die Testphase aus dem in Abbildung 10 dargestellten Vorgehensmodell konkreter als *Systemtest* bezeichnet und zusätzlich deutlich gemacht, dass während des gesamten Projektablaufs eine begleitende Qualitätssicherung erfolgen muss.

Wasserfallmodell nach Boehm

Boehm /Boehm 1981/ berücksichtigt die Notwendigkeit einer projektbegleitenden Qualitätssicherung, indem er bei jeder Aufgabe im Vorgehensmodell ausdrücklich eine entsprechende Qualitätssicherung vorsieht (vgl. Abbildung 17). Er unterscheidet dabei folgende Phasen:

- *Machbarkeit des Systems:* Ein Konzept für das zu entwickelnde Softwaresystem wird erstellt und hinsichtlich seiner Umsetzbarkeit und seiner Vorteilhaftigkeit gegenüber alternativen Lösungsansätzen untersucht.

- *Software-Pläne und Anforderungen:* Eine vollständige und validierte Spezifikation der erforderlichen Funktionalität, der erforderlichen Schnittstellen und der erforderlichen Performance des Softwaresystems wird erarbeitet.

- *Produkt-Design:* Eine vollständige und verifizierte Spezifikation der übergreifenden Hardware- und Softwarearchitektur, der Kontrollstrukturen und der Datenstrukturen des Systems wird entwickelt. Zusätzlich werden vorläufige Versionen der Benutzerhandbücher und der Testpläne erstellt.

- *Detailliertes Design:* Für jede einzelne Programmkomponente wird eine vollständige und verifizierte Spezifikation der Kontrollstrukturen, der Datenstrukturen, der Schnittstellenbeziehungen, der Schlüsselalgorithmen und der zugrundeliegenden Annahmen erarbeitet.

- *Programmierung:* Eine vollständige und verifizierte Menge von Programmkomponenten wird erstellt.

- *Integration:* Die Komponenten werden zu einem funktionsfähigen System zusammengesetzt.

- *Einführung:* Das System wird in Betrieb genommen. Dies schließt Aufgaben wie die Benutzerschulung, Datenkonvertierungen von Altsystemen etc. ein.

- *Einsatz und Wartung*

Abbildung 17 fasst dieses Vorgehensmodell zusammen.

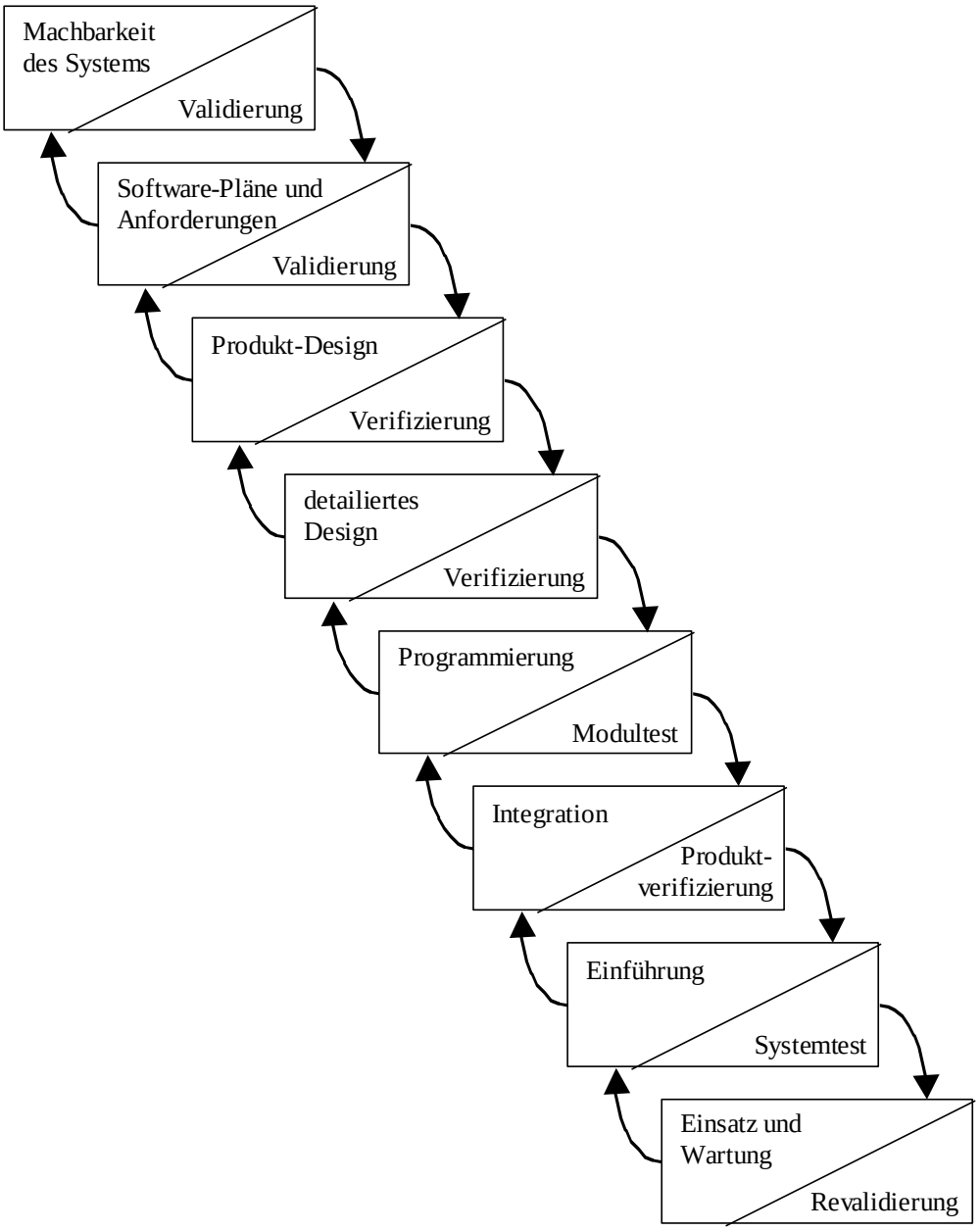


Abbildung 17: Das Wasserfallmodell des Software-Lebenszyklus (nach /Boehm 1981/, S.36)

Validierung und Verifizierung

Die in dieses Vorgehensmodell integrierten Validierungs- und Verifikationsabschnitte sollen eine projektbegleitende Qualitätssicherung gewährleisten. Von einer *Verifikation* ist dabei die Rede, wenn das Ergebnis einer Aktivität gegen seine Spezifikation überprüft wird. Dies ist dann der Fall, wenn die vorherige Aktivität ein hinreichend präzises Ergebnis geliefert hat, das als Aufgabenbeschreibung und Spezifikation für die folgende Phase interpretiert werden kann. Existiert eine solche präzise Vorgabe nicht – z.B. in den ersten beiden Phasen des Ablaufs – so erfolgt eine *Validierung*, die allgemeiner die Eignung des Prüfgegenstands für den angestrebten Zweck untersucht.

Der Unterschied zwischen Verifikation und Validierung wird etwas überspitzt oft dadurch zusammengefasst, dass in der Verifikation überprüft wird, ob das Produkt richtig entwickelt wird, während sich eine Validierung der allgemeineren Frage zuwendet, ob überhaupt das richtige Produkt entwickelt wird.

1.6.6 Prototyping

Durch die Entwicklung von Prototypen kann man im Rahmen einer Projektentwicklung frühzeitig zu einem lauffähigen – aber nicht unbedingt einsatzfähigen – System kommen. Dadurch kann einerseits die Motivation der Mitarbeiter erhöht und andererseits die Abstimmung mit dem Auftraggeber erleichtert werden. Wir müssen dabei aber unterschiedliche Arten von Prototypen differenzieren.

Abbildung 18 zeigt die Integration eines Prototyps zur Validierung der Produktdefinition in das Vorgehensmodell.

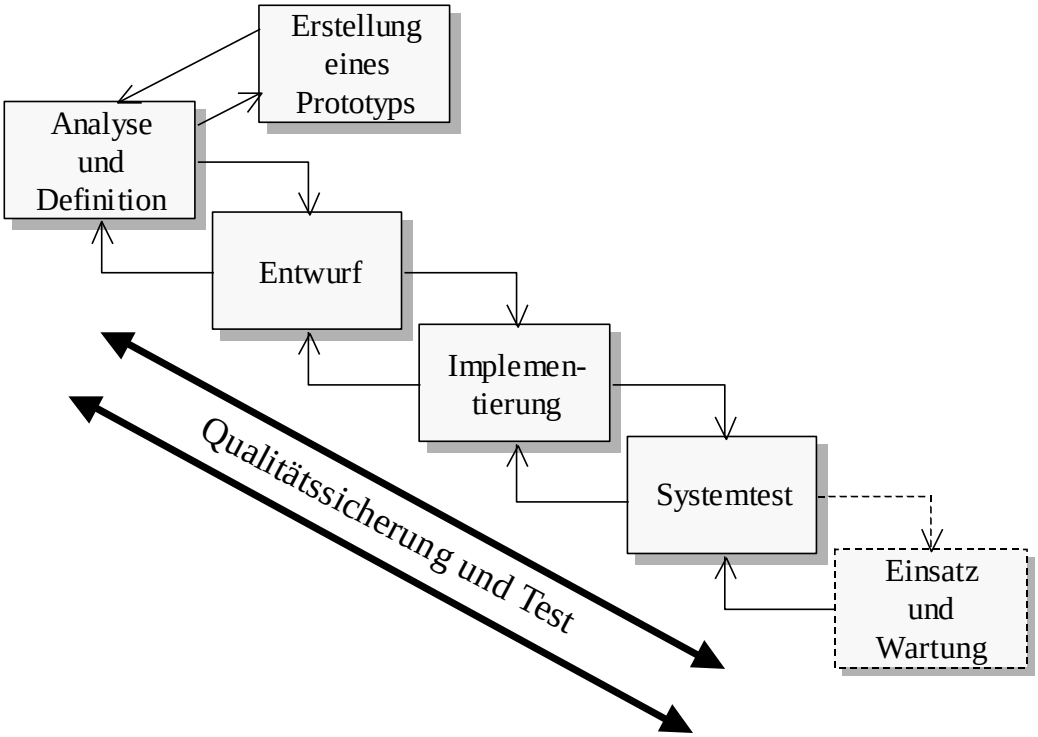


Abbildung 18: Prototypen zur Absicherung der Produktdefinition

Rapid Prototyping

Die Erstellung des Prototyps dient hier lediglich der besseren Abstimmung mit dem Auftraggeber. Der Prototyp wird so lange revidiert, bis die Anforderungen feststehen. Im Allgemeinen reicht hierzu ein Prototyp der Benutzungsoberfläche, der sich mit einem entsprechenden Werkzeug relativ schnell erstellen lässt. Es handelt sich dabei aber um einen Wegwerfprototyp – analog zu einem Modell in der Architektur. Man bezeichnet den Einsatz eines Prototyps in diesem Sinne oft auch als *Rapid Prototyping.*

Ein Problem im Hinblick auf den praktischen Einsatz des Rapid Prototyping ergibt sich bisweilen dadurch, dass den Auftraggebern, nachdem diese in Form des Prototyps bereits eine „fast“ fertige Benutzungsoberfläche gesehen haben, schwer zu vermitteln ist, warum zur Erstellung des fertigen Systems noch ein so hoher Aufwand nötig sein soll.

Evolutionäre SW-Entwicklung

Eine andere Art des Prototyping ist die evolutionäre Softwareentwicklung. Insbesondere bei innovativen Projekten ist bisweilen das Problem nicht ausreichend

definiert oder Daten (z.B. im Hinblick auf Performanceanforderungen) sind nicht vollständig vorhanden. In diesen Fällen kann ein experimentelles und exploratives Vorgehen während der Systementwicklung hilfreich sein. Statt eines Wegwerfprototyps wird hier eine Serie von Prototypen entwickelt, die direkt in die Produktionsversion der Software konvergiert.

Dabei wird für jeden einzelnen Prototypen der vollständige Entwicklungszyklus durchlaufen. Die ersten Prototypen erreichen jedoch nicht die Phase *Einsatz und Wartung*, sondern sie dienen nach einer ausführlichen Begutachtung lediglich als Basis für eine Verfeinerung der Analyse- und Definitionsergebnisse, die dann zu einer Weiterentwicklung des Prototyps in dem folgenden Entwicklungszyklus führt. Abbildung 19 verdeutlicht diesen Prozess, der sich gegebenenfalls noch über mehrere Stufen erstrecken kann.

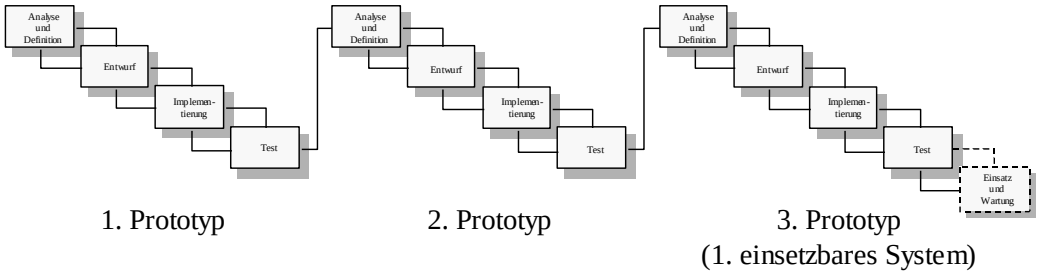


Abbildung 19: Evolutionäre Softwareentwicklung

Bei einem derartigen Vorgehen erscheint es im Allgemeinen sinnvoll, in den frühen Phasen der Prototyperstellung (*Analyse und Definition* sowie *Entwurf*) durchaus auch das Gesamtsystem im Auge zu behalten. Die Implementierung wird aber in jedem Fall auf den eingeschränkten Prototyp reduziert.

Idealtypisch können dabei, wie in Abbildung 20 dargestellt, horizontale und vertikale Prototypen unterschieden werden.

horizontaler Prototyp

Horizontale Prototypen realisieren nur eine bestimmte technische Facette des Systems, wie z.B. die Benutzungsoberfläche oder die fachliche Anwendungsfunktionalität. Letzteres könnte z.B. bedeuten, dass man sich bei einem System zur Routenplanung für Außendienstmitarbeiter zunächst auf die eigentliche Routenplanung beschränkt, die ihre Eingaben und Ausgaben in einfacher textueller Form erhält bzw. ausgibt. Damit könnte zunächst verifiziert werden, ob die berechneten Ergebnisse den Anforderungen entsprechen.

vertikaler Prototyp

Bei einem vertikalen Prototyp wird dagegen für einen Teilausschnitt des geforderten Funktionsumfangs ein System entwickelt, das von der Benutzungsschnittstelle über die fachliche Funktionalität bis zur Systemintegration alle Bereiche überdeckt. So könnte man in einem umfangreichen System z.B. zunächst die vollständige Stammdatenverwaltung realisieren.

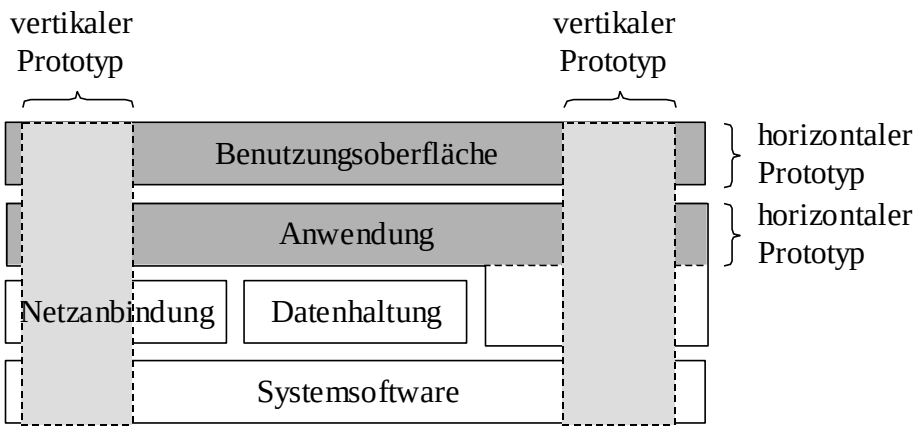


Abbildung 20: Arten von Prototypen (nach /Balzert 2008/, S. 540)

größere Nähe zum Auftraggeber

Die Vorteile einer solchen evolutionären Softwareentwicklung liegen zunächst in einer größeren Nähe zum Auftraggeber/Benutzer. Da man in kürzeren Abständen fertige Teilergebnisse mit dem Auftraggeber abstimmen kann, erhöht sich im Allgemeinen auch die Akzeptanz des Systems.

frühes Training der Benutzer

Ein weiterer Vorteil besteht darin, dass – insbesondere dann, wenn der Prototyp die Benutzungsoberfläche umfasst – ein frühes Training der Benutzer möglich wird. Auch die Motivation der Entwickler wird durch das zügige Erstellen eines ersten lauffähigen Systems erhöht.

Nachteile

Diesen Vorteilen stehen allerdings auch Nachteile gegenüber. So erfordert die evolutionäre Softwareentwicklung leicht änderbare Software. Eine „quick & dirty-Realisierung“ wie beim Wegwerf-Prototyp ist hier nicht geeignet. Ein weiteres Problem ist, dass auch die komplette Dokumentation des Systems evolutionär weiterentwickelt werden muss – ein Aspekt, der in der Praxis oft vernachlässigt wird.

Daneben ist anzumerken, dass man das – dann falsch verstandene – Prinzip der evolutionären Softwareentwicklung oft auch nur als Ausrede für einen mehr oder weniger unstrukturierten Entwicklungsprozess verwendet. Dabei erfordert gerade die evolutionäre Entwicklung eine sorgfältige Planung, um die einzelnen Stufen der Evolution gezielt und angemessen umzusetzen.

1.6.7 Das Spiralmodell

das Spiralmodell als Metamodell

Das von Boehm in den 80er Jahren des 20. Jhs. vorgestellte Spiralmodell (vgl. /Boehm 1986/ und /Boehm 1988/) ist ein Versuch, die Stärken verschiedener Phasenmodelle so miteinander zu kombinieren, dass das Projektrisiko, d.h. das Risiko einer Zeit- oder Kostenüberschreitung, ggf. sogar eines vollständigen Scheiterns, minimiert wird. Das Spiralmodell kann dabei als Metamodell betrachtet werden, das für die einzelnen Phasen im Projektablauf wiederum einen allgemeinen Ablauf definiert.

Abbildung 21 zeigt den grundsätzlichen Ablauf des Spiralmodells. Ein Umlauf, der im Spiralmodell in vier Teilphasen aufgebrochen wird, entspricht dabei einer Phase oder Stufe der Projektentwicklung. So könnte der in der Abbildung grau hinterlegte Bereich z.B. einer Phase im Sinne des klassischen Wasserfallmodells entsprechen.

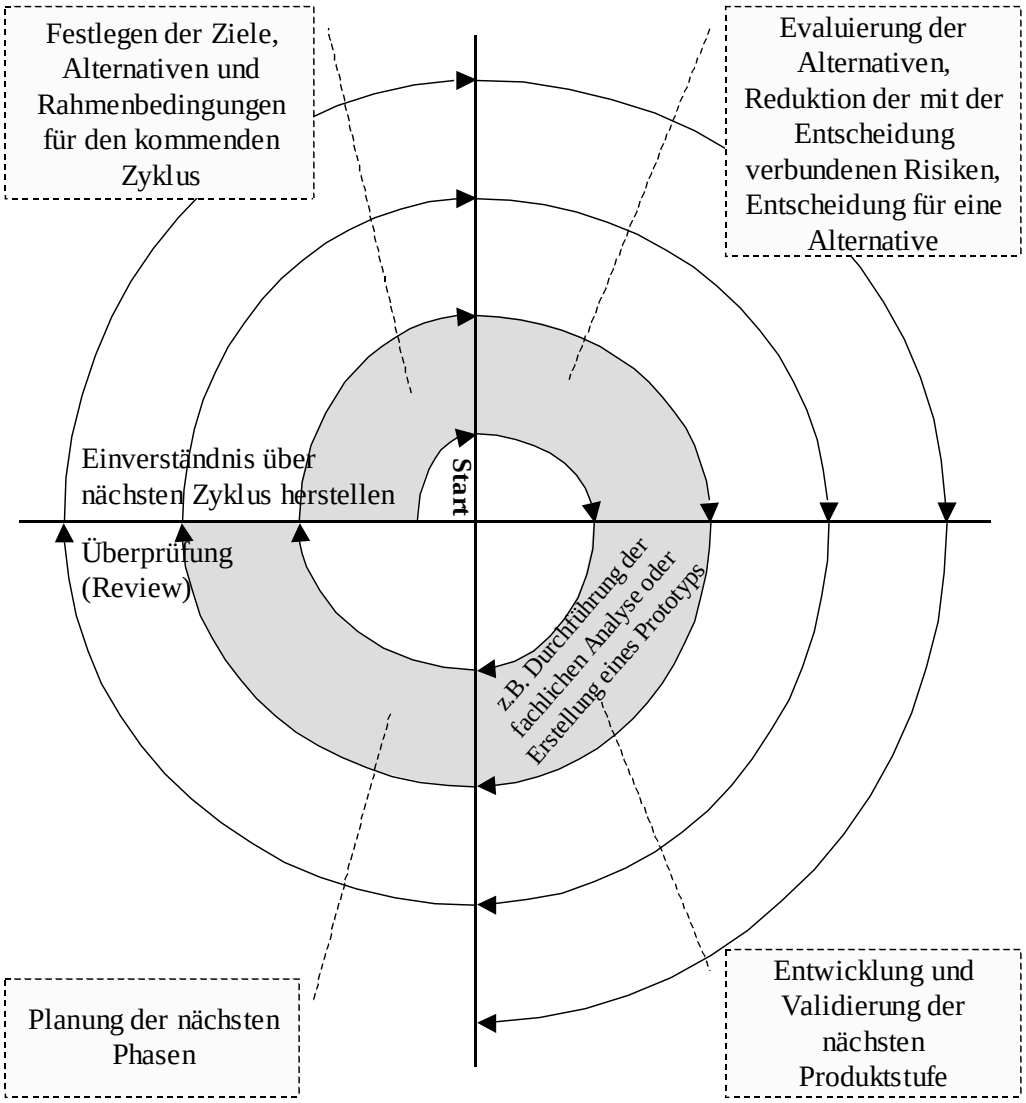


Abbildung 21: Grundgedanke des Spiralmodells

Die vier Teilphasen eines Umlaufs können wie folgt beschrieben werden:

- *1. Phase: Ziele, Alternativen, Rahmenbedingungen*

  In dieser Phase geht es zunächst um die Festlegung der inhaltlichen Vorgaben für den nächsten Arbeitsschritt. Will man z.B. im Sinne eines evolutionären Prototyping eine weitere Evolutionsstufe angehen, so könnte dies bedeuten, dass man Überlegungen hinsichtlich der Funktionalität, der Qualitätskriterien oder der Nutzung von Bibliotheken zur Wiederverwendung anstellt.

Hieran schließt sich das Herausarbeiten alternativer Vorgehensweisen an. So könnte man z.B. unterschiedliche Analyse- oder Entwurfstechniken, verschiedene mögliche Werkzeuge oder die Möglichkeiten einer Eigenentwicklung im Verhältnis zu einer Fremdvergabe in Betracht ziehen.

Schließlich sind noch die Einschränkungen zu ermitteln, die hinsichtlich Zeit, Personal, Kosten oder z.B. der Hardware- und Software-Umgebung bei der Entscheidung zu beachten sind.

- *2. Phase: Evaluierung der Alternativen und Reduktion der Risiken*

Ziel: Risikominimierung

In dieser Phase geht es nun um die Auswahl einer der möglichen alternativen Vorgehensweisen. Dabei wird im Spiralmodell besonderer Wert darauf gelegt, dass die mit dieser Entscheidung verbundenen Risiken so gering wie möglich gehalten werden sollten. Hierzu wird empfohlen, bei der Einschätzung und Überprüfung der Alternativen ggf. auf Ansätze wie Prototyping, Simulation, Datenmodellierung, Benchmark-Tests (z.B. Leistungs- und Durchsatzmessungen bei großen Datenbankanwendungen), Marktforschung oder Literaturrecherchen zurückzugreifen. Das Ziel ist jeweils, die Folgen der Umsetzung einer Alternative möglichst sicher abschätzen zu können, um so die mit der Entscheidung verbundenen Risiken zu reduzieren.

Schließlich sollte auf dem in dieser Weise abgesicherten Informationsstand die Auswahl einer umzusetzenden Alternative erfolgen. Hierzu bieten sich die aus der Entscheidungstheorie bekannten Entscheidungstechniken an, auf die wir in Kapitel 3 eingehen werden.

- *3. Phase: Realisierung und Überprüfung des Zwischenprodukts*

In dieser Phase erfolgt nun die Realisierung der ausgewählten Alternative. Dabei sollte in jedem Fall eine sorgfältige Qualitätssicherung der Ergebnisse durchgeführt werden. Die Maxime lautet auch hier *Risikominimierung*!

- *4. Phase: Planung der Projektfortsetzung*

In dieser Phase geht es um die Planung des nächsten Zyklus. Dabei ist auch an die Planung der benötigten Ressourcen zu denken. Im Sinne der Risikominimierung ist hier ggf. an eine Aufteilung in mehrere parallele Teilprojekte zu denken.

Wie geht es weiter?

Die Phase sollte dabei einen ausführlichen Review der Ergebnisse und Erfahrungen aus dem zurückliegenden Zyklus mit einschließen, der eine Bewertung des erzielten Fortschritts zum Ziel hat. Auf dieser Basis sollte ein breiter Konsens unter allen Projektbeteiligten über die Projektsituation

hergestellt werden. Das Ergebnis kann dabei ein „*Commitment*“ über den Eintritt in einen weiteren Zyklus oder auch eine Entscheidung zum Projektabbruch sein. Durch die Erwägung eines möglichen Projektabbruchs soll vermieden werden, dass sich Projekte, die eigentlich in einer ausweglosen Situation stecken, ohne konkrete Entscheidung noch über eine lange Zeit hinziehen. Ziel ist dabei wieder die Risikominimierung, die man in diesem Fall mit *„Lieber ein Ende mit Schrecken als ein Schrecken ohne Ende“* charakterisieren kann.

Rahmen für die Anwendung und Nutzung anderer Vorgehensmodelle

Das Spiralmodell versucht einen Rahmen für die Anwendung und Nutzung anderer Vorgehensmodelle aufzubauen. Durch den Einbau bewusster Sollbruchstellen im Projektablauf wird die Möglichkeit geschaffen, aus dem ursprünglich angestrebten Vorgehen bei Bedarf auch wieder ausbrechen zu können.

Ein Beispiel hierfür könnte darin bestehen, dass man im ersten Umlauf im Sinne des klassischen Wasserfallmodells mit einer Analyse des Gesamtsystems begonnen hat. Dabei hat sich aber herausgestellt, dass das Gesamtsystem zu umfangreich und auch zu ambitioniert war. Man geht nun also im nächsten Zyklus des Spiralmodells einen Schritt zurück und konzentriert sich im Sinne einer evolutionären Systementwicklung auf die Erstellung einer ersten Kernfunktionalität.

Damit bietet das Spiralmodell die Möglichkeit, einen einmal eingeschlagenen Weg gezielt und strukturiert immer wieder zu hinterfragen und eben nicht „ohne Rücksicht auf Verluste“ bis zum Ende zu gehen.

1.6.8 Das V-Modell

Nachdem wir in den bisherigen Abschnitten die wesentlichen grundlegenden Vorgehensmodelle angesprochen haben, wollen wir in diesem und den folgenden Abschnitten nun vier in der Praxis recht populäre Vorgehensmodelle vorstellen, die als Kombination und praxisorientierte Umsetzung der Ideen der bisher besprochenen Ansätze verstanden werden können. Es handelt sich um das *V-Modell*, den *Rational Unified Process*, das *eXtreme Programming* (XP) und *Scrum*.

Entwicklungsstandard für IT-Systeme des Bundes

Das V-Modell, das wir zunächst betrachten wollen, wurde als Entwicklungsstandard für IT-Systeme des Bundes entwickelt[7]. Es wurde zunächst im Auftrag des Verteidigungsministeriums erstellt und dann allgemein für die Bundesverwaltung übernommen. Das V-Modell beschreibt sehr detailliert das Vorgehen bei der Entwicklung von IT-Systemen, die sowohl aus Hardwarekomponenten als auch aus Softwarekomponenten bestehen können.

---

7 Weiterführende Informationen zum V-Modell einschließlich der vollständigen Modellbeschreibungen können unter www.v-modell-xt.de im Internet abgerufen werden. Die Ausführungen in diesem Abschnitt basieren im Wesentlichen auf den dort verfügbaren umfangreichen Dokumenten.

Das ursprüngliche Modell von 1992 wurde kontinuierlich überarbeitet, ergänzt und an neue technologische Entwicklungen angepasst. Im Jahr 1997 wurde das *V-Modell 97* publiziert, im Jahr 2005 schließlich das *V-Modell XT*, das seit Mai 2012 in Version 1.4 vorliegt. *XT* steht dabei für *eXtreme Tailoring* und verweist darauf, dass das Modell (seinem Anspruch nach) für unterschiedliche Projekt- und Organisationsprofile hoch anpassungsfähig ist.

Auch wenn das V-Modell für die Bundesverwaltung entwickelt wurde, kann es grundsätzlich für alle IT-Projekte eingesetzt werden. Es ist unter der *Apache License* frei verfügbar und bietet umfangreiche Referenzdokumente und Produktvorlagen an.

Ziele des V-Modells XT

Die Ziele, die dabei verfolgt werden, sind:

- Minimierung der Projektrisiken
- Verbesserung und Gewährleistung der Qualität
- Eindämmung der Gesamtkosten
- Verbesserung der Kommunikation zwischen allen Beteiligten

Die Zielgruppenbeschreibung ist sehr breit angelegt. Sie umfasst Auftraggeber wie Auftragnehmer eines Projekts, Projektleiter wie Projektmitarbeiter.

Gesamtstruktur des V-Modells XT

Nicht jedes Projekt wird nach dem V-Modell in der gleichen Weise organisiert und gesteuert. In Abhängigkeit von bestimmten Eigenschaften werden Projekte zunächst klassifiziert und nach *Projekttypen* eingeteilt. Für jeden Projekttyp wiederum sind *Projekttypvarianten* vorgesehen, die maßgeblich sind für die *Projektdurchführungsstrategie* und für die Wahl von *Vorgehensbausteinen*. Die Durchführungsstrategie determiniert die *Entscheidungspunkte*, an denen im Projektverlauf der Stand des Projekts evaluiert wird und Entscheidungen über die weitere Entwicklung getroffen werden.

Die *Vorgehensbausteine* stehen jeweils für konkrete Aufgabenstellungen im Rahmen des Projekts und sind teilweise obligatorisch, teilweise können sie zusätzlich ausgewählt werden. Ein Vorgehensbaustein legt sowohl das *Produkt* fest, das innerhalb einer Aufgabenstellung zu erarbeiten ist, als auch die zugehörigen *Aktivitäten* und *Rollen*. Einige Vorgehensbausteine sind verbindlich für alle V-Modell-konformen Projekte; sie bilden in ihrer Gesamtheit den V-Modell-Kern.

Das V-Modell sieht vier verschiedene *Projekttypen* vor:

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

Klassifizierung von Projekten

Die Klassifizierung nach Projekttyp unterscheidet also nach der Projektrolle als *Auftraggeber* (AG – Projekte zur Vergabe eines Auftrags), *Auftragnehmer* (AN – Projekte zur Systementwicklung) bzw. danach, ob beide Rollen innerhalb einer Organisation ein Projekt durchführen (AG/AN). Jeder Rolle korrespondieren eine besondere Sichtweise und unterschiedliche Aufgaben im Rahmen eines Projekts. Außerdem bietet das V-Modell einen eigenen Projekttyp für die Einführung und Pflege eines organisationsspezifischen Vorgehensmodells an.

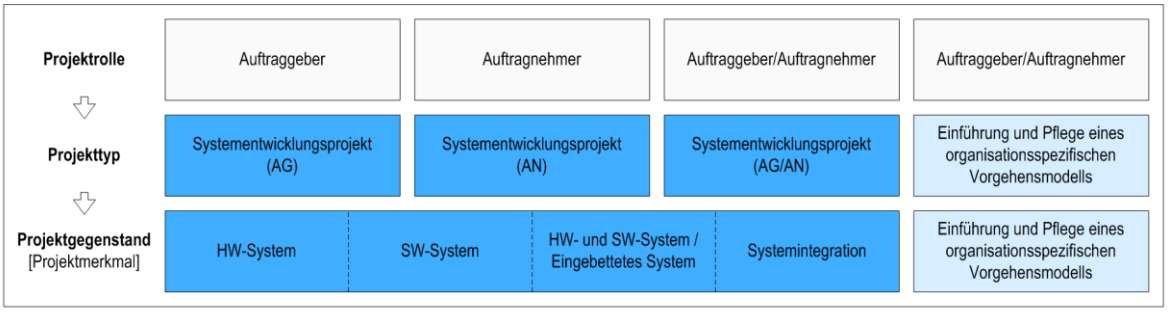


Abbildung 22: Klassifizierung von Projekten in Projekttypen
(aus: /V-Modell XT, Version 1.4/ 1-13)

Eine weitere Spezifikation erfolgt über den *Projektgegenstand*. Im Rahmen des V-Modells XT kann dies die Entwicklung von Hardware (HW), Software (SW) oder komplexen bzw. eingebetteten Hardware- und Software-Systemen und die Systemintegration sein.

Vorgehensbausteine

Der Kern des V-Modells sind die *Vorgehensbausteine*. Sie sind modular strukturiert als eigenständige Einheiten, die jeweils geändert und erweitert werden können. Ein Vorgehensbaustein kapselt alle Komponenten, die zur Erfüllung einer Aufgabenstellung im Rahmen des Projekts notwendig sind. Das können *Produkte* und *Aktivitäten* sein.

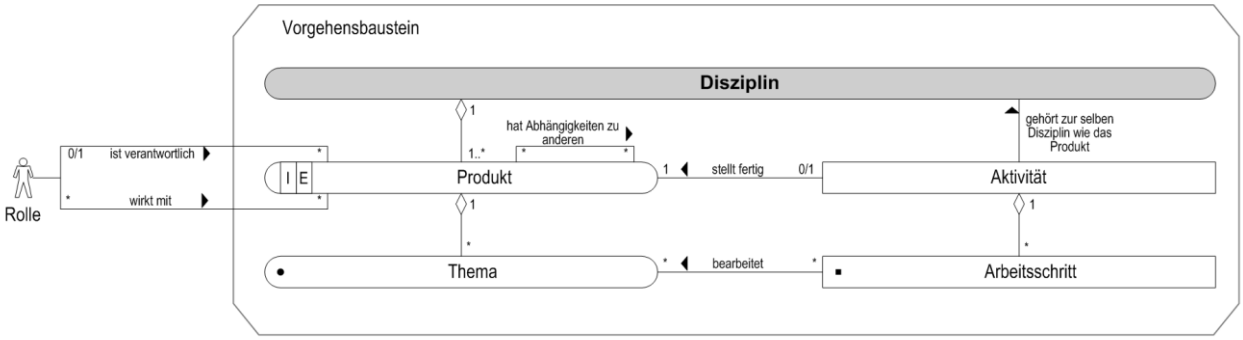


Abbildung 23: Vorgehensbausteine und ihre Komponenten (aus: /V-Modell XT, Version 1.4/ 1-15)

Produkte sind Ergebnisse bzw. Zwischenergebnisse in der Projektdurchführung, z.B. ein Projektplan oder das Release einer SW-Komponente. Inhaltlich zusammenhängende Produkte werden zu einer *Disziplin* zusammengefasst. Es gibt Produkte, die obligatorisch in jedem V-Modell-Projekt erstellt werden, die sogenann-

ten *initialen Produkte* (im Unterschied zu *externen Produkten*), z.B. ein Projektplan oder das Projekthandbuch.

Ein Produkt wird im V-Modell von genau einer Aktivität erstellt. *Aktivitäten* beschreiben die Art und Weise, wie ein Produkt zu erarbeiten ist. Sie lassen sich weiter untergliedern in *Arbeitsschritte*, die als eine Art Arbeitsanleitung ausformuliert werden.

Schließlich gehören zu einem Vorgehensbaustein *Rollen* mit der Festlegung von Mitwirkungsbeziehungen und Verantwortlichkeiten. Beim Tailoring, also beim Anpassen des abstrakten V-Modells an ein konkretes Projekt, wird jedem Produkt genau eine verantwortliche Rolle, zudem weitere mitwirkende Rollen zugewiesen und es werden für jede Rolle konkrete Personen bzw. Organisationseinheiten benannt.

Projektdurchführungsstrategie und Entscheidungspunkte

Die grundlegende Struktur des Projektverlaufs wird festgelegt durch Wahl einer *Projektdurchführungsstrategie*. Sie legt die Reihenfolge der zu erstellenden Produkte und der zugehörigen Aktivitäten fest. Im Projektfortschritt gibt es dann definierte *Entscheidungspunkte*, an denen der Projektstand auf Basis der erstellten Produkte evaluiert wird. Anschließend trifft das Projektmanagement die Entscheidung, ob die nächste Projektphase eingeleitet werden kann oder eine Iteration eingeschaltet wird oder ob im äußersten Fall das Projekt abgebrochen wird.

In *allen* nach V-Modell organisierten Projekten gibt es die Entscheidungspunkte: Projekt genehmigt, Projekt definiert, Iteration geplant und Projekt abgeschlossen. Zur Systementwicklung gehören: Anforderungen festgelegt, System spezifiziert, System entworfen, Feinentwurf abgeschlossen, Systemelemente realisiert und System integriert. Soweit ein Projekt noch vor der Festlegung der Anforderungen in Teilprojekte aufgeteilt wird, werden zusätzlich die folgenden Entscheidungspunkte gesetzt: Gesamtprojekt aufgeteilt und Gesamtprojektfortschritt. Ein dritter Bereich legt die Entscheidungspunkte für die Schnittstelle von Auftraggeber und Auftragnehmer fest, zum Beispiel: Projekt ausgeschrieben und Abnahme erfolgt. Schließlich gibt es weitere vorgegebene Entscheidungspunkte für den besonderen Fall der Entwicklung eines organisationsspezifischen Vorgehensmodells.

Für die *Systementwicklung* (Entwicklung von HW- und/oder SW-Komponenten) lassen sich die Entscheidungspunkte in der folgenden Weise strukturieren:

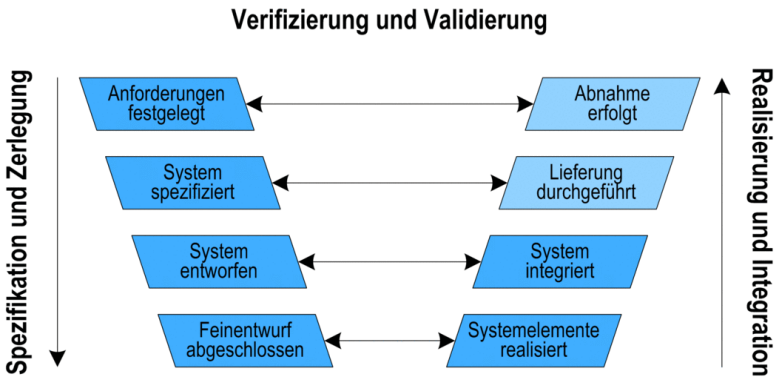


Abbildung 24: Struktur der Systementwicklung (aus: /V-Modell XT, Version 1.4/ 1-35)

Den Stufen der Spezifikation und Zerlegung des zu entwickelnden Systems entsprechen in umgekehrter Reihenfolge die Prozessstufen der Realisierung und Integration. In diesem V-förmigen Modell korrespondieren jeweils die Entscheidungspunkte der Spezifikation zu denen der Realisierung. Eine Verifikation (= Überprüfung der Übereinstimmung von Spezifikation und Produkt) und die Validierung (= Überprüfung der Relation von Produkt und Zweck) erfolgen auf jeder Stufe.

Tailoring

Im Prozess des *Tailoring* wird dieses allgemeine Modell durch Auswahl und Sequenzierung der Entscheidungspunkte heruntergebrochen auf ein konkretes Projekt.

Für die projektspezifische Anpassung des sogenannten *Tailoring* stellt das V-Modell als Softwarewerkzeug einen Projektassistenten zur Verfügung, der die Identifikation und die Auswahl von Vorgehensbausteinen und Projektmerkmalen unterstützt und frei verfügbar ist. Um dem dynamischen Charakter eines Projektverlaufs Rechnung zu tragen und ein hohes Maß an Flexibilität zu gewährleisten, werden beim Tailoring nicht nur die Anpassungsschritte der initialen Planungsphase berücksichtigt, das sogenannte *statische* Tailoring, sondern auch neue Justierungen während der Projektlaufzeit (*dynamisches* Tailoring). Davon unberührt bleibt allerdings der Kernbestand an Vorgehensbausteinen, die für jedes V-Modell-konforme Projekt obligatorisch sind.

Einordnung des V-Modells

Zusammenfassend stellt das V-Modell ein sehr gut ausgearbeitetes und umfassendes Modell zur Verfügung, das durch die Möglichkeiten des Tailoring und die Berücksichtigung unterschiedlicher Projektarten flexibel einsetzbar ist. Es ermöglicht eine standardisierte Abwicklung von Projekten und trägt damit den oben genannten Zielen Rechnung. Das Modell ist frei zugänglich, sehr gut dokumentiert und stellt den Anwendern unterstützende Werkzeuge und Produktvorlagen zur Verfügung. Wir werden daher im Laufe des Kurses noch an mehreren Stellen auf das V-Modell zurückkommen.

Als Schwachstelle des V-Modells bleibt in der Hauptsache der Umfang des Modells und die Vielzahl der erzeugten Dokumente, die trotz der Möglichkeiten des Tailoring deutlich machen, dass das V-Modell letztlich doch in erster Linie für große eingebettete Systeme entwickelt worden ist. Trotz aller aufwändigen Formalismen ist das V-Modell aber auch als Vorlage für kleinere Projekte durchaus zu empfehlen. Dabei kommt es dann auf eine pragmatische Anwendung an, mit der ein unverhältnismäßiger bürokratischer Aufwand vermieden werden kann.

1.6.9 Rational Unified Process

Vorgehensmodell zur UML

Mit dem *Rational Unified Process* (*RUP*) /Kruchten 1999/ wurde versucht, nach der Standardisierung objektorientierter Beschreibungsmethoden in der *Unified Modelling Language* (*UML*) /Kecher 2011/ auch einen Rahmen für das Vorgehen bei Softwareprojekten vorzugeben. Hinter dem *RUP* steht dabei ein Werkzeug, das den Prozess unter anderem durch Vorgabe zahlreicher *Templates* (Schablonen, Muster) unterstützt. Wir wollen uns an dieser Stelle aber auf das Vorgehensmodell konzentrieren, das durch dieses Werkzeug unterstützt werden soll.

Konzepte

Der Rational Unified Process versucht nach /Rational 2001/ sechs „in der Praxis bewährte“ Konzepte umzusetzen:

1. *die iterative Softwareentwicklung*

   evolutionäre Softwareentwicklung

   Der Rational Unified Process nimmt Elemente der evolutionären Softwareentwicklung und des Spiralmodells auf, um so der Tatsache Rechnung zu tragen, dass große Projekte nicht in einem Schritt entwickelt werden können. Damit soll ein im Projektverlauf zunehmendes Verständnis des Problembereichs durch schrittweise Verfeinerung des Systems unterstützt werden. Im Mittelpunkt der Überlegungen steht die Reduzierung des Projektrisikos durch eine Aufteilung in überschaubare Abschnitte, die jeweils zu einer lauffähigen Version des Systems im Sinne des evolutionären Prototypings (vgl. hierzu die Ausführungen in Abschnitt 1.6.6) führen.

2. *das Management der Anforderungen*

   Methode der *Use Cases*

   Das vorgeschlagene Modell unterstützt die Erarbeitung, Organisation und Dokumentation der benötigten Funktionalität und der zu beachtenden Rahmenbedingungen. Hierzu wird im Wesentlichen auf die Methode der *Use Cases* zurückgegriffen, mit deren Hilfe die Abwicklung der Geschäftsvorfälle durch das System modelliert wird. Dabei ist vorgesehen, diese *Use Cases* während des Projektablaufs immer wieder zur Qualitätssicherung heranzuziehen, um die Erfüllung der Anforderungen durch das System zu gewährleisten. So kann z.B. überprüft werden, ob in einem Systementwurf auch wirklich alle fachlichen Anforderungen abgedeckt werden.

3. *die Verwendung komponentenbasierter Architekturen*

frühzeitige Entwicklung einer Systemarchitektur

Um die Verwendung vorgefertigter Komponenten zu unterstützen, sieht der Rational Unified Process die frühzeitige Entwicklung einer Systemarchitektur vor, die auf die Einsatzmöglichkeiten wiederverwendbarer Komponenten hin untersucht werden kann.

4. *die visuelle Modellierung der Software*

Um die Lesbarkeit und Übersichtlichkeit der Dokumente zu fördern, nutzt der Rational Unified Process im Wesentlichen grafische Notationen für die zu erstellenden Dokumente. Dabei handelt es sich – naheliegenderweise – um die in der UML vorgegebenen Notationen.

5. *die Qualitätssicherung*

Integration von Qualitätssicherungsaktivitäten in allen Teilaktivitäten

Der Notwendigkeit einer umfassenden Qualitätssicherung trägt der Rational Unified Process durch die Integration von Qualitätssicherungsaktivitäten in allen Teilaktivitäten des Prozesses Rechnung. Dabei wird besonderer Wert darauf gelegt, dass Qualitätssicherung keine nachrangige Angelegenheit, sondern integraler Bestandteil des Prozesses sein muss.

6. *die Kontrolle der Änderungen der Software*

Änderungen nachvollziehbar machen

Dieser Aspekt des Rational Unified Process umfasst Aktivitäten des Konfigurationsmanagements. Es geht darum, Änderungen von Anforderungen, Dokumenten, Programmen etc. nachvollziehbar zu machen und den Zugriff auf aktuelle und zurückliegende Dokumente zu ermöglichen.

zwei Dimensionen

Das eigentliche Vorgehensmodell – das sich in gewisser Weise als die Essenz vorhergehender Ansätze auf Basis der UML versteht – betrachtet den Ablauf des Projekts in zwei Dimensionen. Zum einen gibt es die zeitliche Dimension, die in die vier Phasen *Konzeptphase* (*Inception phase*), *Ausarbeitung* (*Elaboration phase*), *Konstruktion* (*Construction phase*) und *Übergang in den Betrieb* (*Transition phase*) unterteilt wird [8]. In der zweiten Dimension werden die Arbeitsabläufe (Workflows) des Projekts, unterteilt in die wichtigsten Kernabläufe und die wichtigsten unterstützenden Abläufe, betrachtet. Abbildung 25 veranschaulicht diese Sicht eines Projekts. Wir haben dabei absichtlich die englischen Originalbegriffe verwendet, weil sie uns präziser erscheinen.

Zyklus

Bei der Interpretation von Abbildung 25 ist zu beachten, dass der gesamte dargestellte zeitliche Ablauf von *Inception* bis *Transition* einem *Zyklus* im Sinne des evolutionären Vorgehens entspricht. Jede neue Version des Systems wird also nach diesem Modell entwickelt. Die im unteren Teil der Abbildung angegebenen

---

8 In /Kruchten 1999/ werden zum Teil etwas andere Übersetzungen der englischen Orginalbegriffe verwendet. So wird dort *inception phase* mit *Konzeptualisierung* und *elaboration phase* mit *Entwurf* übersetzt.

Iterationen können so interpretiert werden, dass jede Iteration ein wohldefiniertes Zwischenergebnis liefert, das Gegenstand der Qualitätssicherung sein muss.

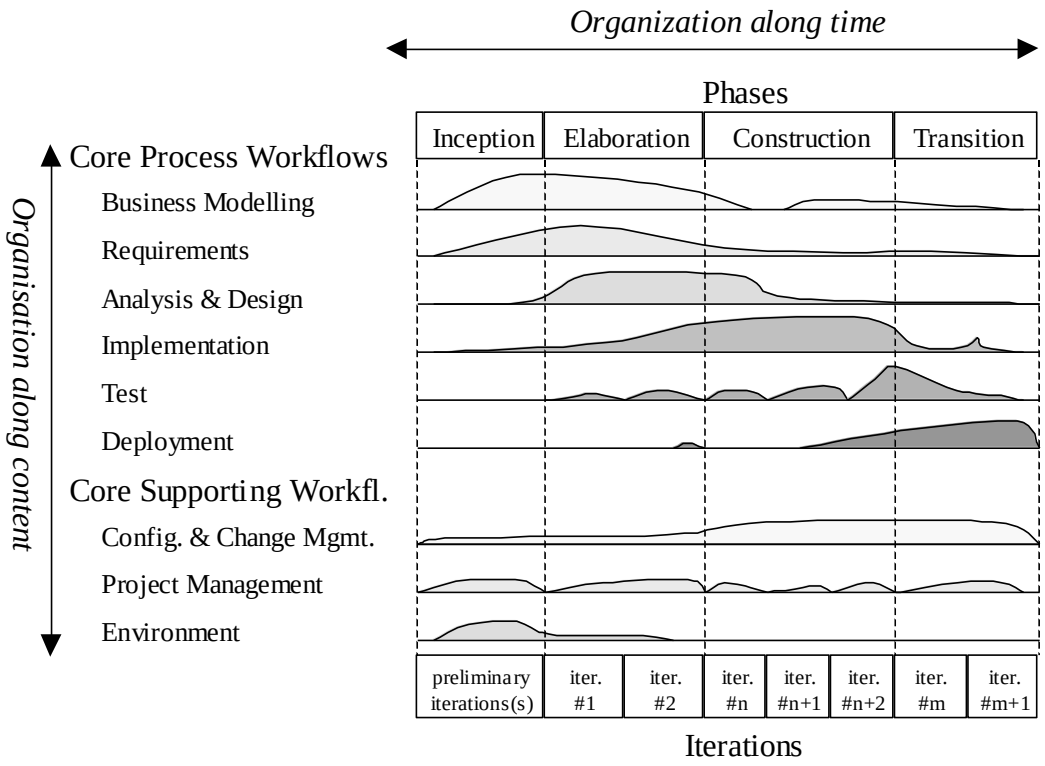


Abbildung 25: Überblick über die Phasen und Aktivitäten im Rational Unified Process (nach /Rational 2001/ S. 3)

Wie wollen uns nun zunächst mit den zeitlichen Phasen (horizontale Untergliederung) und dann mit den Aktivitäten (vertikale Untergliederung) beschäftigen.

zeitliche Dimension

Die vier Phasen *Inception*, *Elaboration*, *Construction* und *Transition* werden jeweils durch einen sogenannten Meilenstein abgeschlossen. Ein Meilenstein bildet dabei einen Zeitpunkt, an dem aufgrund des Erreichten eine Entsch eidung über den Projektfortgang getroffen werden muss. Dies ist im Prinzip mit der vierten Teilphase (unterer, linker Quadrant) des Spiralmodells vergleichbar[9].

9 Kruchten gibt in seinem Referenzwerk zum Rational Unified Process /Kruchten 1999/ zum Begriff des Meilensteins die Definition: „Ein Zeitpunkt, zu dem eine Iteration beendet wird; steht im Zusammenhang mit einem Release.“ Damit bezieht er die Definition eines Meilensteins auf die Begriffe Iteration („Eine bestimmte Folge von Aktivitäten, die mit einem Release endet.“) und Release („Ein Teil des Endprodukts, der Gegenstand der Prüfung an einem wichtigen Meilenstein ist.“).

Im Einzelnen können die Phasen wie folgt beschrieben werden:

- *Konzeptphase* (*Inception phase*)

  Während dieser Phase werden die fachlichen Anforderungen an das System erhoben und der Umfang des Projekts definiert. Dabei werden alle systemexternen Stellen identifiziert, die mit dem System interagieren müssen. Ferner werden alle Geschäftsvorfälle, die das System betreffen, identifiziert und die wichtigsten von ihnen ausführlich beschrieben. Hinzu kommt eine Ermittlung der kritischen Erfolgsfaktoren für das System, eine Auflistung der möglichen Projektrisiken und eine erste Aufwandsschätzung, verbunden mit einem ersten Terminplan.

  In dem dieser Phase zugeordneten Meilenstein geht es in erster Linie darum, zu überprüfen, ob die ermittelten Anforderungen die Wünsche des Auftraggebers korrekt wiedergeben und ob der Kosten- und Terminplan realistisch ist.

- *Ausarbeitung* (*Elaboration phase*)

  Diese Phase dient der Fortführung der Analyse des Problembereichs, der Erarbeitung einer groben Systemarchitektur, der Entwicklung eines Projektplans und der Eliminierung der bedeutendsten Projektrisiken (analog zur Risikominimierung im Spiralmodell z.B. durch Simulation, Benchmarking etc.). Hierzu ist ein umfassendes und tiefes Verständnis des zu entwickelnden Systems hinsichtlich seines Umfangs, seiner Funktionalität, der nicht-funktionalen Anforderungen (Qualität, Zuverlässigkeit etc.) und der Performanzanforderungen erforderlich.

  Am Ende dieser Phase liegen alle Informationen über das zu erstellende System vor. Bis zu diesem Zeitpunkt können Änderungswünsche noch mit vertretbarem Aufwand berücksichtigt werden. Spätestens nach Ende dieser Phase sollten die Anforderungen aber stabil sein.

evolutionäre Systementwicklung

  Im Sinne der objektorientierten Systementwicklung sollte am Ende der Ausarbeitung „ein ausführbarer Prototyp der Architektur“ stehen (/Kruchten 1999/, S. 70). Dieser Prototyp hat als oberstes Ziel die Eingrenzung der technischen Risiken. Zu diesem Zweck wird empfohlen, im Sinne eines evolutionären Prototyping zunächst die wichtigsten Geschäftsvorfälle zu betrachten. Obwohl eine evolutionäre Systementwicklung vorgezogen wird, werden auch Wegwerf-Prototypen nicht explizit ausgeschlossen. Die evolutionäre Entwicklung auch innerhalb der einzelnen Phasen wird in Abbildung 25 durch die Unterteilung der einzelnen Phasen in mehrere Iterationen in der untersten Zeile dargestellt.

Auch diese Phase wird mit einem Meilenstein abgeschlossen, in dem unter anderem die Erreichung der Systemziele sowie die Eingrenzung der wesentlichen Projektrisiken adressiert werden.

- *Konstruktion* (*Construction phase*)

  In dieser Phase geht es um die Erstellung des Systems. Dies beinhaltet – im Sinne einer Weiterentwicklung des in der vorherigen Phase entwickelten Prototyps – die Implementierung der noch ausstehenden Komponenten und fachlichen Funktionalitäten. Diese werden dann zum vollständigen System integriert und sorgfältig getestet. Bei der Implementierung wird ausdrücklich ein paralleles Arbeiten an den verschiedenen Komponenten mit dem Ziel der Verkürzung der Projektlaufzeit in Betracht gezogen.

  Am Ende der Phase steht damit eine „Beta-Version“ des Systems. Im abschließenden Meilenstein ist zu prüfen, ob dieses System für den Übergang in die Einsatzphase geeignet ist.

  „Beta-Version“ des Systems

- *Übergang in den Betrieb* (*Transition phase*)

  In dieser Phase wird das System beim Endanwender in Betrieb genommen. Dabei kann zunächst ein „Beta-Test“ mit einer ausgewählten Nutzergruppe durchgeführt werden.

  Wichtig ist für den Übergang in den Betrieb vor allem auch die Benutzerdokumentation zum System, die Schulung der Anwender und deren Unterstützung (z.B. durch ein Support-Center). Wichtig ist dabei ferner, dass man gerade in dieser Phase darauf vorbereitet ist, relativ schnell auf Probleme der Anwender zu reagieren. Dies sollte sich aber im Wesentlichen auf Produktdetails beschränken und nicht generelle Aspekte der Systemkonzeption betreffen.

  Der Meilenstein zu dieser Phase kann schon auf den ersten Einsatzerfahrungen mit dem System aufsetzen. Es geht hier darum, eine abschließende Betrachtung des zurückliegenden Zyklus vorzunehmen (z.B. die Nachkalkulation) und über den Eintritt in einen neuen Zyklus zur Erstellung einer neuen Systemversion zu entscheiden.

geschachtelte Zyklen

Wichtig zum Verständnis des Prozesses ist dabei, dass der in Abbildung 25 dargestellte zeitliche (horizontale) Ablauf ein Zyklus in einem umfassenderen evolutionären Vorgehen sein kann und sollte. Ein Zyklus hat gewöhnlich eine Version des Systems als Ergebnis. Außerdem kann und soll aber auch innerhalb der einzelnen Phasen wieder evolutionär gearbeitet werden, d.h. auch die einzelnen Phasenergebnisse werden in mehreren Iterationen (über die Arbeitsabläufe hinweg) evolutionär entwickelt.

Aktivitäten des Entwicklungsprozesses

Dem zeitlichen Ablauf stehen auf der anderen Seite (vertikale Untergliederung in Abbildung 25) die Arbeitsabläufe (Workflows) gegenüber. Für die wichtigsten Kernabläufe (*Core Process Workflows*) und die wichtigsten unterstützenden Abläufe (*Core Supporting Workflows*) ist in Abbildung 25 der zeitliche Arbeitsanfall als schattierte „Gebirgszüge“ dargestellt.

Die Aufteilung in eine zeitliche Dimension einerseits und inhaltlich definierte Arbeitsabläufe andererseits geht dabei auf die unter anderem bereits bei Boehm /Boehm 1981/ festgestellte Tatsache zurück, dass im Prinzip in allen Phasen eines Projekts fast alle Arten von Tätigkeiten anfallen können. So kann während der Analyse eine Implementierungstätigkeit bei der Realisierung eines Prototyps anfallen oder während der Implementierungsphase eine Analysetätigkeit, um erst jetzt deutlich gewordene offene Analysefragen nachträglich zu klären.

Im Rational Unified Process erfüllen die Arbeitsabläufe (*Workflows*) folgende Aufgaben:

- *Geschäftsprozessmodellierung (Business Modelling)*

  In diesem Arbeitsablauf werden die für das System relevanten Geschäftsprozesse in sogenannten *Use Cases* (vgl. hierzu die Literatur zur UML, wie z.B. /Kecher 2011/) dokumentiert, um ein gemeinsames Verständnis des Projektumfelds zwischen Systementwicklern und Anwendern zu gewährleisten.

- *Anforderungen (Requirements)*

  Hier geht es darum zu ermitteln, was das System leisten soll. Dazu sind sowohl die funktionalen Anforderungen als auch die gegebenen Rahmenbedingungen zu dokumentieren.

- *Analyse und Design*

  Dieser Arbeitsablauf beschäftigt sich mit der Frage, wie das System realisiert werden soll. Dazu wird ein Designmodell zur Festlegung der Systemarchitektur und optional ein Analysemodell erstellt. Wichtig ist insbesondere, dass sich sowohl die fachlichen Anforderungen als auch die zu beachtenden Rahmenbedingungen im Designmodell wiederfinden lassen.

- *Implementierung*

  Hier sind die einzelnen Bausteine (Klassen) zu entwickeln und zu integrieren. Dabei wird ein besonderes Augenmerk auf die komponentenbasierte Implementierung und auf Aspekte der Wiederverwendung gelegt.

- *Test*

  Durch die Testabläufe sollen sowohl das Zusammenspiel der Komponenten als auch die Erfüllung der für das System definierten Anforderungen überprüft werden.

- *Inbetriebnahme (Deployment)*

  Die Arbeitsabläufe zur Inbetriebnahme des Systems umfassen alle Aktivitäten, die sich mit der Schulung und Unterstützung der Benutzer, mit der Zusammenstellung und Installation der Software und ggf. auch mit Beta-Tests und Migrationsproblemen bei der Ablösung von Altsystemen beschäftigen.

- *Projektmanagement*

  Der Projektmanagement-Arbeitsablauf ist der erste querschnittlich unterstützende Arbeitsablauf. Seine Hauptaufgabe wird im Rational Unified Process „in der Kunst gesehen, zwischen konkurrierenden Zielen zu vermitteln, Risiken zu managen und Rahmenbedingungen zu überwinden, um ein Produkt zu entwickeln, das den Bedürfnissen der Auftraggeber (derjenigen also, die das Geld geben) und der Benutzer (derjenigen also, die mit dem System arbeiten müssen) Rechnung trägt“ /Rational 2001/.

  querschnittlich unterstützender Arbeitsablauf

- *Konfigurationsverwaltung (Configuration- & Change Management)*

  In diesem Arbeitsablauf geht es primär um die Verwaltung der im Projekt erstellten Dokumente, die ja im Allgemeinen von mehreren Personen gemeinsam bearbeitet werden. Dabei geht es neben der Gewährleistung von Konsistenz und der Möglichkeit des Rückgriffs auf ältere Versionen auch um die Benachrichtigung Betroffener bei Änderungen in Dokumenten.

  Neben der Konfigurationsverwaltung ist in diesem Arbeitsablauf auch die Verwaltung von Änderungswünschen angesiedelt.

- *Umgebung (Environment)*

  Dieser Arbeitsablauf beschäftigt sich im Wesentlichen mit der Bereitstellung der Software-Entwicklungsumgebung. Hierzu gehört sowohl die Bereitstellung der Hardware und der Software als auch eine entsprechende Unterstützung der Projektmitarbeiter.

Zusammenfassend kann der *Rational Unified Process* als Kombination eines klassischen Wasserfallmodells mit Elementen des Spiralmodells und insbesondere des evolutionären Prototyping charakterisiert werden. Der Rational Unified Process stützt sich dabei auf die *Unified Modelling Language (UML)* und scheint im Verhältnis zum V-Modell eher pragmatisch angelegt zu sein.

1.6.10 eXtreme Programming

Ende der 90er Jahre des 20. Jhs. wurde von Kent Beck – sicherlich auch als Reaktion auf die nicht immer eingehaltenen Versprechen des klassischen Software Engineering und Software-Projektmanagements – der Ansatz des *eXtreme Programming* (/Beck 1999/) vorgeschlagen. Sein umwälzender Neuansatz wurde zum Teil begeistert aufgenommen. Wie der Name bereits andeutet, handelt es sich dabei um einen stark auf die Implementierungsaktivitäten konzentrierten Ansatz, der mit einigen Annahmen und Vorstellungen des klassischen Software Engineering bricht.

Extreme Programming (auch als *XP* für *eXtreme Programming* bezeichnet) ist ein Ansatz, der in einem evolutionären Prozess in kleinen Schritten – sogenannten Releases – die Entwicklung vorantreibt. Man kann dabei auch von einer Beschränkung der Aktivitäten auf das Wesentliche sprechen. Der Ansatz wurde von Kent Beck in der 2. Auflage seines Buches 2004 /Beck, Andres 2004/ noch einmal weiterentwickelt und ergänzt, ohne die Grundprinzipien zu verändern. Man spricht seitdem aber häufig auch von *XP2*.

Welche Kosten verursachen Änderungen?

Das Extreme Programming geht davon aus, dass die Änderungskosten höchstens logarithmisch mit der Zeit steigen, die zwischen der Entwicklung eines Programms und seiner Änderung liegt. Demnach unterscheiden sich die Kosten für Programmanpassungen – seien sie durch sich ändernde Anforderungen der Anwender oder durch Korrekturen von Programmfehlern verursacht –, die frühzeitig oder sehr spät in einem Projektverlauf erfolgen, nicht signifikant. Dies steht im Widerspruch zur gängigen Lehrmeinung, die davon ausgeht, dass der Änderungsaufwand mit der Zeit stark anwächst. Die Begründung für die klassische These von im Zeitverlauf stark ansteigenden Kosten für Änderungen ist grob gesprochen, dass Entscheidungen, die länger zurückliegen, zwischenzeitlich zahlreiche Folgewirkungen im System hatten, die bei einer späteren Änderung berücksichtigt werden müssen.

Die wesentlichen Arbeitsschritte eines Projekts beschränken sich beim Extreme Programming auf die Planung einzelner Releases, Iterationen mit Programmierarbeiten sowie Abnahme-Tests für die einzelnen Releases. Das Entwicklerteam konzentriert sich ganz auf das eigentliche Programmieren, indem es auf umfassende Analysen verzichtet. Dahinter steckt das Ziel, einzelne Software-Releases möglichst schnell zu erzeugen. Typische Release-Intervalle betragen nur ein bis drei Monate. Hieraus soll ein zweifacher Nutzen für das Projekt erwachsen. Zum einen erhält der Auftraggeber bereits sehr früh ein Produkt, das seinen Verwendungszweck zumindest zum Teil erfüllt. Zum anderen erhalten die Anwender sehr früh die Gelegenheit, den Entwicklern aufgrund erster Erfahrungen mit dem System Kommentare und Änderungswünsche mitzuteilen.

Analogien zum Prototyping

In diesem Punkt weist XP Analogien zur evolutionären Softwareentwicklung auf, die hier ins Extreme getrieben wird (vgl. Abbildung 19 auf Seite 44). Neben As-

pekten der evolutionären Softwareentwicklung finden sich in XP aber auch Aspekte des Prototyping zur Absicherung von Analyseergebnissen (vgl. Abbildung 18 auf Seite 43). Wo eine Voraussetzung oder eine Anforderung nicht klar ist, sieht XP die Entwicklung eines „Spike“ vor. Ein Spike ist ein einfaches Programm, das ohne Rücksicht auf andere Randbedingungen nur eine ganz spezielle Aufgabe behandelt. Solche Spikes gelten von vorneherein als Wegwerfprogramme. Sie sollen lediglich dabei helfen, das zu Grunde liegende Problem auszuloten und Lösungsmöglichkeiten zu erproben.

Praktiken in XP

Eine Charakterisierung des XP kann am besten durch die in XP eingesetzten Techniken erfolgen, die in der XP-Terminologie *Praktiken* heißen. Wir wollen diese Praktiken im Folgenden in Anlehnung an /Reißing 2000/ und /Elting, Huber 2001/ genauer betrachten. Die Praktiken sind dabei nur in ihrer Gesamtheit sinnvoll. Das Weglassen einzelner Praktiken stellt das gesamte Vorgehen in Frage, weil sich die Praktiken gegenseitig unterstützen. Keine der Praktiken ist dabei wirklich neu – so werden u.a. Ideen des Spiralmodells und des Prototyping aufgegriffen –, neu ist aber die Kombination und die „extreme“ Interpretation, in der diese Praktiken im XP verwendet werden.

- *Kleine Releases*

  Wie oben bereits erwähnt, besteht eine erste Kernidee des XP darin, einzelne Software-Releases möglichst schnell zu erzeugen. Damit soll den sich ständig ändernden Anforderungen in einem Software-Projekt Rechnung getragen werden. Hierbei wird ein Release-Zyklus von ein bis drei Monaten angestrebt. Ein Release besteht seinerseits aus mehreren Iterationen von ein bis drei Wochen. Die Iterationen zerfallen in Arbeitspakete von ein bis drei Tagen. Der Kunde kann dadurch bereits sehr früh Abweichungen von seinen Anforderungen feststellen.

- *Planspiel (Planning Game)*

  Die Planung der einzelnen Releases wird in XP als Planspiel (Planning Game) bezeichnet. Es geht dabei darum, für die kommende Iteration die verfügbaren Entwicklungsressourcen und die Kundenwünsche in Einklang zu bringen und zu entscheiden, welche der wünschenswerten Funktionen realisiert werden können. Die Anforderungen werden vom Kunden in Form sogenannter *User Storys* beschrieben. Dabei handelt es sich um einfache Anwendungsfälle, die auf eine sogenannte Indexkarte geschrieben werden. Der Kunde versieht diese Storys mit Prioritäten. Er legt fest, welche Storys in der kommenden Iteration umgesetzt werden sollten und wann die Iteration abgeschlossen werden sollte. Die Entwickler geben dann den Aufwand an, den sie für die Realisierung eines bestimmten Features erwarten. Im Allgemeinen wird der Aufwand die verfügbaren Ressourcen übersteigen, sodass der Kunde einige Storys auf spätere Iteratio-

nen verschieben muss. Dieses Verfahren wird wiederholt, bis ein Gleichgewicht erreicht ist.

- *Tests*

  Eine der zentralen Praktiken von XP ist das automatische Testen. Die Testfälle, die unbedingt vor der Implementierung der entsprechenden Features festgelegt werden müssen, repräsentieren neben den User Storys die einzige Festlegung der gewünschten Funktionalität. Ein Programmfeature, für das kein Test existiert, ist damit praktisch nicht vorhanden. Wichtig ist, dass die Tests automatisch ausgeführt werden können. Sie werden dazu in einem Repository gesammelt. Jedes Mal, wenn ein Programmierer eine Änderung am System vornimmt, ist er verpflichtet, alle vorhandenen Tests erneut ablaufen zu lassen, um die Funktionstüchtigkeit des Systems zu gewährleisten. Die Entwickler selbst schreiben dabei Tests für Klassen, Methoden und Komponenten. Der Kunde entwickelt funktionale Testfälle für die User Storys, die von einem Tester in Tests umgesetzt werden.

- *Systemmetapher*

  Statt der in herkömmlichen Vorgehensmodellen vorgesehenen Designaktivitäten sieht XP die Verwendung sogenannter Metaphern vor, damit alle Projektbeteiligten ein gemeinsames Verständnis für die Zusammenhänge der Systemkomponenten haben. Das Metapherkonstrukt ersetzt dabei eine Softwarearchitektur. Die Systemmetapher sollte sowohl für die Entwickler als auch für den Kunden verständlich sein. Das Konzept der Systemmetapher bleibt dabei aber letztlich (absichtlich ?) recht schwammig.

- *Einfacher Entwurf*

  Die wichtigste Regel bei der Software-Entwicklung mit XP ist *„to do the simplest thing that could possibly work“* /Jeffries 2001/. Die Einfachheit der Lösungen wird als wichtiger Schritt auf dem Weg zu einer schnellen Lösung angesehen. Dem liegt auch die These zugrunde, dass ein Bemühen um technisch perfekte, allgemein wiederverwendbare, zukünftige Erweiterungen antizipierende Lösungen sinnlos ist, weil sich die Anforderungen bis dahin ohnehin geändert haben werden. Sollten später Änderungen oder Erweiterungen erforderlich sein, wird das System – wie es im XP-Jargon heißt – zunächst *refaktorisiert*.

- *Refaktorisierung*

  Eine Refaktorisierung dient der Vereinfachung des Entwurfs eines bestehenden Systems unter Beibehaltung der Semantik. Die Refaktorisierung zielt dabei darauf ab, die Verständlichkeit und Änderbarkeit des Codes zu verbessern. Zu diesem Zweck können entsprechende Änderungen am Pro-

grammcode vorgenommen werden, wie z.B. das Zusammenziehen von Funktionen in einer neuen Klasse oder Ähnliches. Refaktorisierungen sollen allerdings nicht aufgrund von Spekulationen über zukünftige Anforderungen vorgenommen werden. Vielmehr sollen sie aus der konkreten Situation heraus erfolgen, in der sie sinnvoll und hilfreich erscheinen. Wichtig ist dabei aber, dass hier auch Aufwand erlaubt ist, der nicht direkt zur Erweiterung der Funktionalität des Systems beiträgt. Eine vorgenommene Refaktorisierung muss natürlich sofort durch ein erneutes Ausführen der Tests überprüft werden.

- *Dokumentation*

  Im Hinblick auf die Dokumentation des Systems setzt XP auf Programmcode, der in hohem Maße selbsterklärend ist. Dazu ist auf eine entsprechende Programmstruktur und eine sinnvolle Namensgebung zu achten. Wenn ein Programm an einer Stelle eines erklärenden Kommentars bedarf, sollte es refaktorisiert werden. Hierzu bietet sich z.B. die Auslagerung von Programmteilen in sinnvoll benannte Unterprozeduren an. Die Motivation, auf eine zusätzliche Kommentierung und Dokumentation zu verzichten, ergibt sich aus der Beobachtung, dass ein großer Teil der zusätzlichen Dokumentation mangels laufender Anpassung sehr schnell veraltet und daher unbrauchbar oder sogar irreführend ist.

- *Programmieren in Zweierteams*

  Die Programmierung geschieht bei XP immer in Zweierteams mit einer klaren Rollenverteilung. Der erste Partner kümmert sich um die aktuelle Kodierung, während der zweite Partner den Code auf Schreibfehler und logische Fehler überprüft und Strategien für weitere Implementierungen entwickelt. Auf diese Weise werden alle Arbeiten einer kontinuierlichen Qualitätssicherung unterzogen. Dabei ist auch ein Rollentausch zwischen den beiden Entwicklern im Bedarfsfall vorgesehen. Die Paarbildung erfolgt dynamisch: man sucht für jedes Arbeitspaket neu geeignete Partner aus. Ein Effekt dieser dynamischen Paarbildung ist, dass sich das Wissen über die verschiedenen Bereiche des Systems auf alle Entwickler verbreitet. Damit sind auch z.B. durch Krankheit bedingte Ausfälle einzelner Entwickler relativ problemlos zu kompensieren.

- *Gemeinsames Code-Eigentum*

  Der Entwickler bzw. das Paar, das einen bestimmten Programmcode erarbeitet hat, hat an diesem Code keine besonderen Rechte. Programmcode gehört in XP grundsätzlich allen Entwicklern gemeinsam. Das bedeutet, dass jedes Entwicklerpaar jederzeit überall Änderungen vornehmen darf. Der Gefahr, dass diese Änderungen fehlerhaft sind, wird zum einen durch den selbstdokumentierenden Code und zum anderen durch das selbstver-

ständlich nach den Änderungen erforderliche Ausführen der Tests entgegengewirkt.

- *Kontinuierliche Code-Integration*

  Um neu entwickelte Programmteile möglichst schnell in die bestehende Umgebung einzubinden, ist in XP eine kontinuierliche Integration des Programmcodes in das System vorgesehen. Neu entwickelte oder geänderte Programmteile sollen dazu bereits nach wenigen Stunden wieder in die aktuelle Code-Basis integriert werden. Dazu wird ein Integrations- und Testsystem verwendet, auf dem nach dem Einspielen der Änderungen selbstverständlich wieder alle Testfälle überprüft werden müssen. Das Team, das die Änderungen einspielt, ist dafür verantwortlich, eventuell auftretende Fehler zu beseitigen.

- *Geregelte Arbeitszeiten*

  Weil das Entwickeln im Paar sehr hohe Anforderungen an die Konzentration und die Leistungsfähigkeit der Mitarbeiter stellt, ist unbedingt auf geregelte Arbeitszeiten zu achten. Überstunden zeugen von falschem Projektmanagement und sind zu vermeiden.

- *Kundenvertreter im Team*

  XP sieht die Aufnahme eines Kundenvertreters in das Projektteam vor, der insbesondere im Zusammenhang mit den Planspielen die Sicht des Kunden vertreten soll. Damit wird der Tatsache Rechnung getragen, dass XP keine echte Spezifikation der Kundenanforderungen enthält. Der Kundenvertreter im Entwicklerteam sollte dabei ein zukünftiger Anwender des Systems sein, der auch für die Definition der funktionalen Testfälle verantwortlich ist.

- *Programmierrichtlinien*

  Sowohl das Arbeiten in Paaren als auch das gemeinsame Code-Eigentum machen es erforderlich, dass Programmcode allgemein verständlich ist. Hierzu werden in XP feste Programmierrichtlinien verwendet, die von den Entwicklern untereinander vereinbart werden. Dadurch erhält man einheitlichen Code, der von allen Entwicklern verstanden und leicht modifiziert werden kann.

Zusammenhänge zwischen den Praktiken

Die einzelnen beschriebenen Praktiken im XP hängen stark voneinander ab. So begünstigt die strikte Einhaltung von Programmierrichtlinien das Programmieren in Paaren, das gemeinsame Code-Eigentum und die kontinuierliche Code-Integration. Kurze Release-Zyklen sowie die Beschränkung der Analyse und Definition auf das Planspiel und die Testfälle machen einen Kundenvertreter im

Team erforderlich. Diese engen Zusammenhänge führen dazu, dass XP als Ganzes gesehen werden muss. Ein Aufgreifen einzelner Ideen ohne Berücksichtigung der anderen Praktiken erscheint nicht sinnvoll.

Voraussetzungen von XP

Wichtig ist dabei, die Voraussetzungen zu sehen, auf denen XP aufbaut. Neben der Annahme, dass die Änderungskosten höchstens logarithmisch mit der Zeit steigen, sind hier insbesondere folgende Aspekte zu nennen:

- Die Entwickler sollten alle über eine sehr gute und breite Qualifikation verfügen, da sie laufend in anderen Paaren und mit anderen Aufgaben eingesetzt werden. Insbesondere sehr inhomogene Paare aus einem erfahrenen Mitarbeiter und einem „Anfänger“ passen kaum in das Bild von XP.

- Alle am Projekt Beteiligten (Management, Teammitglieder, Auftraggeber und Anwender) müssen sich auf XP-Praktiken einlassen, und der Kunde muss in der Lage sein, einen qualifizierten Mitarbeiter für das Projektteam abzustellen. Dabei ist auch die problematische Vertragsgestaltung zwischen Auftraggeber und Softwareentwickler zu beachten. Aufgrund der kurzen Planungszeiträume und der sich erst im Laufe des Projekts ergebenden Systemanforderungen kann hier kein Vertrag abgeschlossen werden, der allen Beteiligten Planungssicherheit gibt.

- XP ist für kleine Projekte mit 10 bis 15 Entwicklern gedacht und typischerweise in einem Umfeld mit sich rasch ändernden Anforderungen sinnvoll.

- Die ganze Vorgehensweise im XP-Ansatz ist stark auf Kommunikation ausgerichtet und damit am besten umsetzbar, wenn die Entwickler an einem Ort konzentriert sind und die gleichen Arbeitszeiten haben.

- Da die Tests sehr häufig ausgeführt werden müssen, sollten sie weder zu umfangreich noch zu laufzeitintensiv sein. Andererseits müssen sie aber die gesamte Funktionalität des gewünschten Systems abdecken.

Einordnung von XP

Zusammenfassend wirft der Ansatz des Extreme Programming zahlreiche Fragen auf. Neben dem Fehlen einer sauberen Anforderungsdefinition ist dabei auch das relativ knappe Qualitätsmanagement und die ungenügende Orientierung der Entwicklungstätigkeiten an den Anforderungen des Gesamtsystems kritisch zu hinterfragen.

Dabei erscheint es durchaus plausibel, dass XP für Teams mit erfahrenen und hochqualifizierten Mitarbeitern bei nicht zu großen Projekten erfolgreich eingesetzt werden kann. Dies ist letztlich sogar wahrscheinlich, weil erfahrene und hochqualifizierte Mitarbeiter vermutlich mit fast jedem Vorgehensmodell Erfolg haben werden. Denn die Erfahrung mit zahlreichen Projekten zeigt, dass es „gar nicht so wichtig [ist], mit welcher Methodik man ein Projekt abwickelt. Primär

kommt es darauf an, überhaupt ein Vorgehensmodell zu verwenden und dieses auch hinreichend gut zu verstehen“, wie Elting und Huber /Elting, Huber 2001/ es formulierten. XP hat dabei als extreme Variante des evolutionären Prototyping zahlreiche innovative Ansätze zu bieten.

1.6.11 Scrum

In der kritischen Distanzierung von klassischen Vorgehensmodellen der Softwareentwicklung nimmt das Extreme Programming Ideen der sogenannten *agilen Softwareentwicklung* auf. Kent Beck gehörte selbst zu den Unterzeichnern des *Manifesto for Agile Software Development*, das 2001 veröffentlicht wurde und das zu einer „leichteren“, schlankeren sowie strikt am Kunden und seinen Zielen orientierten Programmentwicklung führen soll.

Ein sehr bekanntes und weit verbreitetes Vorgehensmodell (um in unserer Terminologie zu bleiben), das den Werten und Prinzipien der agilen Softwareentwicklung folgt, ist *Scrum*. Konzeptionell vor allem von Jeff Sutherland und Ken Schwaber in den 90er Jahren des 20. Jhs. entwickelt und empirisch erprobt, erschien 2001 das Grundlagenwerk *Agile Software Development mit Scrum* /Schwaber, Beedle 2001/, mit dem sich Scrum weltweit durchsetzen sollte. Heute ist Scrum in Tausenden Unternehmen implementiert; es gibt eine sehr rege Scrum Community, dedizierte Interessenorganisationen wie die Scrum Alliance und Scrum.org, internationale Ausbildungsstrukturen sowie Scrum-Zertifizierungen usf. Das Ziel von Scrum ist eine schnelle, inkrementelle und kundenorientierte Produktentwicklung unter Verzicht auf umfassende vorgängige Planung (Überblicksdarstellungen zu Scrum z.B. in: /Sutherland 2010/; /Wintersteiger 2013/; /Gloger 2013/).

Begriff „Scrum“

Der Begriff *Scrum* entstammt der Sportsprache. Im Rugby bezeichnet er das kreisförmige „Gedränge“ zweier Mannschaften beim Neustart eines Spiels. Das Vorgehensmodell Scrum nimmt aus diesem Bild vor allem den Teamgedanken und die zentrale Fragestellung, wie sich Entwicklungsteams organisieren sollten. Zugleich markiert es schon begrifflich die Abkehr vom Wasserfallmodell mit seiner linearen Prozessmodellierung.

Das Vorgehensmodell von Scrum kennt *Rollen*, *Events* und *Artefakte*. Sie müssen in optimaler Weise aufeinander abgestimmt sein, damit Entwicklungsaufgaben von Teams produktiv umgesetzt werden können.

Rollen

In Scrum gibt es sechs verschiedene *Rollen* für Projektbeteiligte, davon drei interne, also dem Scrum-Team zugehörige (Product Owner, Entwicklungsteam, Scrum Master) und drei externe (Kunde, Anwender. Management).

- *Product Owner*

  Der Product Owner ist der Visionär und Stratege. Er entwickelt die Produktidee und Produktspezifikation und entscheidet darüber, ob ein Release freigegeben wird. Für das Projekt trägt er die kaufmännische Verantwortung gegenüber dem eigenen Unternehmen und führt die Kommunikation zum Kunden.

- *Entwicklungsteam*

  Das Entwicklungsteam (*Development Team*) ist interdisziplinär besetzt und besteht aus allen Personen, die für die Entwicklung des Produkts benötigt werden. Seine Größe ist auf 3 bis 9 Personen begrenzt, um einerseits alle notwendigen Qualifikationen zu integrieren, andererseits aber den internen Organisationsaufwand zu minimieren. Entscheidend für Scrum ist, dass das Entwicklungsteam sich selbst managt und alle internen Prozesse eigenverantwortlich steuert. Beispielsweise hat auch der Product Owner keinen Einfluss darauf, wer welche Aufgabe im Team übernimmt. Das Team ist „als Team“ verantwortlich für das Erreichen der Entwicklungsziele und für die Qualität der Leistung, entsprechend den vorher vereinbarten Standards.

- *Scrum Master*

  Der Scrum Master ist der Supervisor, Berater und Moderator in allen Scrum-Prozessen. Ohne weisungsbefugt zu sein, sorgt er doch dafür, dass die Regeln von Scrum im Team eingehalten werden. Er arbeitet an der Beseitigung von Hindernissen und Kooperationsproblemen innerhalb des Entwicklungsteams bzw. zwischen Entwicklungsteam und Product Owner. Zudem schult er die an Scrum beteiligten Personen.

Diese drei Rollen des Scrum-Teams werden ergänzt durch drei externe Rollen: den *Kunden* (*customer*) als dem externen oder internen Auftraggeber des Projekts, den *Anwender* (*user*) der Produktentwicklung, der mit seinem Feedback frühzeitig in die Entwicklung einbezogen wird, und das *Management*, das innerhalb des Unternehmens die notwendigen Ressourcen für das Projekt bereitstellt und ggf. den Scrum Master bei der Beseitigung von Hindernissen unterstützt.

Events

Die Produktentwicklung erfolgt in sogenannten *Sprints*. Dabei handelt es sich um zeitlich vorab befristete Entwicklungsschritte, an deren Ende das fertige, qualitätsgesicherte und dokumentierte Release bestimmter Produktfunktionalitäten ausgeliefert wird. Die Dauer eines Sprints beträgt zwischen einer und maximal vier Wochen, in der Regel sind es zwei Wochen. Eine Verlängerung ist nicht vorgesehen. Für die Projektplanung bedeutet dies, das Entwicklungsvorhaben in relativ kleine, kurzfristig fertigzustellende Arbeitspakete bzw. *Tasks* herunterzubrechen. Basis hierfür sind keine umfassenden Anforderungsspezifikationen, sondern

Funktionsbeschreibungen in Form von *User Stories* aus Anwendersicht, die die Entwicklung jeweils eines Sprints instruieren. Durch die kurze Taktung und den Auftrag an das Entwicklerteam, innerhalb eines Sprints tatsächlich ein fertiges Release auszuliefern, gibt es regelmäßig und sehr schnell die Möglichkeit, die Zielerreichung zu überprüfen und Feedback vom Kunden bzw. von den Anwendern auf Basis ausgearbeiteter Produktfunktionalität einzuholen.

Ein Sprint wird initialisiert und begleitet von *Meetings*, die entsprechend ihrer Position im Scrum-Prozess, ihrer Aufgaben und ihrer Organisationsform in verschiedene Kategorien unterteilt werden:

- Im *Sprint Planning Meeting 1* zu Beginn eines Sprints verständigen sich Product Owner, Entwicklungsteam, Scrum Master, Management und Anwender auf die Ziele und Anforderungen des anstehenden Sprints. Am Ende wählt das Entwicklungsteam die sogenannten *Product Backlog Items* (siehe unten) aus, die in diesem Sprint entwickelt werden sollen.

- Das anschließende *Sprint Planning Meeting 2* dient dem Entwicklungsteam dazu, die Umsetzung der Anforderungen detailliert zu planen. Dazu gehören etwa der Entwurf des Softwaredesigns sowie die Festlegung aller notwendigen Aufgaben des Sprints.

- Im *Daily Scrum* treffen sich sämtliche Mitglieder des Entwicklungsteams täglich zur selben Uhrzeit für eine Viertelstunde, um die Tagesarbeit zu koordinieren. Unter Moderation des Scrum Masters legen die Teammitglieder eigenverantwortlich fest, welche Aufgabe sie übernehmen. Außerdem kann über eventuelle Probleme und Hindernisse an den Scrum Master berichtet werden.

- Mindestens einmal während eines Sprints findet ein *Estimation Meeting* statt. Der Product Owner und das Entwicklungsteam beraten hier gemeinsam auf der Basis neuer Schätzungen, ob die frühere Planung des *Product Backlog* (siehe unten) aktualisiert werden sollte. Dies kann zur Veränderung des Sprint-Fahrplans, aber auch zur Aufnahme neuer *Backlog Items* führen.

- Am Ende eines Sprints stellt das Entwicklungsteam im *Sprint Review* den Kunden und Anwendern die Resultate ihrer Arbeit vor. Dabei werden nur die fertigen und getesteten Funktionalitäten präsentiert. Alle nicht abgeschlossenen oder nicht hinreichend getesteten Items gelten als nicht geliefert.

- In der *Sprint Retrospective* analysiert das Entwicklungsteam schließlich rückblickend den Verlauf seiner Arbeit im Sprint und diskutiert Verbesserungsmöglichkeiten. Diese „Lessons learned“ werden im *Impediment Backlog* (siehe unten) festgehalten und gehen unmittelbar in die Planungs-

phase des folgenden Sprints ein. In dieser Weise ist der Scrum-Prozess auf den kontinuierlichen Lernprozess des Teams und die inkrementelle Verbesserung der Planungs-, Organisations- und Arbeitsverfahren ausgelegt.

Artefakte

Im Scrum-Prozess entsteht eine Reihe unterschiedlicher *Artefakte*, die entweder als Planungs- und Ergebnisdokumente den Entwicklungsprozess unterstützen oder das Produkt-Inkrement als Ergebnis des Entwicklungssprints beinhalten. Hier seien nur einige dieser Artefakte aufgeführt:

- *Product Backlog*: In einer Liste werden alle zu entwickelnden Funktionalitäten festgehalten. Die Darstellungsweise sollte anwenderorientiert sein und sich möglichst auf *User Stories* stützen, in denen kurz notiert ist, welcher Anwender welche Funktionalität mit welchem erwarteten Nutzen wünscht. Verantwortlich für den Product Backlog ist der Product Owner, der die *Items* der Liste auch priorisiert.

- *Sprint Backlog*: Hier wird die Funktionalitätenliste des Product Backlog auf die zu erledigenden Entwicklungsaufgaben eines Sprints heruntergebrochen. Festgehalten werden nicht nur die Tasks in der vorgegebenen Reihenfolge, entsprechend den Diskussionen und Entscheidungen in den beiden Sprint Planning Meetings, sondern auch kontinuierlich der jeweilige Bearbeitungsstand der Tasks, wie er im Daily Scrum besprochen wird.

- *Impediment Backlog*: Im Rahmen seiner Aufgabe, Hindernisse im Scrum-Prozess abzubauen, führt der Scrum Master eine Liste mit der Beschreibung aller Hindernisse (*impediments*), die während der Produktentwicklung auftreten, und dokumentiert ihre Beseitigung.

Einordnung von Scrum

Scrum ist als Vorgehensmodell auf die inkrementelle Produktentwicklung in kurzen, schnellen Zyklen ausgelegt. Die dafür bereitgestellten Instrumente erlauben tatsächlich neben der zügigen und regelmäßigen „Sichtbarkeit“ von fertigen Produktinkrementen eine sehr flexible Prozessgestaltung auf Grundlage eng getakteter Performance-Analysen der Entwickler und regelmäßiger Feedbacks der Anwender. Der Fokus liegt auf der Organisation des Entwicklerteams, die für die laufende Projektarbeit weitgehend eine Selbst-Organisation des Teams ist. Inwieweit dies in produktiver Weise gelingt, hängt von den beteiligten Personen ab sowie ihrer Fähigkeit und Bereitschaft, sich den Anforderungen eines Sprints flexibel anzupassen und beispielsweise auch Aufgaben zu übernehmen, die dem eigenen Qualifikationsprofil nicht oder nur teilweise entsprechen. Zugleich müssen die geforderte Abkehr von hierarchischen Strukturen innerhalb des Entwicklerteams und die gemeinschaftliche Regelung der Aufgabenbearbeitung erst einmal mit Leben gefüllt werden. Bemerkenswert in Scrum ist die systematische Einbettung von „Lessons learned“ in die Taktung der Sprints. Im Idealfall vollzieht das Entwicklerteam auf diese Weise einen kontinuierlichen Lernprozess noch während des Entwicklungsprojekts und wird von Sprint zu Sprint „besser“.

Scrum zeigt in dem Konzept der kurzen Entwicklungssprints, der schnellen Produktreleases und der relativ unscharfen Anforderungsdefinition der User Stories viele Analogien zum Extreme Programming. Kritische Fragen stellen sich daher analog zum XP, etwa zu der unzureichend behandelten Sicht auf das Gesamtprodukt bzw. den gesamten Lebenszyklus des Entwicklungsprojekts.

Anders als XP ist Scrum schon seit Langem dem engeren Feld des Software Engineering entwachsen und hat sich zu einem allgemeinen Managementframework entwickelt, für das Projektmanagement ebenso wie für die Produktentwicklung oder die Organisationsentwicklung. Die agilen Prinzipien von Scrum können erfolgreich auch außerhalb der Softwareentwicklung implementiert werden, und das nicht nur in kleinen, sondern auch in hochskalierten Projekten (siehe hierzu etwa /Gloger 2013/).

1.7 Zusammenfassung

Aufgaben des Projektmanagements

In der vorliegenden Kurseinheit wurden zunächst als Einführung in den Kurs die Aufgaben des Projektmanagements umrissen. Dabei wurde auch ein Überblick über den Kurs und seinen Aufbau gegeben.

Als Aufgaben des Projektmanagements wurden das Setzen von Zielen, sicheres Entscheiden, das Delegieren von Aufgaben, die Koordination und Organisation sowie eine Kontrolle im Sinne einer Qualitätssicherung identifiziert. Ergänzend kommen Aspekte der Motivation, der Information, der Kommunikation sowie der Steuerung hinzu. All diese Bereiche werden in den weiteren Kurseinheiten ausführlich betrachtet.

Vorgehensmodelle

In einem zweiten Teil haben wir dann verschiedene Vorgehensmodelle für Softwareentwicklungsprojekte vorgestellt. Diese Vorgehensmodelle regeln im Wesentlichen den Ablauf der Aktivitäten in einem Softwareprojekt. Wir haben zunächst die wichtigsten Grundformen (Wasserfallmodell, Prototyping, Spiralmodell) vorgestellt. Anschließend haben wir mit dem V-Modell, dem Rational Unified Process, dem Extreme Programming sowie Scrum vier bedeutsame Vorgehensmodelle behandelt. Die Betrachtung dieser Modelle verdeutlicht, dass in der Praxis eine Mixtur aus den Grundformen nicht nur möglich, sondern sogar notwendig ist.

Wichtig ist dabei, dass Softwareentwicklung immer auf einem Vorgehensmodell aufbauen muss, denn wenn keine Klarheit über das Vorgehen besteht, kann der Entwicklungsprozess weder geplant noch kontrolliert werden (im Sinne einer Feststellung von Planabweichungen).

Es erscheint deshalb sinnvoll, in einem Unternehmen für alle durchzuführenden Softwareprojekte das Vorgehensmodell zu standardisieren. Um dabei den verschiedenen Projektarten (Kleinprojekte ↔ Großprojekte, Neuentwicklung ↔ Weiterentwicklung usw.) Rechnung zu tragen, sollte man verschiedene Varianten

vorsehen. Das im V-Modell vorgesehene *Tailoring* kann dabei einen möglichen Ansatz zur flexiblen Gestaltung bilden.

Das Vorgehensmodell sollte Möglichkeiten des Prototyping integrieren und Risiko- und Qualitätssicherungs-Aspekte analog zum Spiralmodell berücksichtigen.

Bei der Definition eines solchen Vorgehensmodells für ein Unternehmen sollte man sich von den bekannten Modellen aus der Literatur (wie z.B. dem V-Modell oder Scrum) leiten lassen. Anpassungen sind dabei bezüglich der typischen Projektgröße, der eingesetzten Methoden, der Wünsche der Auftraggeber, der Fähigkeiten der Mitarbeiter, der Unternehmenskultur usw. möglich und notwendig.

Lehrziele

Nach dem Durcharbeiten dieser Kurseinheit sollten Sie

- die wesentlichen Aufgabenbereiche des Projektmanagements beschreiben können,
- die Funktion und den Aufbau von Vorgehensmodellen für Softwareprojekte kennen,
- die in dieser Kurseinheit vorgestellten Vorgehensmodelle beschreiben und miteinander vergleichen können,
- in der Lage sein, im Kurs nicht behandelte Ansätze zu Vorgehensmodellen einzuschätzen und zu den behandelten Ansätzen in Relation zu setzen.

Literaturverzeichnis

/Balzert 1996/ Balzert, Helmut: *Lehrbuch der Software-Technik: Software-Entwicklung*, Spektrum Akademischer Verlag, Heidelberg, Berlin, 1996

/Balzert 1998/ Balzert, Helmut: *Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*, Spektrum Akademischer Verlag, Heidelberg, Berlin, 1998

/Balzert 2008/ Balzert, Helmut: *Lehrbuch der Software-Technik: Software-Management,* 2. Auflage, Spektrum Akademischer Verlag, Heidelberg, Berlin, 2008

/Beck 1999/ Beck, Kent: *Extreme Programming Explained – Embrace Change*, Addison-Wesley, Reading, Mass., 1999

/Beck, Andres 2004/ Beck, Kent und Andres, Cynthia: *Extreme Programming Explained – Embrace Change*, 2. Auflage, Addison-Wesley Longman, Amsterdam, 2004

/Boehm 1981/ Boehm, Barry W.: *Software Engineering Economics*, Prentice Hall, Upper Saddle River, New Jersey, USA, 1981 (auch in Deutsch: *Wirtschaftliche Software-Entwicklung*, Prentice Hall, 1981)

/Boehm 1986/ Boehm, Barry W.: *A Spiral Model of Software Development and Enhancement*, in: ACM SIGSOFT, August 1986, S. 14-24

/Boehm 1988/ Boehm, Barry W.: *A Spiral Model of Software Development and Enhancement*, in: IEEE Computer, May 1988, S. 61-72

/Charette 2005/ Charette, Robert N.: *Why Software Falls*, in IEEE Spektrum, Sept. 2005, S. 36-43 (http://spectrum.ieee.org/computing/software/why-software-fails)

/Corsten et al. 2008/ Corsten, Hans; Corsten, Hilde; Gössinger, Ralf: Projektmanagement. Einführung, 2. Auflage, Oldenbourg Wissenschaftsverlag, München, 2008

/Corsten, Gössinger 2008/ Corsten, Hans und Gössinger, Ralf (Herausgeber): *Lexikon der Betriebswirtschaftslehre*, 5. Auflage, Oldenbourg Wissenschaftsverlag, München, 2008

/Derniame et al. 1999/ Derniame, J.-C. ; Kaba, B.A. und Wastell, D. (Herausgeber): *Software Process: Principles, Methodology, Technology*, Springer, Lecture Notes in Computer Science, Vol. 1500, Berlin/Heidelberg 1999

/Elting, Huber 2001/ Elting, Andreas und Huber, Walter: *Schnellverfahren – Mit Extreme Programming immer im Plan?*, in c't magazin für computer technik, Heft 3, 2001, S. 186-191

/Elzer 1994/ Elzer, Peter F.: *Management von Softwareprojekten – Eine Einführung für Studenten und Praktiker*, Braunschweig; Wiesbaden: Vieweg, 1994

/Feyhl, Feyhl 1996/ Feyhl, Achim W. und Feyhl, Eckhard: *Management und Controlling von Softwareprojekten*, Wiesbaden: Gabler, 1996

/Gloger 2013/ Gloger, Boris: *Scrum. Produkte zuverlässig und schnell entwickeln*, 4., überarbeitete Auflage, Carl Hanser Verlag, München, 2013

/Grupp 1998/ Grupp, Bruno: *Qualifizierung zum Projektleiter: DV-Projektmanagement im Wandel*, 4., überarbeitete Auflage, Computerwoche-Verlag, München, 1998

/Humphrey 1990/ Humphrey, W.S.: *Managing the Software Process*, Addison-Wesley, Reading, Massachusetts, USA, 1990

/Jeffries 2001/ Jeffries, Ron (Editor): *XProgramming.com*, 2001 (http://www.xprogramming.com/index.htm)

/Kecher 2011/ Kecher, Christoph: *UML 2: Das umfassende Handbuch*, 4. Auflage, Galileo Computing, Bonn, 2011

/Kellner 2001/ Kellner, Hedwig: *Die Kunst, IT-Projekte zum Erfolg zu führen: Ziele – Strategien – Teamleistungen*, Hanser Verlag, München, 2001

/Kraus, Westermann 2010/ Kraus, Georg und Westermann, Reinhold: *Projektmanagement mit System: Organisation, Methoden, Steuerung*, 4. Auflage, Gabler, Wiesbaden, 2010

/Kruchten 1999/ Kruchten, Philippe*: Der Rational Unified Process: Eine Einführung*, Addison-Wesley, München, 1999

/Lehner et al. 1991/ Lehner, Franz; Auer-Rizzi, Werner; Bauer, Robert; Breit, Konrad; Lehner, Johannes; Reber, Gerhard: *Organisationslehre für Wirtschaftsinformatiker*, Hanser Studienbücher, München, 1991

/Müller-Merbach 1998/ Müller-Merbach, Heiner: *Einführung in die Betriebswirtschaftslehre für Erstsemester*, 2., überarbeitete Auflage, WiSo-Kurzlehrbücher, Verlag Franz Vahlen, München, 1998

/Pagel, Six 1994/ Pagel, Bernd-Uwe; Six, Hans-Werner: *Software Engineering: Die Phasen der Softwareentwicklung*, Oldenbourg-Verlag, München, Wien, 1994

/Rational 2001/ *Rational Unified Process Whitepaper: Best Practices for Software Development Teams*, TP-026B Rev. 11/01, 1998 (ftp://public.dhe.ibm.com/software/rational/web/whitepapers/2003/rup_bestpractices.pdf)

/Reißing 2000/ Reißing, Ralf: *Extremes Programmieren*, Aktuelles Schlagwort in Informatik Spektrum, Band 23, Heft 2, April 2000, S. 118-121

/Schwaber, Beedle 2001/ Schwaber, Ken und Beedle, Mike: *Agile Software Development with Scrum*, Prentice Hall, Upper Saddle River, NJ, 2001

/Sommerville 2012/ Sommerville, Ian: *Software Engineering*, 9., aktualisierte Auflage, Pearson Studium, München, 2012

/Sutherland 2010/ Sutherland, Jeff: *Scrum Handbook*. Scrum Training Institute Press, Somerville, MA, 2010 (http://www.misuratau.edu.ly/uploads/3K75P9W79B5HC4Q.pdf)

/V-Modell XT 1.4 2012/ Hrsg.: Bundesrepublik Deutschland. Quelle: www.v-modell-xt.de

/Wintersteiger 2013/ Wintersteiger, Andreas: *Scrum – Schnelleinstieg*. 2. Auflage, Entwickler.Press, Frankfurt, 2013

/Wöhe, Döring 2010/ Wöhe, Günter und Döring, Ulrich: *Einführung in die Allgemeine Betriebswirtschaftslehre*, 24., überarbeitete und aktualisierte Auflage, Verlag Franz Vahlen, München, 2010

Index zu Kurseinheit 1

Ablauforganisation 6, 23, 27, 31

Aktivität 28, 42

Analyse und Definition 27, 33, 35, 36, 44, 64

Arbeitsabläufe (Workflows) 54, 57, 58, 59

Aufgabe 8, 9, 19, 22, 28, 29, 30, 31, 40, 41, 61

Ausarbeitung 54, 56

Charakteristika eines Projektes 8

COTS-Software 12, 13

Delegieren 22, 70

Einsatz und Wartung 36, 41, 44

Einzelentwicklung 10

Entscheiden 5, 21, 70

Entwurf 27, 35, 38, 44, 54, 62

Entwurfsspezifikation 36

ERP-Systeme 12

evolutionäre Softwareentwicklung 43, 45, 53

Extreme Programming (XP) 4, 48, 60, 65, 70, 75

horizontaler Prototyp 44

Implementierung 27, 36, 40, 44, 57, 58, 62

Information 20, 21, 25, 70

Integration 6, 12, 36, 39, 41, 43, 54, 64

Kleine Releases 61

Kommunikation 17, 20, 21, 25, 65, 70

Konstruktion 54, 57

Kontrollieren 5, 23, 24

Konzeptphase 54, 56

Motivation 7, 21, 24, 39, 43, 45, 63, 70

Netzplantechnik 7, 21, 32, 33, 34

Neuentwicklung 10, 12, 13, 70

Organisation 6, 22, 23, 25, 53, 70, 76

Phase 27, 28, 33, 34, 35, 36, 37, 38, 39, 40, 42, 44, 46, 47, 56, 57

Planen 5, 21

Planspiel 61, 64

Praktiken 61, 62, 64, 65

Produktdefinition 27, 35, 36, 39, 43

Produktivität 17, 18, 19, 21

Projekt 6, 7, 8, 9, 11, 12, 13, 15, 19, 21, 22, 25, 29, 30, 32, 33, 34, 35, 38, 49, 59, 60, 61, 65

Projektfortschritt 14, 17, 26, 27, 29

Prototyping 39, 43, 44, 45, 46, 47, 56, 59, 60, 61, 66, 70, 71

Prozessanalyse 12

Rapid Prototyping 43

Rational Unified Process (RUP) 4, 48, 53, 54, 55, 58, 59, 70, 71, 76

Realisieren 5

Refaktorisierung 62

Scrum 4, 48, 66, *70*

Software-Architektur 35, 36

*Software*projekt 8, 9, 11, 12, 13, 23, 28, 70

Software-Projektmanagement 5, 60

Software-Qualitätssicherung 5, 6, 75

Softwaretechnik 4, 5, 6, 27

Spiralmodell 45, 47, 48, 56, 70, 71

Standardisierung 29, 53

Steuern 21, 23

Steuerung 7, 21, 23, 34, 70, 76

Systementwicklung 51

Systemmetapher 62

Tailoring 52, 53, 71

Test 36, 38, 40, 57, 59, 62

Transparenz 17, 25

Übergang in den Betrieb 54, 57

Unified Modelling Language (UML) 53, 59

Use Cases 53, 58

Validierung 42, 43

Verifizierung 42

vertikaler Prototyp 44

V-Modell 4, 7, 48, 50, 52, 53, 59, 70, 71, 77

Vorgehensbausteine 50

Vorgehensmodell 4, 6, 17, 27, 28, 29, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 41, 42, 43, 53, 54, 65, 70, 71

Wartung 12, 36, 38

Wasserfallmodell 35, 39, 40, 41, 42, 70

Wegwerfprototyp 43

Weiterentwicklung 10, 12, 24, 44, 57, 70

Workflow-Managementsystem 31

Ziele setzen 5, 21

Zyklus 47, 48, 54, 57, 61