DISSERTATION

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

unter der Leitung von:
o. Univ. Prof. Dipl.-Ing. Dr. Franz Wojda
Institut für Betriebswissenschaften, Arbeitswissenschaft und Betriebswirtschaftslehre (E330)

2. Begutachter:
o. Univ. Prof. Dr. A Min Tjoa
Institut für Softwaretechnik und Interaktive Systeme (E188)

eingereicht an der Technischen Universität Wien
Fakultät für Maschinenbau

SP^{4}M Software Projekt Produkt Programm Management & Patterns

von:
Dipl.-Ing. Christian Exl
Maxingstraße 60
A – 1130 Wien

Matr.Nr. 8825946

Wien, im April 2004

Christian Exl

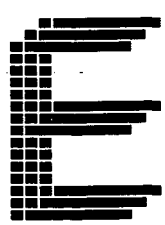
Diese Dissertation haben begutachtet:

....

SP^{4}M Software Projekt Produkt Programm Management

& Patterns

Christian Exl



Es sind Persönlichkeiten, nicht Prinzipien, die etwas bewegen.

Oscar Wilde

Für Tamara und Philip

Kurzfassung

Die Anwendung der Disziplin Projektmanagement auf unterschiedliche Sachgebiete erfordert jeweils entsprechende Anpassungen, welche auf die speziellen Rahmenbedingungen Rücksicht nehmen – in dieser Arbeit sind sie durch die Charakteristika der Software-Entwicklung bzw. –Produktion gegeben. Darüber hinaus wird auf die Einbindung des Projektmanagements in das Produkt- und das Programm-Management Bezug genommen. Die inhaltliche Gliederung (Projekt-Produkt-Programm-Management, Betriebswirtschaftliche Theorien, Projekt-Management-Leitfaden, Kernprozesse eines Softwarehauses, Software-System-Modellierung im Entwicklungs-Prozess, Aufwandsschätzung und Produktivität, Metrik-Programme, Organisationsgestaltung, Risiko-Management, und Patterns) resultiert aus der Identifikation erfolgskritischer Thematiken. Als Ausgangsbasis dienen die Erfahrungen und Prozesse eines der größten Softwarehäuser Europas. Darauf aufbauend erfolgt eine Diskussion der kritischen Themen des Software-Projektmanagements. Gegenüber der herkömmlichen Strukturierung wird in dieser Arbeit insbesondere Wert auf die Gestaltung der Strategien und der Kooperationen zwischen den Stakeholdern gelegt. Dies wird umso wichtiger als Software in zunehmendem Maße dezentral – über Kontinente hinweg – entwickelt wird, wobei das Projektergebnis (= Software) den widersprüchlichen Requirements der Anpassung an Kundenprozesse einerseits und dem Wunsch nach standardisierten Lösungen (hohe Absatzmengen des Software-Produkts) andererseits genügen sollte. Um das aufgrund dieser Rahmenbedingungen in den Organisationen (des Softwarehauses) aggregierte Wissen für möglichst viele Mitarbeiter zur Verfügung zu stellen wird ein für das Projektmanagement neues Konzept – die Pattern-Language – dargestellt. Damit lässt sich Wissen unterschiedlichster Struktur fragmental abbilden (d.h. in Pattern-Notation) und mit anderem Erfahrungswissen in Beziehung setzen.

Abstract

The application of the discipline project management to various subject areas requires specific adaptations which consider specific business environments. In this thesis, project management know-how is applied to the field of software development. Beyond it, product management and program management are complementary topics with tight links to project management in the software industry. The content of the thesis is based on critical success factors for software development. These are: Project-Product-Program-Management, Management Theories, Project Management Guidelines, Core Processes of a Software House, Software System Modeling within the Development Process, Effort Estimation and Productivity, Metrics Programs, Organization Design, Risk Management, and Patterns. The selection of those topics is based on the experience of one of the largest software development companies in Europe. This experience led to the discussions about software project management as presented here. In contrast to the emphasis of other scientific papers written on this subject, the main focus lies in the design of the business strategy and the communication between the stakeholders. These topics are also of growing importance due to decentralization of software development activities. There are two boundary conditions which are essential for successful software products: on the one hand the software has to be adapted to the individual business processes, on the other hand software products have to be as much standardized as possible to answer multiple requests and lower development costs. To distribute the collected know-how within a company a new concept within the discipline of project management is recommended: The creation and application of a pattern language. By applying it, knowledge will be stored in a fragmented manner (i.e. in pattern notation) and in relationships between individual patterns.

Inhaltsverzeichnis

1. Einleitung

1.1 Präambel

Obwohl ein großes Softwarehaus - gemessen an der Personalkapazität - nach außen hin einen teils statischen Eindruck erweckt, kommt es innerhalb der Organisation regelmäßig zu starken Erschütterungen. Dass ein Projekt, an dem seit 3 Jahren 150 Personen arbeiten, innerhalb nur einer Stunde ohne Vorwarnung eingestellt wird, oder ein Software-Entwickler nach 20 Berufsjahren noch an keinem einzigen zu Ende ausgeführten Projekt mitgearbeitet hat, ist Realität. Auf der anderen Seite ist es ebenfalls Realität, dass die Personal-Verantwortung eines Mitarbeiters innerhalb nur eines Jahres von 0 auf 100 Mann wachsen kann; oder: Hat man sich in einem Jahr darauf eingestellt, mit einem Visualisierungswerkzeug Klassen in C++ zu erstellen, könnte sich die Anforderung im nächsten Jahr auf das Verhandeln von Preisen, oder das Verkaufen auf Messen verlagern.

Dies sind nur einige wenige Beispiele, die die gewaltige Dynamik widerspiegeln mit denen die Mitarbeiter in einem Softwarehaus konfrontiert sind. Parallel dazu wird aber auch das Wissen über Software- und Prozess-Technologien weiterentwickelt und detaillierter in ihren Zusammenhängen erarbeitet. Neben wissenschaftlichen Publikationen sammeln sich in großen Firmen umfangreiche Regelwerke an, die aber schlussendlich außer von den Erstellern kaum von jemand anderem gelesen, oder gar eingehalten werden. Die Schere zwischen verfügbarem Wissen und tatsächlich genutztem öffnet sich hierbei kontinuierlich weiter, wobei durch das an vielen Stellen verteilte Detailwissen der Überblick auf das Wesentliche versperrt wird.

Darüber hinaus unterstützen die modernen Kommunikationsmedien die weltweit verteilte Software-Entwicklung, wodurch die Produktivität über niedrige Personalkosten und / oder entsprechend spezialisierte Expertenteams in einfacher Weise optimierbar ist.

1.2 Ausgangssituation

Die Ausgangssituation muss aufgrund der unterschiedlichen Einflüsse, die zu einer Variation der Betrachtungsweise führen, mehrdimensional gesehen werden. Einerseits gelten die Grundlagen der Standard-Literatur zum Projekt-Management bzw. Projekt-Programm-Management und der Organisationslehre auch für Software schaffende Organisationen. Andererseits entstehen laufend neue Methodiken (z.b. Capability Maturity Model) und Tools (z.b. UML-unterstützende Software-Werkzeuge), die speziell auf eine Effizienz- und Qualitätsverbesserung in der Software-Entwicklung abzielen. Hinzu kommen die sich rasch ändernden Kooperationen mit anderen Software-Firmen, die in verstärktem Ausmaß über die Software-Entwicklung hinausgehende Kenntnisse notwendig machen.

Diese Situation verdeutlicht, dass ein weiterer Handlungsbedarf darin besteht, die komplexen Zusammenhänge im Software produzierenden Sektor zu untersuchen und in ihrer wechselseitigen Beeinflussung zu diskutieren.

1.3 Zielsetzung

Erarbeitung der wesentlichen Konzepte um ein Projekt-Produkt-Programm-Management innerhalb eines Softwarehauses erfolgreich zu planen und zu implementieren. Die Konzepte bauen auf den hierfür notwendigen Komponenten des Projekt-Managements auf und werden anschließend in ihrem Zusammenwirken und unter Ausrichtung auf die Erfordernisse der Software-Entwicklung im Detail dargestellt. Des weiteren soll ein Konzept gefunden werden, das die Defizite der herkömmlichen Wissens-Strukturierung über das Software-Projekt-Management behebt. Durch das Hinzufügen der Begriffe: „Produkt“ und „Programm“ im Titel wird die enge Verknüpfung des Software-Projekt-Managements, der Produkt-Charakteristika, und eines Programms auf strategischer Ebene für die weiteren Ausführungen unterstrichen.

1.4 Methodik

Beginnend mit der Präambel, der Ausgangssituation und der Zielsetzung wurde mehrmals ein iterativer Prozess durchlaufen, wie er in der Abb. 1.1 gezeigt ist. Im Zentrum des Interesses stehen dabei Modelle des Managements von Software-Projekten, die durch Ergänzungen aus dem Programm- und Produkt-Management abgerundet werden.

Wesentlich für die Erarbeitung der Thematik war neben dem Einbringen der eigenen Erfahrung das Studium der Literatur, welche auch das Medium Internet bzw. World Wide Web

miteinschloss. Die Literatur in Form von Büchern und Journalen, seien sie nun auf elektronischer Basis oder in klassischer Papierform, erwies sich als besonders wertvoll wenn es um das Studium einzelner Aspekte ging. Demgegenüber erwiesen sich die Suchmaschinen, allen voran www.google.com oder www.google.de als besonders hilfreich im Finden von Querbezügen. Ein neben dem Projekt-Management wichtiges Wissensgebiet auf dem diese Arbeit basiert, ist jenes des Software-Engineering.

Durch das Einbringen eigener einschlägiger Erfahrungen eignet sich aus der Menge der erkenntnistheoretischen Ansätze vor allem der Positivismus. Auch im Falle der Einbindung und Referenzierung auf Literatur (Internet) diente die eigene Praxis der Hervorhebung diverser Aussagen. Nur in seltenen Fällen wird auf den kritischen Rationalismus – das Prinzip der Falsifizierung – zurückgegriffen. Wesentlicher Grund hierfür ist die noch unbefriedigende Existenz von auf das Software-Projekt-Management anwendbarer Theorien.

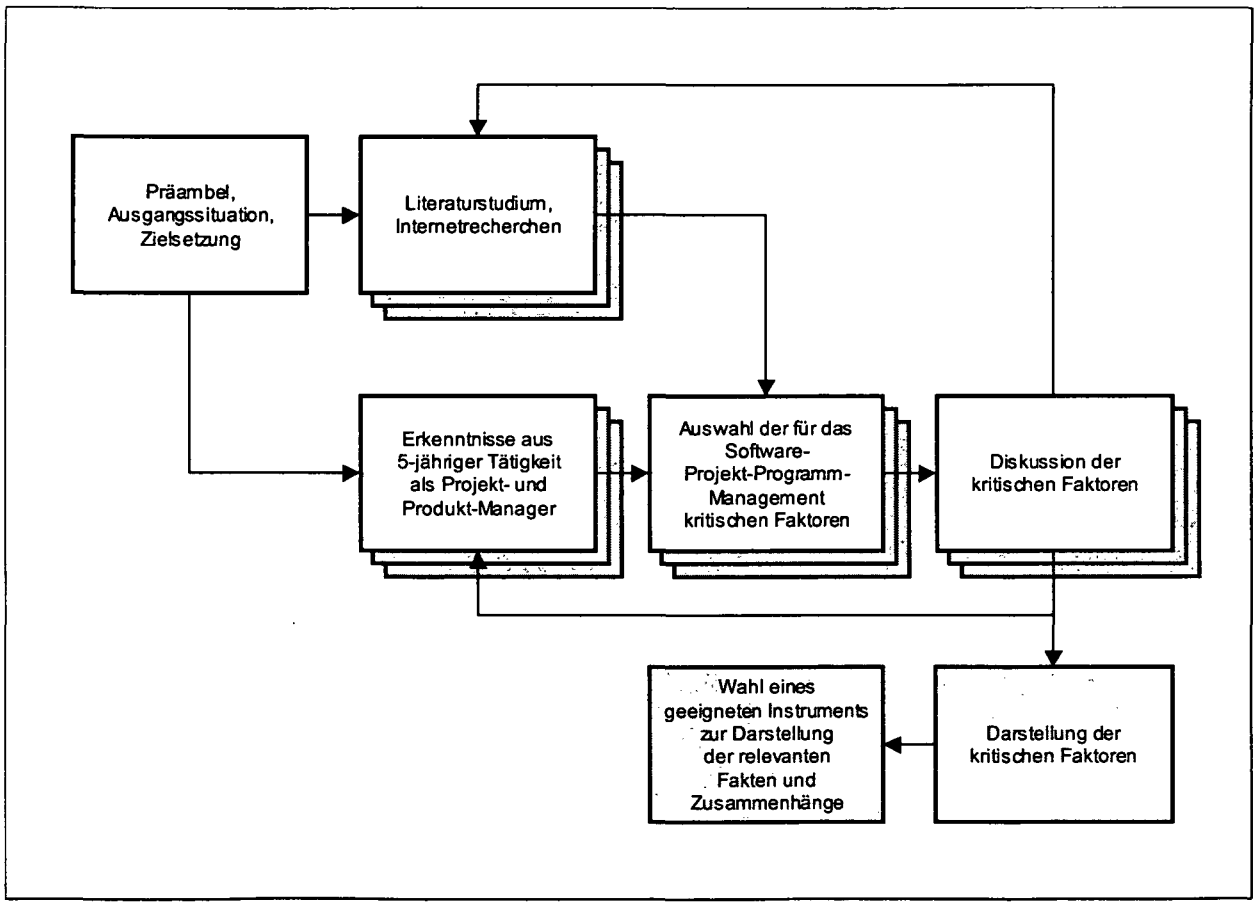


Abb. 1.1: Methodik hinsichtlich Durchführung der Dissertation

Für die Auswahl (vgl. Abb. 1.1) der in diesem Werk enthaltenen Schwerpunkte waren mehrere Kriterien ausschlaggebend:

- große bis sehr große Software-Projekte (ab etwa 10 Mannjahren Aufwand)
- Konzentration auf die Software-Entwicklungs-Phasen hinsichtlich Software-Lebenszyklus
- Themen, die für die Anbieterseite von Software relevant sind
- die organisatorischen Rahmenbedingungen entsprechen Softwarehäusern, die in Form eines Software-Programms parallel an mehreren Software-Projekten arbeiten
- hinsichtlich der Software-Organisation wird ein international verteiltes Team nicht ausgeschlossen
- qualitativ sollen hoch- bis höchstqualitative Software-Ergebnisse erzielbar sein
- die Einstellung zum Projekt-Risiko soll im Bereich: avers bis neutral liegen
- die prozessorientierte Sichtweise dient der primären Strukturierung
- wirtschaftlich-organisatorische Aspekte stehen gegenüber technischen Aspekten im Vordergrund
- durch eine ausgewogene strategische Ausrichtung sollen die Interessen aller Stakeholder berücksichtigt werden

Zusammengefasst orientieren sich diese Kriterien an mittelgroßen und großen Softwarehäusern, die zumeist in vielen Anwendungsbereichen tätig sind. Sie treffen dagegen nicht auf Unternehmen zu, in welchen die Software-Produktion nicht zumindest einer der Schwerpunkte im Unternehmens-Portfolio ist.

Ein weiteres Kriterium für die Themenselektion sind die aus der Literatur, im besonderen aber aus den eigenen praktischen Erfahrungen, identifizierten Risken und deren zur Steuerung notwendigen Methoden. Hierzu ein Ausschnitt aus dem Kapitel zum Risiko-Management:

Im folgenden wird anhand der zuletzt genannten Gliederung eine Auflistung der in der US-Software-Community genannten Top-Risken, jeweils aus Software-Konsumenten- und Software-Produzenten-Sicht, gegeben [1]). Dies unter der Voraussetzung, dass der Kunde bzw. Konsument am Software-Projekt beteiligt ist. Hierbei kennzeichnet „K" ein Risiko für den Konsumenten mit entsprechender Wertung (1 ... höchste Priorität; 10 ... geringste Priorität). Die Top-10-Risken der Produzenten wurden mit „P" gekennzeichnet und ebenfalls entsprechend gereiht.

Als Beispiele für das Projektrisiko können genannt werden:

- *Zu eng bemessene finanzielle Ressourcen (K1, P1)*
- *Unklare Rollen- und Verantwortlichkeits-Verteilung (K2)*
- *Unzureichende fachliche Qualifikation (K3)*
- *Abstimmungsprobleme hinsichtlich Zeitpläne, konkurrierender Leistungsmerkmale und technischer Abhängigkeiten (P4, K6)*
- *Terminrisiko (K9)*

Als Beispiele für Prozess-Risken wären zu nennen („M" ... Management bedingt; „T" ... technisch bedingt):

- *Unzureichende Entwicklungsprozess-Definition (P3T, K4T)*
- *Unzureichende Projekt-Planung (K5M, P5M)*
- *Mängel hinsichtlich der Software-Entwicklungs-Umgebung (z.b. ungenügend geschultes Personal für einen effizienten Einsatz (P6T)*
- *Systems-Engineering-Mängel (z.b. Mängel hinsichtlich der Systemarchitektur, Benutzeroberfläche, Sicherheitseinrichtungen, Upgrademöglichkeit) (K7T)*
- *Management-Fehler (z.b. unzureichendes Reporting und Qualitäts-Management) (P8M)*
- *Kommunikationsprobleme zwischen Hierarchien und Teams (P9M)*
- *Unzureichende Systemtests (K10T)*

Beispiele für Produkt-Risken sind:

- *Unzureichende Leistungs-Merkmal-Dokumentation (P2)*
- *Performance-Probleme (z.b. zu lange System-Antwortzeiten) (P7)*
- *Abweichung zwischen gewünschter und realisierter Funktionalität; auch hinsichtlich der MTBF (= Mean Time Between Failures) (K8)*
- *Mängel bei der System-Integration mit Legacy-Systemen (Legacy-Systeme sind bereits beim Kunden in Betrieb befindliche Software-Systeme, die mit den neu zu liefernden Systemen kooperieren müssen) (P10)*

Basierend auf diesen Kriterien wurden die Kapitel derart gegliedert, dass allgemeine Grundlagen zu Beginn und die Schwerpunkte aus der Praxis daran anschließend dargestellt werden. Während bei manchen Themen der Informationsgehalt in der Akzentuierung möglicher Lösungsvarianten liegt, werden in anderen Fällen eigene Lösungen präsentiert (siehe hierzu z.b. das 7-D-Modell nach Exl).

Den Abschluss bildet das Kapitel über Patterns – einem Schema, mit dessen Hilfe die Zusammenhänge des Software-Projekt-Produkt-Programm-Managements in idealer Weise modelliert werden können.

1.5 Gliederung und Inhalt

Die Reihenfolge der Kapitel ist derart gewählt, dass zu Beginn die grundlegenden Begriffe definiert und der Rahmen der Dissertation abgesteckt wird. Die vorgestellten betriebswirtschaftlichen Theorien dienen als Grundlage einer optimalen Ressourcen-Verteilung. Der Projektmanagement-Leitfaden beinhaltet konkrete Handlungsanweisungen zu den wesentlichsten Software-Projekt-Management-Thematiken. Im weiteren Verlauf wird auf einzelne im Rahmen des Software-Projekt-Managements wichtige Teilgebiete näher eingegangen, die im Leitfaden unberücksichtigt blieben. Im abschließenden Kapitel über

[1] Hall, Elaine M., Managing Risk, USA, 2001

Patterns wird ein neuer Zugang zur Strukturierung wesentlicher Sachverhalte im Software-Projekt-Management vorgestellt. Die Abb. 1.2 zeigt den stufenweisen Aufbau des Inhalts der Dissertation.

Im Kapitel 1: „Einleitung" wird die Motivation und Methodik zum Aufbau der Dissertation vorgestellt. Zudem werden die Kapitel übersichtsweise dargestellt um deren Beitrag zum Gesamtbild zu verstehen.

Im Kapitel 2: „Projekt-Produkt-Programm-Management" werden die Begriffe: Projekt, Projekt-Management, Projekt-Programm-Management, Produkt-Management und Systemgeschäft im Umfeld der Software-Entwicklung erklärt.

Im Kapitel 3: „Betriebswirtschaftliche Theorien" werden theoretische Ansätze zum strategischen Management vorgestellt, die der Erarbeitung von Visionen und Zielen zugrunde liegen, aus denen schlussendlich die Zielformulierungen folgen. Anhand des Portfolio-Konzeptes, der Balanced Scorecard und unternehmenswert-orientierter Konzepte werden wichtige Tools zur Strategiebildung und –visualisierung vorgestellt. Strategische Konzepte sind insbesondere für die Gestaltung der Ressourcen-Verteilung innerhalb eines Projekt-Programms und im Produkt-Management von Bedeutung.

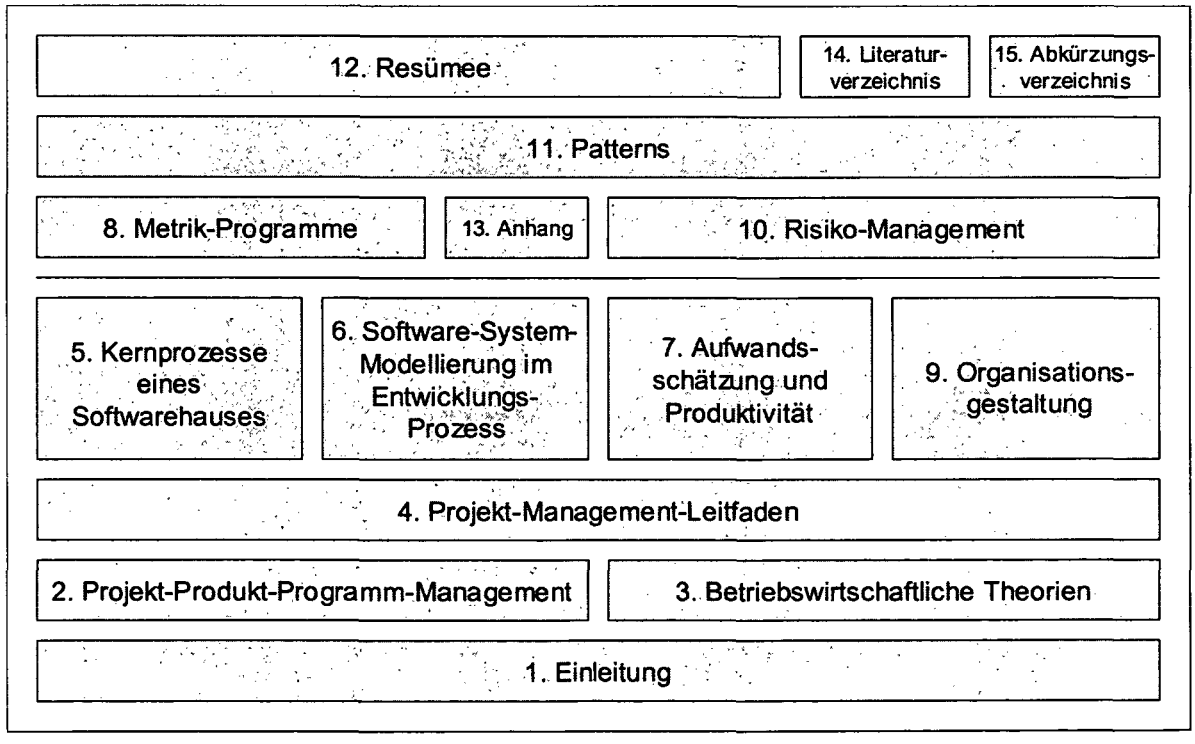


Abb. 1.2: Kapitel-Struktur

Im Kapitel 4: „Projekt-Management-Leitfaden" wird vom Dissertanten anhand konkreter Handlungsanweisungen ein in der Praxis zum Einsatz kommender Leitfaden zum Projekt-Management von Software-Projekten vorgestellt. Sein Zweck ist die schriftliche Zusammenstellung der wichtigsten Faktoren eines Software-Projekts. Dadurch erfüllt er gleichzeitig die Aufgabe einer Orientierungshilfe für die am Projekt beteiligten Mitarbeiter. Inhaltlich baut er auf den für die Software-Entwicklung vordefinierten Prozessen auf und regelt in Form eines Rahmenwerkes die wesentlichen darüber hinausgehenden Sachverhalte. Er wird in Form eines für den praktischen Einsatz gehaltenen Stils formuliert und entstammt einem zum Großteil selbst erstellten Dokument (für eine Teilorganisation der Firma Siemens).

Im Kapitel 5: „Kernprozesse eines Softwarehauses" werden die Basisprozesse einer Software erstellenden Organisation dargestellt. Sie sind bezüglich der Strukturierung an jene der Siemens-Gruppe angelehnt, jedoch um eigene Erfahrungen und die Einarbeitung strategischer Konzepte ergänzt. Die Prozesse sind die Grundlage der Arbeitsabläufe eines Software-Projekts und berücksichtigen sowohl das Systemgeschäft als auch die Produkt-Entwicklung.

Im Kapitel 6: „Software-System-Modellierung im Entwicklungs-Prozess" wird jener in Kapitel 5 vorgestellte Entwicklungs-Prozess in verschiedenen Realisierungsmodellen dargestellt. Beginnend vom einfachsten und ältesten Modell – dem Wasserfall-Modell – bis zum derzeit noch selten in die Praxis umgesetzten Spiral-Modell werden die Vor- und Nachteile der jeweiligen Vorgangsweise diskutiert. Die zweite Hälfte des Kapitels ist den frühen Phasen des Entwicklungs-Prozesses gewidmet (Analyse-, Definitions- und Entwurfsphase). Hierzu zählt die Erörterung der prinzipiellen Möglichkeiten zur Abbildung der Realität in ein für die Software-Entwicklung brauchbares Modell. Des weiteren wird die in Verbreitung befindliche „Unified Modeling Language" präsentiert. Diese basiert im Gegensatz zu vielen anderen Definitionssprachen nicht auf einer formalen Grundlage, sondern setzt sich aus in der Praxis bewährten Methoden zum Erfassen von komplexen Relationen zusammen.

Im Kapitel 7: „Aufwandsschätzung und Produktivität" wird die Quantifizierung des Software-Entwicklungs-Prozesses hinsichtlich Ressourcenverbrauch (Entwicklungskosten und -dauer) thematisiert. Zunächst wird ein eigener Ansatz zur Leistungsstand-Beurteilung vorgestellt. Die durch empirische Studien verdeutlichte hohe Schwankung der Produktivität führt zur Diskussion der Einflussfaktoren und einer Klassifikation der in der Literatur erwähnten Schätzverfahren. Die Darstellung des derzeit in der Praxis am häufigsten eingesetzten Verfahrens – Function Point Verfahren - und deren Weiterentwicklung schließen dieses Kapitel ab.

Im Kapitel 8: „Metrik-Programme" werden aufbauend auf unterschiedlichen Konzepten des Qualitäts-Managements die in der aktuellen Praxis der Software-Entwicklung am stärksten verbreiteten Modelle zusammengefasst. Die in Kapitel 7 auf die Produktivität bezogenen Maßstäbe werden in diesem Kapitel auf das Kriterium der Qualität und im Sinne eines umfassenden Anspruchs auf Charakteristika der gesamte Organisation ausgedehnt. Die hierzu gebräuchlichen Metriken werden im Detail dargestellt. Darüber hinaus wird auch gezeigt, wie

eine auf Metrik-Programmen basierende kontinuierliche Verbesserung der Abwicklung von Software-Projekten erreicht werden kann.

Im Kapitel 9: „Organisationsgestaltung" werden die Grundlagen zur Gestaltung einer Organisation, deren Zweck die Software-Erstellung ist, dargestellt. Durch die Anforderungen einer flexiblen Projekt-Struktur und die damit einhergehenden oftmals notwendigen Strukturänderungen wird ein in der Praxis praktikabler Weg des Change-Management diskutiert. Abgerundet wird das Kapitel durch eine Auflistung der Qualifikationen der einzelnen Rollenbilder in einem Softwarehaus.

Im Kapitel 10: „Risiko-Management" wird der Software-Entwicklungs-Prozess in ein umfassendes Konzept des Risiko-Managements eingebettet. Ein Software-Entwicklungs-Prozess-Ansatz, der das Risiko-Management in einen evolutionären Software-Entwurf integriert, wird in Kapitel 6 unter dem Begriff: Spiral-Modell dargestellt. Darauf aufbauend konzentriert sich dieses Kapitel auf die Erfolgsfaktoren bzw. Risken im Detail. Im weiteren wird ein Stufenkonzept skizziert anhand dessen man den Entwicklungsstand einer Software-Organisation hinsichtlich des Umgangs mit Risken und Chancen beurteilen kann. Anhand eines selbst weiterentwickelten Modells werden die wesentlichen Dimensionen eines Software-Entwicklungs-Prozesses – speziell der Wirkungsmechanismus von Risken – in Form eines Regelkreises dargestellt.

Im Kapitel 11: „Patterns" wird schließlich eine bis dato weitgehend unbekannte Methode zur Gestaltung des Software-Projekt-Produkt-Programm-Managements gezeigt. Dieses aus der Disziplin der Architektur kommende Schema der Patterns ermöglicht eine mosaikartige Darstellung der relevanten Sachverhalte einer Software-Entwicklungs-Organisation. Der große Vorteil dieser Darstellungsvariante liegt in der gleichzeitigen Adressierung unterschiedlichster Aspekte. Insbesondere nimmt das Schema Rücksicht auf die verschiedenen Anforderungen von Software-Projekten und reduziert dabei den Umfang zur Beschreibung der Erfolgsfaktoren. Abgerundet wird dieses Kapitel durch eine Sammlung relevanter Inhalte für Patterns aus dem Umfeld der Software-Entwicklung.

Im Kapitel 12: „Resümee" wird eine abschließende Zusammenfassung mit Betonung der wesentlichsten Aspekte gegeben.

Im Kapitel 13: „Anhang" werden die detaillierten Kriterien angeführt, die zur Erreichung eines bestimmten Reifegrades einer Software-Organisation entsprechend dem Capability Maturity Model erfüllt sein müssen (Anhang zu Kapitel 8).

Kapitel 14 und 15 beinhalten das Literatur- bzw. Abkürzungsverzeichnis.

Auf der letzten Seite ist der Dissertation der Lebenslauf beigefügt.

1.6 Danksagungen

Zum Zeitpunkt der Drucklegung sind 5 Jahre vergangen, in denen ich mit vielen Unterbrechungen an diesem Werk gearbeitet habe. In dieser Zeit habe ich meine Frau Tamara geheiratet, ist mein Sohn Philip zur Welt gekommen und bin obendrein zum vierten Mal übersiedelt – u.a. auch für 1 Jahr nach München. Obwohl mir in dieser Zeit ungeheuer viele Gedanken zum Thema dieses Werkes sporadisch und ohne viel Zutun in den Sinn kamen, lag doch der Großteil der Arbeit im Sortieren, Gruppieren und letztendlich im Niederschreiben all meiner Ideen.

Die Grundlage für das andauernde Hinterfragen aller Dinge und das Interesse an den Zusammenhängen in unserer Welt lehrten mich meine Eltern – Wilhelm und Maria Exl. Einblicke in andere akademische Disziplinen, die mitentscheidend für diese Arbeit waren, bekam ich dank meiner Brüder – Reinhard erwarb seinen Abschluss in der Disziplin der Betriebswirtschaft und Wolfgang in jener der Rechtswissenschaften – und nicht zuletzt durch meine Frau Tamara, die sich in ihrer Studienzeit ebenfalls der Betriebswirtschaft ausgiebig widmete. Meiner Frau möchte ich aber auch aus jenem Grund danken, dass sie mich stets motivierte meine vor langer Zeit begonnene Arbeit fertigzustellen.

Meinen Kollegen bei Siemens in Wien und München möchte ich für die vielen Diskussionen über komplexe Zusammenhänge im Software-Projekt- und Produkt-Management danken. Sie gaben mir dadurch wertvolle Anregungen die den Inhalt der Dissertation im großen Umfang mitbeeinflussten.

Von der Technischen Universität Wien möchte ich allen Kollegen danken, die mir anregende und wertvolle Hinweise zum Thema dieser Arbeit gaben. Insbesondere möchte ich Herrn Professor Dr. Franz Wojda danken, der mir mit Rat und Tat zur Seite stand, meine Gedanken in wissenschaftlicher Kultur zu formulieren. Und dies von der ersten Stunde an, als außer einer Menge Ideen noch wenig zu Papier gebracht war. Herrn Professor Dr. A Min Tjoa danke ich für die Bereitschaft als Zweitbegutachter zur Verfügung gestanden zu sein.

Wien, im April 2004 Christian Exl

2. Projekt–Produkt–Programm–Management

Bevor auf den Inhalt von Projekt-, Produkt- und Programm-Management näher eingegangen wird, erfolgt in diesem Kapitel eine Begriffsbestimmung entsprechend dem im deutschsprachigen Raum üblichen Gebrauch.

2.1 Begriffsbestimmungen

2.1.1 Projekt

Unter dem Begriff Projekt wird ein komplexes Vorhaben verstanden, das durch die folgenden Randbedingungen gekennzeichnet ist [1]):

- Zielvorgabe in Form einer Aufgabenstellung (Projektdefinition)
- Personelle, sachliche, finanzielle und zeitliche Abgrenzung gegenüber anderen Vorhaben
- Nichtreversible Projektbedingungen
- Arbeitsteilige Projektorganisation durch adäquate Projekt-Strukturierung

Eng verwandt ist der Begriff des Projekts mit jenem des Systems. Charakteristisch ist hierbei insbesondere die Abgrenzung der zur Disposition stehenden Elemente, die untereinander und mit der Systemumwelt (außerhalb des Systems) in Beziehung stehen.

2.1.2 Projekt-Management

Das Projekt-Management umfasst die Gesamtheit aller Tätigkeiten, ein Projekt innerhalb des vorgegebenen Termin- und Kostenvolumens vertragsmäßig abzuwickeln [2]). Die im Rahmen des Projekt-Managements wichtigsten Aufgaben umfassen folgende Funktionen [3]):

- Projektplanung
    - Projektdefinition
    - Aufgabengliederung

[1] Wojda, Franz, Projektorganisation-Projektmanagement, Wien, 2001
[2] Wojda, Franz, Projektorganisation-Projektmanagement, Wien, 2001
[3] Patzak, Gerold, Rattay, Günter, Projektmanagement, Wien, 1998

- Umfeldanalyse
- Terminplanung
- Ressourcenplanung
- Finanzplanung

- Projektorganisation
  - Projektkultur
  - Rollendefinition
  - Kompetenzverteilung
  - Kommunikationsgestaltung

- Projektteamführung
  - Mitarbeiterauswahl
  - Führung
  - Motivation

- Projektcontrolling
  - Kontrolle kritischer Erfolgsfaktoren
  - Regelung der Projektplanung

Der Ursprung des Projekt-Managements geht auf die logistischen Leistungen der amerikanischen Streitkräfte im 2. Weltkrieg zurück. Die über große Gebiete ausgedehnten Operationen im Pazifik erforderten umfassende Planungen. Im weiteren Verlauf wurden die erworbenen Management-Erkenntnisse für Projekte der Weltraumforschung und schließlich in der Industrie und im öffentlichen Dienst angewandt.

2.1.3 Projekt-Programm-Management

Strukturiert man die auf ein Projekt einwirkenden Umwelteinflüsse selbst wieder in Form von Projekten lässt sich zwischen den Projekten eine mehr oder minder starke Kopplung feststellen. Eng gekoppelte Projekte erfordern aufgrund ihrer gegenseitigen Beeinflussung ein übergeordnetes Management, das als Projekt-Programm-Management (PPM) bezeichnet wird. Die Projekte eines Programms stehen hinsichtlich Inhalt, Terminen, Ressourcen und Kosten in enger Beziehung.

Durch diese gegenseitigen Abhängigkeiten lassen sich die im Rahmen des PPM zu bewältigenden Aufgaben folgendermaßen strukturieren [4]):

[4] Wojda, Franz, Lernende Organisation durch Projekt-Programm-Management, in: Office Management 1-2, 1996

- Projektinitiierung:
  Die Initiierung eines Projekts kann vielfältigste Gründe haben. Grundsätzlich ist zu unterscheiden, ob es extern (Auftrag durch einen Kunden) oder intern initiiert wird. Im Falle eines externen Projekts muss ein wesentliches Augenmerk auf das Customer-Relationship-Management gelegt werden. Beiden Arten ist jedoch gemein, dass das als Projekt angedachte Vorhaben zunächst formalisiert und als Projektantrag einer Bewertung zugeführt wird. Im Falle eines positiven Projekt-Entscheids wird der Projektantrag im Detail ausgearbeitet. Zum Inhalt zählt u.a.:

  - Basisdaten
  - Präambel
  - Ausgangssituation
  - Zielsetzung
  - Projektnutzen
  - Umfang und Inhalt
  - Schnittstellen
  - Risikobeurteilung
  - Projektorganisation
  - Ablauf- und Terminplanung
  - Aufwandsabschätzung

- Projekt-Programm-Bildung:
  Aufgrund der Bewertung des Projektantrags wird in einem zweiten Schritt die Kompatibilität mit den bereits existierenden Projekten und Projekt-Programmen geprüft. Im Vergleich zu einzelnen Projekten gestatten Projekt-Programme eine Vergleichbarkeit der Projekte hinsichtlich des wirtschaftlichen Nutzens für das Unternehmen. Ein ausgezeichnetes Instrument hiefür sind die unterschiedlichsten Portfolio-Methoden. Je nach Strategievorgabe können sie an die individuellen Anforderungen des Unternehmens adaptiert werden. In weiterer Folge werden diese Konzepte anhand der Portfolio-Matrix (von: The Boston Consulting Group) und dem EVA-Konzept (Economic Value Added – Konzept von: Stern Stewart & Co.) vorgestellt. Einschränkend muss angemerkt werden, dass das EVA-Konzept nur in den Fällen sinnvoll angewandt werden kann, wo aus einem Projekt ein Zusatznutzen in Form wiederholter Vermarktung gewonnen werden kann. Portfolio-Konzepte eignen sich insbesondere zur optimalen Verteilung des Ressourceneinsatzes, und hier nicht nur zur optimalen Verteilung zwischen den Projekten, sondern auch hinsichtlich des Zeitablaufs.

- Jahresplanung:
  Im Rahmen der Jahresplanung erfolgt die Neubewertung des Projektportfolios. Dies geschieht auf der Grundlage modifizierter Strategien (z.b. aufgrund veränderter Umweltbedingungen). Wichtiger Bestandteil der Jahresplanung ist die Genehmigung des Budgets.

- Projektabwicklung:
  Die Grundlage bildet die klassische Lehre des Projekt-Managements. Zentrale Themen sind hierbei die

  - Projektplanung mit den Teilaufgaben: Projektdefinition, Umfeldanalyse, Aufgabengliederung, Terminplanung, Ressourcenplanung und Kostenplanung;
  - Projektorganisation mit den Teilaufgaben: Rollendefinition, Kompetenzverteilung, Kommunikationsgestaltung und Schaffung einer Projektkultur;
  - Projektteamführung mit den Teilaufgaben: Mitarbeiterauswahl, Förderung der Teammitglieder und Motivation;
  - Projektcontrolling mit der Verfolgung der kritischen Erfolgsfaktoren: Termine, Ressourcen, Kosten und Qualität.

- Portfolio-Controlling:
  Ergänzend zum Projekt-Controlling müssen auch die gegenseitigen Abhängigkeiten und die Übereinstimmung mit der Jahresplanung überprüft werden.

- Erfolgsfeststellung:
  Obwohl die Erfolgsfeststellung auch als Teil des Controlling-Prozesses angesehen werden kann, soll sie aufgrund der oftmaligen Unterschätzung – vor allem während der Projektlaufphase – extra erwähnt werden.

2.1.4 Projekt-Produkt-Programm-Management

Im Gegensatz zum Projekt-Management liegen die Schwerpunkte beim Produkt-Management in folgenden zwei Sachverhalten. Zum einen ist beim Produkt-Management das Ergebnis der betrieblichen Leistungserstellung im Mittelpunkt des Interesses, und nicht das Produktions-Umfeld. Und zum zweiten impliziert der Begriff: Produkt die serienmäßige Vermarktung von Software-Projektergebnissen. Im Rahmen der Software-Entwicklung liegt der Hauptunterschied in der Berücksichtigung der Anforderungen mehrerer Kunden, einer versionsgetriebenen Weiterentwicklung und oft in höheren Qualitätsansprüchen. Hinsichtlich der finanziellen Ressourcen ist der wesentliche Vorteil eines mehrfachen Absatzes die Kostenaufteilung entsprechend der Anzahl verkaufter Software-Lizenzen.

2.1.5 Systemgeschäft

Aus der Sicht des Absatzmarktes wird ein Projekt in Form eines Systemgeschäfts vermarktet. Dies ist dadurch gekennzeichnet [5]), dass die Leistungsangebote oder Problemlösungs-Systeme sich als ein durch die Vermarktungsfähigkeit abgegrenztes, von einem oder mehreren Anbietern in einem geschlossenen Angebot erstelltes Anlagen-Dienstleistungs-Bündel (Hardware-Software-Kombination) zur Befriedigung eines komplexen Bedarfs darstellen.

Im Gegensatz zum Produktgeschäft und traditionellen Anlagengeschäft ist das Systemgeschäft geprägt durch zusätzliche, über Engineering-Leistungen hinausgehende, umfangreiche Dienstleistungen (System-Software) in Form von Pre-Sales-Services, After-Sales-Services, sowie episodenbegleitende Dienstleistungen des Systemanbieters während der Leistungserstellung (z.b. Projektorganisation oder Projektmanagement). Typische Merkmale des Systemgeschäfts sind:

- hohe Komplexität des Hardware-Software-Bündels
- Langfristigkeit des Beschaffungs-, Entscheidungs-, Erstellungs- und Abwicklungs-Prozesses
- Anbieterkoalitionen
- hohe Interaktionskomplexität
- hoher Auftragswert
- Internationalität des Geschäfts
- Bedeutung der Auftragsfinanzierung
- hohe Individualität der Endleistung
- Bedeutung von Referenzanlagen

[5] Gabler-Wirtschafts-Lexikon, Wiesbaden, 1997

3. Betriebswirtschaftliche Theorien

Im Rahmen des Programm-Managements werden potentielle Projekte aufgrund einer Gegenüberstellung mit der Unternehmens-Philosophie angenommen oder abgelehnt. Da diese Thematik (basierend auf eigenen Erfahrungen) oft vernachlässigt wird, geben die nachfolgenden Abschnitte eine Übersicht über theoretische Konzepte anhand derer die Entscheidungs-Vorbereitung, bzw. die Zusammenstellung des Projekt-Portfolios vorgenommen werden sollte. In den nachfolgenden Abschnitten wird auf diese Theorien an geeigneter Stelle Bezug genommen.

3.1 Grundlegende Theorien

Ebenso wichtig wie das Ziel ist der Weg der dorthin führt. Obwohl einzelne Arbeitspakete aufgrund allgemein akzeptierter Verfahrensmethoden die einzelnen Arbeitsschritte größtenteils implizieren, ergeben sich beim Betrachten eines größeren Umfangs bzw. über eine größere Zeitspanne hinweg merkliche Unterschiede zwischen Software-Projekten. Diese Unterschiede bestehen selbst unter Annahme keiner systematischen Fehler aufgrund der unterschiedlichen Zielsysteme des Umfeldes. Um diesen Sachverhalt mit einem Beispiel zu belegen, sei auf die unterschiedlichen Handlungsweisen zwischen Fixpreis-Geschäften und nach Arbeitsstunden vergebenen Beauftragungen verwiesen.

Um ein Zielsystem zu formalisieren, bedient man sich der Errungenschaften des Strategischen Managements. Dieses aus der militärischen Disziplin in die Betriebswirtschaftslehre übernommene Gedankengut wurzelte zunächst in der Spieltheorie. So entspricht die Strategie eines Spielers einem vollständigen Plan, der für alle denkbaren Situationen eine richtige Wahlmöglichkeit beinhaltet. Dieser Plan, bei dem der Spieler sowohl die eigenen Aktionen als auch die der Gegner simultan und antizipativ berücksichtigt, wird Strategie genannt.[1]) Die Theorien zum Strategiebegriff lassen sich in zwei Gruppen einteilen. Zum einen ist es das klassische Verständnis, wonach eine Strategie das Ergebnis rationaler Planungen ist. Barny [2]) ordnet dieser Betrachtung einige charakteristische Merkmale zu:

- Strategien setzen sich aus mehreren zusammenhängenden Einzelentscheidungen zusammen
- Strategien lassen sich hierarchisch, ausgehend von einer umfassenden Mission, darstellen

[1] Neumann, I. von, Morgenstern, O., Spieltheorie und wirtschaftliches Verhalten, 1961
[2] Barny, J.B., Gaining and Sustaining Competitive Advantage, New York, 1997

- Strategien basieren auf dem SWOT-Ansatz (Strengths, Weaknesses, Opportunities, Threats)
- Strategien führen schließlich zur Ressourcen-Allokation

Im Duden [3]) wird "Strategie" (vom griechischen "strategia" abgeleitet) als "genauer Plan des eigenen Vorgehens, der dazu dient, ein militärisches, politisches, psychologisches o.ä. Ziel zu erreichen, in dem diejenigen Faktoren, die in die eigene Aktion hineinspielen könnten, von vornherein einzukalkulieren versucht werden" erklärt. Ein klassisches Beispiel hierzu ist das Buch von Sunzi "Die Kunst des Krieges" [4]).

Demgegenüber geht die Schule um Mintzberg [5]) von praktischen Beobachtungen aus, die eine Strategie als Plan, List, Muster, Position, oder allgemein als Denkhaltung sieht. Des weiteren folgert er aus empirischen Studien ein Grundmuster von Strategietypen:

- Geplante Strategien, die nicht realisiert wurden
- Realisierte Strategien, die nicht geplant waren
- Geplante Strategien, die auch realisiert wurden

Eine interessante Abhandlung, die diesem Ansatz von Strategie entspricht, kommt von A. Grove [6]) , die den Titel trägt: "Nur die Paranoiden überleben". Darin führt er den Erfolg auf die Erkennung strategischer Wendepunkte zurück, an denen sich die gesetzten Handlungen grundauf ändern müssen.

In der weiteren Folge möchte ich einige theoretische Perspektiven skizzieren, die wertvolle Gedanken zum SPM systematisieren. Eine Notwendigkeit dafür resultiert aus dem Umstand, dass das Strategische Management einen multidisziplinären Charakter hat. Neben Theorien über die Kriegsführung (wie oben erwähnt), bedient es sich u.a. politischer Theorien, mathematischer Theorien, wie der Chaos- oder Spieltheorie, aber auch Erkenntnissen über einfache Darstellungsmöglichkeiten komplexer Sachverhalte.

In Kritiken sind vor allem die präskriptiven von den deskriptiven Ansätzen zu unterscheiden. Während deskriptive Ansätze Strategien empirisch beschreiben, geben präskriptive Ansätze Empfehlungen ab. Sie sind aufgrund ihres höheren Abstraktionsgrades leichter kritisierbar. Da die deskriptiven Ansätze vom Endzustand (Ende der Strategie-Implementierung) ausgehen, ist die Ungewissheit des Erfolges oder Nichterfolges gegenüber präskriptiven geringer. Die im folgenden angeführten Theorien werden entsprechend Welge u.a. [7]) folgendermaßen gruppiert:

---

[3] Duden, 1996
[4] Sunzi, Die Kunst des Krieges, zwischen 300 und 500 v.Chr., hrsg. von Clavell, James, München, 1988
[5] Mintzberg, H., et al., Strategy Safari. A Guided Tour Through The Wilds Of Strategic Management, NY, 1998
[6] Grove, Andrew S., Nur die Paranoiden überleben, Frankfurt/Main, 1997
[7] Welge, M.K., Al-Laham, A., Kajüter, P., (hrsg.), Praxis des strategischen Managements, Wiesbaden, 1999

1. betriebswirtschaftliche, rational-entscheidungsorientierte Theorien
2. volkswirtschaftliche, ökonomische Theorien
3. interdisziplinäre, systemtheoretisch-evolutionäre Theorien

3.1.1 Betriebswirtschaftliche, rational-entscheidungsorientierte Theorien

Diesen Theorien liegt ein Prozessansatz zugrunde, welcher auch die fundamentale Basis für das SPM ist. Der Prozess gliedert sich in zwei Phasen, der Strategieformulierung und der Strategieimplementierung.

Die Strategieformulierung umfasst (wird u.a. im SWOT–Ansatz abgebildet):

- strategische Umweltanalyse
- strategische Unternehmungsanalyse
- Festlegung strategischer Ziele

Die Strategieimplementierung umfasst all jene Aktivitäten, die zur Umsetzung der formulierten Strategie notwendig sind. Dies können z.b. die Gestaltung der Aufbau- und Ablauforganisation sein oder auch die Auswahl der Mitarbeiter, die für die Implementierung verantwortlich sind. Erste, umfassende Inhalte arbeitete Ansoff 1988 [8]) aus:

- Förderung des Entrepreneurgedankens unter den Mitarbeitern
- Diagnose des Managements hinsichtlich Werte, Machtstrukturen, Potentiale, und anderen Charakteristika
- strategische Veränderung der Potenziale (z.b. Know How der Mitarbeiter)
- Betonung auf strategische und operative Ziele (z.b. Budgets)
- Handhabung von diskontinuierlichen Ereignissen (z.b. Konflikte)

Eine erste geschlossene Konzeption, die sowohl die Strategieformulierung als auch die Implementierung umfasst, geht auf Learned, Christensen, Andrews und Guth zurück. Sie wurde unter dem "LCAG-Schema" der Harvard Business School bekannt. Die für die Strategieformulierung notwendigen Analysen konzentrieren sich auf:

- Chancen und Risken in der Unternehmungsumwelt (technologische, ökonomische und ökologische Parameter)
- Stärken und Schwächen der eigenen Unternehmung (sachliche, finanzielle und personelle Ressourcen)

[8] Ansoff, H.I., The New Corporate Strategy, New York, 1988

Diese Aspekte sind im brancheneigenen Umfeld und in einer globaleren (staatsweit oder z.b. EU-weit) Einflusssphäre zu sehen. Hinzu kommt noch die zeitliche Dimension, die auch deren Änderungen berücksichtigt.

- Um diese Bewertung zu objektivieren, werden im nächsten Schritt die Wertvorstellungen der die Strategie formulierenden und der die Strategie implementierenden Mitarbeiter integriert.

- Interessant ist die vierte Einflussgröße, die die Verantwortung der Unternehmung gegenüber der Gesellschaft zu berücksichtigen versucht, da sie bei Anwendung der ersten 3 Aspekte auch als Teilziel in der Strategieformulierung identifiziert werden kann, und damit nicht eigens aufgeführt zu werden braucht.

Die Strategieimplementierung umfasst 3 wesentliche Aspekte:

- Gestaltung der Aufbauorganisation (*)
- Gestaltung verhaltensbezogener Aktivitäten (Motivations-, Kontroll-, Personal-Entwicklungssystem, u.a.)
- personelle und organisatorische Gestaltung der obersten Führungsebene

(*) Die Aufbauorganisation wird heute wesentlich durch das interne Informationssystem mitbestimmt. In Unternehmungen, in denen der Produktionsfaktor Information im Zentrum der Wertschöpfung steht, können z.b. anhand Web–basierter Darstellungen virtuelle Aufbauorganisationen etabliert werden, die einem hierarchischen Modell erheblich widersprechen können, oder andererseits, es erst "zum Leben erwecken".

Aufbauend auf diese Grundpfeiler der rational–entscheidungsorientierten Ansätze werden ferner Lernprozesse, Flexibilität der Strategie und dezentrale, partizipative Merkmale eingearbeitet.

3.1.2 Volkswirtschaftliche, ökonomische Theorien

Das **Structure-Conduct-Performance – Paradigma** (= SCP–Paradigma) zerlegt ein Gesamtsystem nach dem Prinzip von Ursache, Transmission und Wirkung in drei Teilsysteme. Als Ursachen werden Eigenschaften des Wettbewerbsumfeldes (structure), z.b. Anzahl und Größe der Wettbewerber, Produktdifferenzierung, Nachfrageelastizität, usw., und als Wirkung Ausprägungen des Erfolges der Unternehmung (performance), z.b. Rendite, Kostenminimierung, u.a., verstanden. Die Transmission bildet schließlich die Strategie der Unternehmung, z.b. Expansion, Preispolitik, usw., ab. Weiterentwicklungen dieses Ansatzes sind unter dem Begriff der "Neuen Industrieökonomik" zu finden.

Spieltheoretisch basierende Ansätze konzentrieren sich auf die Interdependenz–Problematik zwischen den am selben "Spiel" (= selbes System) beteiligten Akteuren. Teilt man die Zukunft in Zeit-Segmente, dann berücksichtigt eine Strategie alle Kombinationen von Reaktionen in allen zukünftigen Zeitsegmenten. Die Vergangenheit definiert hierbei die Anfangsbedingungen. Um sinnvolle Ergebnisse zu erzielen, ist auf eine Limitierung der Komplexität Wert zu legen. Diese Randbedingungen werden in der Praxis z.b. durch Gesetze geschaffen. Ein in diesem Modell wesentlicher Faktor ist die in der Praxis anzufindende Unvollständigkeit des Informationsstandes der Akteure. Diesem Umstand tragen Begriffe wie "Fuzzy Logic" Rechnung. [9])

Von größerer Bedeutung für das SPM haben **Ansätze der Neoinstitutionenökonomik**. Im Zentrum stehen identische Annahmen über das menschliche Verhalten innerhalb von Kulturkreisen. [10]) In unserer westlichen Kultur werden die Handlungen der Individuen insbesondere durch folgende Eigenschaften geprägt:

- individuelle Nutzenmaximierung
- begrenzte Rationalität
- opportunistisches Verhalten

Obwohl ihre Bedeutung von niemandem bestritten wird, mangelt es dennoch an der Berücksichtigung der von ihnen ausgehenden Kräfte. Die Akzeptanz der begrenzten Rationalität der Akteure steht hierbei im krassen Gegensatz zur weit verbreiteten Auffassung eines vollkommenen Wettbewerbs, wie er in der Neoklassik vehement vertreten wird (siehe hierzu auch das 7-D-Modell nach Christian Exl welches im Rahmen der Risikodiskussion vorgestellt wird).

Ein für das SPM ebenfalls bedeutender Ansatz ist die **Principal-Agent – Theorie**. Nach Gabler [11]) ist sie ein Zweig der Wirtschaftstheorie, der die Kooperation zwischen Wirtschaftssubjekten beim Vorliegen von Interessenkonflikten und Informationsasymmetrien zum Gegenstand hat. Ausgangspunkt ist die Principal-Agent – Beziehung, bei der definitionsgemäß eine Partei (der Agent) im Auftrag einer anderen Partei (dem Principal) agiert. Diesem Modell wird als Anfangsbedingung unterstellt, dass Individuen ihr Eigeninteresse verfolgen. Folglich kann nicht davon ausgegangen werden, dass der Agent automatisch im besten Interesse des Principals handelt. Als Lösungsansatz konzentriert sich die PA–Theorie auf Verträge zwischen Principal und Agent, die auf für beide Seiten beobachtbaren Größen aufbauen. Die Verträge sind derart gestaltet, dass das Ziel des Principals über Incentives erreicht werden soll.

---

[9] Heß, G., Marktsignale und Wettbewerbsstrategie, Stuttgart, 1991
[10] Picot, A., Dietl, H.M., Franck, E., Organisation. Die ökonomische Perspektive, Stuttgart, 1997
[11] Gabler-Wirtschafts-Lexikon, Wiesbaden, 1997

Diese Überlegungen bilden die theoretische Basis sowohl in den Beziehungen zwischen Auftraggeber und Auftragnehmer von Software-Projekten, als auch innerhalb der Linien- und Projekthierarchien einer Software-Organisation (vgl. hierzu das 7-Disziplinen-Modell nach Exl im Kapitel 10).

Während die obigen Prämissen auch auf andere Agency–Theorien zutreffen, hebt die PA–Theorie die asymmetrische Verteilung der Information auf Principal und Agent hervor. Sie kann sich auf mehrere Arten äußern, von denen folgende drei Extremsituationen von Bedeutung sind:

- Zwischen Principal und Agent liegen bereits zum Zeitpunkt des Vertragsabschlusses Differenzen bezüglich Umweltzustand vor. Es wird auch als Modell mit versteckten Charakteristika bezeichnet.

Herrscht allerdings Übereinstimmung bezüglich Umweltzuständen zum Zeitpunkt des Vertragsabschlusses, kann es dennoch in weiterer Folge zu folgenden Asymmetrien kommen:

- der Principal kennt den Umweltzustand, aber nicht die vom Agent vorgenommenen Aktionen
- der Principal kennt die vom Agent vorgenommenen Aktionen, aber nicht den Umweltzustand

Im praktischen Arbeitsablauf eines Software-Projektleiters entstehen derartige Informationsasymmetrien alleine schon aufgrund des tagtäglichen EMail-Verkehrs. Geht man z.B. von der Annahme aus – die auf eigenen Erfahrungen beruht – dass ein Software–Projektleiter 30% seiner Arbeitszeit für das Verfassen und Lesen von EMails nutzt, kann man sich leicht vorstellen, wie es einem Auftraggeber ergeht, der für einige Projekte verantwortlich ist und sämtliche EMails seiner Projektleiter als Kopie (cc) erhält.

Neben der Informationsasymmetrie lassen sich PA–Probleme durch die Häufigkeit und Dauer der Interaktionen zwischen Principal und Agent kennzeichnen (statische und dynamische Modelle). Eine weitere Charakteristik von PA–Beziehungen handelt von der Anzahl der Principals und Agents, die an der Beziehung beteiligt sind. Hierbei sind die 1xP : nxA und die nxP : 1xA Beziehungen herauszustreichen.

Die Aufwände zur Etablierung und zum Erhalt der PA-Beziehung werden als *Agency-Kosten* bezeichnet. Sie können eingeteilt werden in:

- Monitoring costs: Ressourcenaufwendungen des Principals zur Überwachung des Agents
- Bonding costs: Aufwendungen seitens des Agents um ein vom Principal unerwünschtes Verhalten glaubhaft auszuschließen

- Residual loss: Wohlfahrtsverluste, die dem Principal entstehen, wenn der Agent ein abweichendes Ziel verfolgt

Um die Agency-costs zu minimieren kam Hüttemann [12]) auf empirischem Wege zu folgenden Maßnahmen:

- Instrumentalisierung des Anreizsystems (wichtigster Erfolgsfaktor)
- Regelmäßige Beurteilung durch Vorgesetzte und Einsatz langfristiger Erfolgsindikatoren
- Diese Maßnahmen bedingen eine hohe Kontrollintensität

Eine weitere der PA–Theorie ähnliche Theorie ist die **Transaktionskosten–Theorie**. Nach Ebers und Gotsch [13]) treten Transaktionskosten dann auf, wenn opportunistisch handelnde, mit begrenzter Rationalität ausgestattete Wirtschaftssubjekte spezifische und mit Unsicherheit behaftete Transaktionen eingehen (z.b. Kommunikationskosten für Vereinbarungen, Abwicklungen und Kontrollen), und es dabei zu einer asymmetrischen Informationsverteilung zwischen den Transaktionspartnern kommt.

Transaktionskostentheoretische Überlegungen spielen z.b. in der vertikalen Integration der Softwareentwicklung eine bedeutende Rolle. Komplexe gegenseitige Abhängigkeiten bedingen so Rahmenverträge, die zu institutionalisierten Anreiz- und Sanktionssystemen, Kapitalbeteiligungen und gemeinsamem Ressourceneinsatz führen.

Ein weiterer Ansatz, der vor allem in wissensbasierten Branchen (z.b. Softwarebranche) relevant ist, konzentriert sich auf die Kernkompetenzen einer Unternehmung (= **Ressourcenorientierter Ansatz**). Die Anfänge dieses Ansatzes gehen auf Penrose, 1959 zurück. Sie heben die Bedeutung der Kernkompetenzen für den Wettbewerbsvorteil einer Unternehmung hervor. Leonard-Barton [14]) argumentiert, dass sich Technologie-, Prozess-, Management- und andere Kenntnisse zu einem systemischen Ganzen aggregieren und erst damit den Charakter von Kernkompetenzen der Unternehmung annehmen. Die darin innewohnende Komplexität ist kaum uneingeschränkt kopierbar, wodurch ein erworbener Wettbewerbsvorteil für gewöhnlich von längerem Nutzen ist.

Eine ausführliche Studie über den Zusammenhang von Wissenspotenzialen und Wettbewerbserfolg haben 1996 Bierly und Chakrabarti publiziert. In der amerikanischen Pharmazie-Branche identifizierten sie 4 verschiedene Wissensstrategien, von denen insbesondere die Gruppen der *Innovatoren* und der *Ausbeuter* für diametrale Strategien stehen.

---

[12] Hüttemann, H.H., Anreizsysteme in schrumpfenden Unternehmungen, Wiesbaden, 1993
[13] Ebers, M., Gotsch, W., Institutionenökonomische Theorien der Organisation, in: Organisationstheorien, hrsg. v. A. Kieser, Stuttgart-Berlin, 1999
[14] Leonard-Barton, D., The Factory as a Learning Laboratory, in: SMR, 33.Jg., 1992

Innovatoren verfolgen eine aggressive Wissensstrategie, die interne und externe Wissensquellen nutzt, und inkrementales als auch radikales Lernen forciert. *Ausbeuter* konzentrieren sich hingegen auf externe Wissensquellen und inkrementale Lernprozesse. Sie haben geringe FuE–Ausgaben und beuten bestehende Patente maximal aus. Was den Profit anging, lagen die Innovatoren an erster Stelle.

3.1.3 Evolutionäre Theorien

Diesen Theorien liegt die Auffassung zugrunde, dass Unternehmungen – und auch Teile davon - zu komplex sind, um durch Strategien beliebig gestaltet werden zu können. Dagegen sind schrittweise evolutionäre Veränderungen innerhalb eines eingeschränkten Gestaltungsspielraumes sehr wohl möglich.

Die Begründung für die Wahl eines evolutionstheoretischen Konzeptes sehen Kieser und Woywode in 3 Gründen [15]):

- Politische Interessengruppen innerhalb einer Unternehmung verfolgen unterschiedliche Ziele und taktische Manöver
- das Management verfügt nur über unvollständige Informationen hinsichtlich Zweck-Mittel-Relationen
- die organisatorische Trägheit verhindert das Schritthalten von Anpassungen an die geänderten Umweltbedingungen

Aufgebaut ist die Theorie mit den Begriffen der Evolutionslehre, wie sie von Charles Darwin in dem mittlerweile klassischen Werk über "Die Entstehung der Arten" aus dem Jahr 1859 geprägt wurden. [16]) Aus Gründen der Vollständigkeit möchte ich hierbei erwähnen, dass erste Überlegungen bezüglich der Entstehung der Arten schon um 1795 von Goethe in Deutschland, Erasmus Darwin in England und Geoffroy Saint-Hilaire in Frankreich angestellt wurden, und, dass sie bis heute aufgrund geologischer und paläontologischer Funde nicht unumstritten. [17]) Ungeachtet dieses Sachverhalts, schafft diese Art der Betrachtung eine neue, interessante Perspektive.

Legt man so z.b. den Begriff der *Population* – der in der Biologie einer Spezies entspricht – auf die Welt der Unternehmungen um, wird damit zum Ausdruck gebracht, dass die Strukturen und Prozesse einer Branche nicht ohne weiteres auf eine andere angewandt werden können.

---

[15] Kieser, A. (hrsg.), Woywode, M., Evolutionstheoretische Ansätze, in: Organisationstheorien, Stuttgart, 1999
[16] Darwin, Charles R., On the Origin of Species by Means of Natural Selection, London, 1859
[17] Zillmer, Hans-Joachim, Darwins Irrtum, München, 2000

Ebenso lassen sich auch *Variation* und *Selektion* auf die Entwicklung von Unternehmungen - aber auch auf Teilbereiche davon – anwenden. Basierend auf diesen Überlegungen ist der später erklärte Prozess der evolutionären Softwareentwicklung anzusehen.

3.1.4 Empirische Betrachtungen

Aufgrund der relativ neuen Disziplin des SPM gibt es über die zu beobachtenden betrieblichen Abläufe, die die Softwareentwicklung begleiten, noch wenige empirische Studien. Aus diesem Grund sollten zunächst jene Beobachtungen widergegeben werden, die im Rahmen von Studien zum strategischen Management gesammelt wurden. Nach Al-Laham [18]) konzentrieren sich die Studien einerseits auf den Inhalt der Strategien – wie: Mergers & Acquisitions, Globalisierung, Allianzen, Joint Ventures, usw. – und andererseits auf die Aktivitäten, die die Inhalte der Strategien liefern, oder deren Implementierung begleiten.

Die im Anschluss dargestellten Studien geben eine Übersicht über den Charakter der Entscheidungsprozesse, wie sie auch in frühen Phasen des SPM anzutreffen sind.

3.1.4.1 Begrenzte Rationalität der SPM–Prozesse

H.A. Simon [19]) beobachtete das Entscheidungsverhalten von Individuen und fokussierte seine Studien auf die Folgen der begrenzten Informations-Verarbeitungs-Kapazität des Menschen. Demnach werden nur wenige Alternativen in Betracht gezogen und jene Lösungen favorisiert, die in der Nachbarschaft des Vertrauten liegen. Im weiteren ist das Entscheidungsverhalten durch eine unzureichende Problemdefinition und eine Reduzierung von Unsicherheiten gekennzeichnet. Cyert und March interpretieren Unternehmungen als Koalitionen von Interessensvertretern, die je nach deren Einflüssen die Ziele als Kompromisslösungen aushandeln.

[18] Al-Laham, A., Strategieprozesse in deutschen Unternehmungen. Verlauf, Struktur und Effizienz, Wiesbaden, 1997

[19] Simon, H.A. (hrsg.), A Behavioural Model of Rational Choice, in: Models of Man, New York – London, 1957

3.1.4.2 Begrenzte Phasenstruktur der SPM-Prozesse

Witte [20]) konzentrierte seine Arbeiten auf die Phasenstruktur von Prozessen. In empirischen Studien zeigten sich beträchtliche Abweichungen vom vorgegebenen Phasenschema der Strategieprozesse. Teilweise wurden einzelne Phasen übersprungen, mehrfach durchlaufen, oder es ließ sich überhaupt keine Gliederung erkennen. Von Bedeutung war jedoch die Erkenntnis, dass eine stärkere Strukturierung der Prozesse einen positiven Erfolgsbeitrag leistete.

3.1.4.3 Prozessvarianten

Nutt [21]) ging von einem Referenzmodell für Strategieprozesse aus, das folgende Phasen umfasste:

1. Problem Formulation
2. Concept Development
3. Strategy Detailing
4. Strategy Evaluation
5. Strategy Implementation

Darauf basierend stellte er fest, dass es in 85% der Fälle zu keiner expliziten Lösungssuche und Strategie-Entwicklung kommt (Phase 2). Hierbei wird entweder eine Strategie der Konkurrenz übernommen, oder eine Strategie aus einer Menge bereits vorhandener Strategien ohne explizite Diskussion ausgewählt.

3.1.4.4 Kontingenz von Entscheidungs-Prozessen

Aufgrund neuerer Studien, wonach kein eindeutiges Optimum für einen Strategieprozess gefunden werden konnte, betont man die *Kontingenz* – die situations-spezifische Abhängigkeit des Strategie-Prozesses. [22]) Hierbei werden Strategien je nach der von innen oder außen bedingten Änderung ausgewählt.

---

[20] Witte, E., Hauschildt, J., Grün, O., (hrsg.), Innovative Entscheidungsprozesse – Das Ergebnis des Projekts Columbus, Tübingen, 1988

[21] Nutt, P.C., Planning Process Archetypes and their Effectiveness, in: Decision Science, 15.Jg., 1984

[22] Staehle, W., Management, München, 1990

3.2 Strategische Konzepte

3.2.1 Das Portfolio-Konzept

Die Portfolio-Technik entstand in den 70er Jahren im Zuge der Neustrukturierung großer Unternehmen, wie z.B. General Electric, die in etwa 170 selbständige Einheiten gegliedert war, ohne jedoch Synergien nutzen zu können. Von der Beratungsgesellschaft McKinsey wurde in diesem Zusammenhang der Begriff der **Strategischen Geschäfts Einheit** (SGE) geprägt. *Kriterium für die Bündelung in SGEs war die Produkt-Markt-Technologie Kombination.* Bezüglich der Abgrenzung zwischen des SGEs waren folgende Voraussetzungen zu erfüllen:

- Funktion als Wettbewerber am externen Markt für ein eindeutig definiertes und klar abgrenzbares Kundenproblem
- weitgehende Eigenständigkeit hinsichtlich Preis, Qualität, Einkauf, Distribution und Service
- jede SGE leistet einen eigenständigen Beitrag zur Steigerung des Erfolgspotentials

Basis für die Segmentierung in SGEs ist die Segmentierung der Umwelt in Strategische Geschäftsfelder (SGFs), die z.B. nach

- Produkt-Funktion,
- Produkt-Technologie,
- Kunden-Region (geographisch) und
- Kunden-Gruppe (Industrie, private Haushalte, usw.)

erfolgen kann.

Die in der Praxis nicht völlig zu eliminierenden Kopplungen oder Interdependenzen resultieren häufig in

- Ressourcen-Interdependenzen (finanziell, personell, sachlich). So wird z.b. ein zentraler Einkauf mit der damit verbundenen Preisreduktion behindert
- Markt-Interdependenzen (konkurrierend, komplementär)

Wird größerer Wert auf die Diversifikation in unterschiedliche SGFs gelegt, könnte durch die Strategie-Wahl das jeweilige Marktsegment optimal bedient werden. Allerdings verhindern die in diesem Fall stark in Erscheinung tretenden Interdependenzen die Entscheidungs-Autonomie.

Dieser Außensegmentierung in SGFs folgt nun eine weitgehend entsprechende Innensegmentierung in SGEs. Aufgrund verschiedener Einwände, wie z.b., dass sich

- der Markt schneller verändert als dies für eine Umstrukturierung der Aufbauorganisation möglich ist, oder,
- eine völlige Dezentralisierung aller Aufgaben in SGEs kosten-ineffizient ist,

besteht auch die Möglichkeit einer Matrix-Organisation mit folgenden Ausprägungen:

- organisatorische Einheit und SGE identisch
- eine organisatorische Einheit besteht aus mehreren SGEs
- eine SGE erstreckt sich über mehrere organisatorische Einheiten

Die zuletzt angeführte Struktur führt in der Praxis zu erheblichen Differenzen, da für die Realisierung des Ziels des SGE's Ressourcen von zwei Organisationseinheiten bereitgestellt werden, die ihrerseits unterschiedliche Ziele verfolgen.

3.2.1.1 Portfolio-Technik

Basis aller Portfolio-Techniken ist der SWOT-Ansatz. Hierbei werden die für ein Erfolgsobjekt (z.b. SGE) maßgebenden Faktoren hinsichtlich Stärken und Schwächen, Chancen und Risiken auf 2 Faktoren reduziert. Einer repräsentiert den Schlüsselfaktor für die Umweltkomponente, der andere jenen für die Unternehmungskomponente.

Grundsätzliches Ziel der Portfolio-Techniken ist die Visualisierung folgender Themen:

- Finanzielle Mittelverteilung auf die SGEs
- Beurteilung des finanziellen Gleichgewichts bez. bindender und frei werdender Mittel
- Gründung und Schließung neuer SGEs

Die theoretische Basis für die Rationalität der Diversifikation des Kapitaleinsatzes in unterschiedliche Projekte liefert das Portfolio-Selection-Modell von Markowitz [23]). Hiernach sind die erwartete Rendite und das Risiko maßgeblich für die Portfolio-Konstruktion. Die Bildung von Portfolios ist wegen der Risiko-Minimierung sinnvoll, wobei die Höhe des Risikos von der Korrelation der Renditen der einzelnen Wertpapiere (Projekte) bestimmt wird. In der Wertpapier-Portfolio-Theorie findet dieser Gedanke z.b. eine Repräsentation im Capital Asset Pricing Model (CAPM) [24]).

---

[23] Steiner, Manfred, Bruns, Christoph, Wertpapiermanagement, Stuttgart, 2000
[24] Levy, Haim, Introduction to Investments, Cincinnati, Ohio, 1999

3.2.1.1.1 Platzierung der Erfolgsobjekte (Projekte) innerhalb der Matrix

Voraussetzung für die Aussagekraft der Platzierungen der Erfolgsobjekte innerhalb der Matrix ist eine Präferenzfunktion, die mittels eines typologischen Rasters die Matrix aufbaut. Jedem Matrix-Element werden hierbei Normstrategien zugeordnet, die Aussagen über die Ressourcenverteilung (z.B. Neuinvestition, Desinvestition, usw.) oder über die finanziellen Mittel (z.b. Mittelüberschüsse, Mitteldefizite, usw.) zulassen.

Die Platzierung der Erfolgsobjekte erfolgt in der Matrix nach dem Kriterium der größtmöglichen Ausnutzung des Erfolgspotentials. Um die Unternehmung in einem stabilen Gleichgewicht zu halten, ist zwischen risikoreichen und risikoarmen Erfolgsobjekten ein Gleichgewichtszustand zu bewahren bzw. herbeizuführen. Des weiteren muss ein Gleichgewichtszustand bezüglich der Finanzflüsse geschaffen werden (Überschuss / Defizit - Ausgleich).

Die weite Verbreitung erhielt die Portfolio-Technik durch die großen Consulting-Unternehmen. Die Bosten Consulting Group entwickelte Ende der 60er Jahre eines der ersten Konzepte, welches der Gruppe der

- absatzmarkt-orientierten Konzepte

zuzurechnen ist. Diese wurden später durch

- ressourcen-orientierte Konzepte und
- wert-orientierte Konzepte

ergänzt.

3.2.1.1.2 Boston Consulting Group - Portfolio

Dieses absatzmarkt-orientierte Konzept hat den relativen Marktanteil (im Vergleich zum stärksten Konkurrenten) als Schlüsselfaktor für die Unternehmungsdimension, und das Marktwachstum als Schlüsselfaktor für die Umweltdimension zu Grunde liegen.

Die Bedeutung des **relativen Marktanteils** ist eng mit dem Modell der Erfahrungskurve verknüpft. Darin wird die Kostensenkung mit der kumulierten Produktionsmenge in Verbindung gebracht [25]). Im Gegensatz zur Lernkurve erstreckt sich die Erfahrungskurve auf alle Wertschöpfungskosten (Forschungs-, Entwicklungs-, Fertigungs-, Distributions-, Vertriebs-, Marketing-, Kapital-, Gemeinkosten, u.a.). Als Ursachen sind im wesentlichen zu nennen:

- Fixkostendegression
- Größendegression (Economies of Scale; z.b. in der Beschaffung und im Absatz)
- Rationalisierung (z.b. Prozess-Optimierungen)

[25] Henderson, Bruce D., Oetinger, Bolko von, (hrsg.), Das Boston Consulting Group Strategie-Buch, Düsseldorf, 2000

- Technologischer Fortschritt (z.b. effizientere Faktorkombination, Produktmodifikation durch billigere Ausgangsmaterialien)

Die Bedeutung des **Marktwachstums** basiert auf dem Modell des Produktlebenszyklus, das auch für Projekte seine Gültigkeit besitzt.

Basierend auf diesen 2 Schlüsselfaktoren (Marktwachstum und relativer Marktanteil) wird nun eine Matrix - bestehend aus 4 Feldern – angelegt. Jedem dieser Felder ist eine Normstrategie zugeordnet. Dementsprechend tragen sie auch bezeichnende Namen:

- **Cash Cow (niedriges Wachstum, hoher Marktanteil):**

  Bruce D. Henderson: „Cash-Kühe finanzieren ihr eigenes Wachstum. Sie bezahlen die Dividende und die Gemeinkosten der Zentrale. Sie decken die Zinsen des Unternehmens. Sie stellen die Gelder für Forschung und Entwicklung bereit. Sie liefern die Investitionsmittel für andere Produkte. Sie bestimmen die Verschuldungskapazität des gesamten Unternehmens. Deshalb müssen sie geschützt werden." Prinzipiell erlauben Cash-Kühe zwei alternative Strategien:

  - Investition, d.h. Kauf von Marktanteilen (z.b. durch Preisreduktion der Produkte)
  - Desinvestition, d.h. Verkauf von Marktanteilen (z.b. durch Preisdifferenzierung)

  Welche der beiden Strategien der Vorzug gegeben wird, hängt von den alternativen Investitionsmöglichkeiten im Unternehmensportfolio ab.

- **Star (hohes Wachstum, hoher Marktanteil):**

  In diese Kategorie zählen all jene SGEs, die während des Marktwachstums eine führende Marktposition errungen haben. In finanzieller Hinsicht haben sie einen ausgeglichenen Cash Flow. D.h., dass sie gerade so viel Mittel freisetzen, als zur Erhaltung bzw. leichtem Ausbau des Marktanteils notwendig ist. Damit haben die Stars die besten Voraussetzungen, die Cash-Kühe der Zukunft zu werden.

  Bruce D. Henderson: „Wer einen Star angreifen will, muss große Summen einsetzen. Ein Marktanteilsverhältnis von 2:1 bewirkt einen 5 – 8%igen Kostenvorteil in der Wertschöpfung." Läuft der Star dagegen Gefahr bei ausgeglichenem Cash Flow seine Marktposition zu verlieren, gefährdet er die finanzielle Zukunft der SGE. Dies insbesondere dadurch, da sein Beitrag für die SGE ausschließlich in der Zukunft liegt, und durch das hohe Marktwachstum hohe Cash Flows (in beiden Richtungen) stattfinden.

- **Question Mark (hohes Wachstum, niedriger Marktanteil):**

  Hierzu zählen alle Nachwuchsprodukte. Bruce D. Henderson: „In schnell wachsenden Märkten wird jedes Geschäft des Portfolios als „?" bezeichnet, das einem oder mehreren Konkurrenten unterlegen ist. Hohe Investitionen sind bereits erforderlich, damit das Geschäft mit dem Markt wachsen kann. Sie fließen nur dann zurück, wenn eine führende Marktposition erreicht ist, ehe das Wachstum nachlässt. Zu diesem Zweck muss das Unternehmen noch stärker investieren, um den Marktführer einzuholen, oder es muss sich endgültig aus diesem Markt zurückziehen. Strategien zum Erwerb von Marktanteilen sind immer äußerst risikoreich, können aber manchmal sogar ohne bedeutenden Mittelaufwand erfolgreich sein. Strategien zur allmählichen Liquidierung des Geschäfts bergen ein weitaus geringeres Risiko und lassen oft das eingesetzte Kapital zurückfließen. Am ungünstigsten ist es, die ursprüngliche Wettbewerbsposition nicht zu verändern. Dort werden fortgesetzt Mittel gebunden, die niemals zurückfließen werden."

- **Dogs (niedriges Wachstum, niedriger Marktanteil):**

  Zu den Dogs werden jene SGEs gezählt, die sich bereits in der Sättigungsphase oder gar in der Degenerationsphase befinden, und die es bis dahin nicht geschafft haben, eine Führungsrolle am Markt einzunehmen. Obwohl grundsätzlich wieder zwei Alternativen (Investition, Desinvestition) zur Auswahl stehen, empfiehlt sich in erster Linie die Desinvestitions-Strategie. Da sich in der Praxis die negativen Cash Flows sich meist nur langsam entwickeln, wodurch die Desinvestition erst relativ spät einsetzt, spricht man in diesem Zusammenhang auch oft von Cash-Fallen.

Obwohl dieses Konzept als Meilenstein der Strategie-Formulierung gilt, sind auch einige Kritikpunkte nicht zu übersehen. Viele Märkte sind mittlerweile gesättigt, wodurch starkes Marktwachstum seltener wird. Ferner wird die Bedeutung des relativen Marktanteils generell durch die Existenz vieler kleiner Unternehmungen relativiert. Existiert für Forschungs- und Entwicklungsaktivitäten noch kein externer Markt, können sie in der Matrix nicht berücksichtigt werden. Oft wird auch das stark vereinfachte Modell kritisiert, das die Komplexität realer Unternehmen nur unzureichend darzustellen vermag.

3.2.2 Unternehmenswert-orientierte Konzepte

In den Diskussionen über die Struktur der Zielformulierung erlang in den 80er Jahren ein neuer Begriff Berühmtheit – jener des **„Shareholder Values"**. Dieser Wert fokussiert auf den Marktwert des Eigenkapitals einer Unternehmung. Untergliedert man diese in mehrere Bereiche, muss der Unternehmenswert größer als die Summe der Wertbeiträge der einzelnen Bereiche sein. Sind die Chancen hierzu eher gering, empfiehlt sich eine Auflösung der Unternehmung.

Begleitet wird dieses Konzept des Shareholder Values mit dem Begriff des Wertmanagements und der Wertschaffung. Unter Wertschaffung wird die im Alternativen-Vergleich (z.b. gegenüber dem Gesamtmarkt) erzielte Steigerung des Wertes im Periodenvergleich verstanden [26]). Im Falle börsennotierter Unternehmen würde dies bedeuten, dass durch erfolgreiches Wertmanagement der Wertzuwachs des Unternehmens (Kurs * Aktien-Anzahl + Dividenden-Zahlungen) das Marktwachstum (z.b. Aktienindex, welcher auch Dividendenzahlungen berücksichtigt) übertrifft. Eine Studie der Boston Consulting Group über die erfolgreichen Wertschaffer ergab eine Korrelation mit folgenden Größen [27]):

- Renditeveränderung
  - Steigerung der Cash Flow Marge (z.b. Preisdifferenzierung und Kostensenkung)
  - Erhöhung der Kapitalproduktivität
- Wachstum

Bevor der finanzielle Aspekt näher erläutert wird, soll noch einiges zu den Auswirkungen dieses Gedankens auf die Mitarbeiter angeführt werden. Durch das Wertmanagement sind grundsätzlich alle Hierarchien in einem Unternehmen betroffen. Von besonderer Bedeutung ist in diesem Zusammenhang das Vergütungssystem [28]). Ein ausgewogenes Vergütungssystem weist demnach 3 Elemente auf.

Das Grundgehalt dient als Basisabsicherung zur Erhaltung eines angemessenen Lebensstandards (inkl. Pensionsvorsorge). Darüber hinaus sollen kurzfristige Incentives (z.b. Jahresbonus) in Übereinstimmung mit der Gesamtstrategie die Wertschaffung auf operativer Ebene steuern. Als Beispiel wären Qualitäts- und Prozess-Kennzahlen zu nennen. Langfristige Incentives (z.b. Aktienoptionen) sollen der Wertschaffung eine mehrjährige Ausrichtung geben (z.b. Ausgestaltung der Optionen mit einer mehrjährigen Sperrfrist).

---

[26] Lewis, T.G., Steigerung des Unternehmenswerts, Landsberg, 1995
[27] Nölting, A., Hebel-Wirkung, in: manager magazin, Nr.5/1998
[28] Roos, Alexander, Stelter Daniek, Oetinger, v. Bolko (hrsg.), Das Boston Consulting Strategie-Buch, Düsseldorf, 2000

Die in diesem Zusammenhang diskutierten Ansätze sind [29]):

- DCF-Verfahren (Discounted Cash Flow)
- Entity-Methoden (auf dem Unternehmens-Gesamtwert basierend)
  - WACC-Methode (Weighted Average Cost of Capital)
  - APV-Methode (Adjusted Present Value)
- Equity-Methode (auf dem Eigenkapital-Wert basierend)
- EVA-Konzept (Economic Value Added)
- CFROI-Konzept (Cash Flow Return on Investment)

3.2.2.1 Stern Stewart & Co's – EVA-Konzept

Dieses von der Beratungsgesellschaft Stern Stewart vermarktete Konzept der Übergewinn-Ermittlung dient unternehmensintern zur Performance-Messung des Managements und unternehmensextern zur Beurteilung der Wertsteigerung innerhalb einer Periode (EVA = Economic Value Added). Der große Vorteil dieses Beurteilungs-Konzeptes ist die einfache Anwendbarkeit auf Produktebene. Hierdurch eignet es sich in optimaler Weise als Kriterium für die Aufnahme eines Produkts ins Portfolio. Dieses Verfahren wird u.a. auch bei Siemens zur Beurteilung von Software-Produkten eingesetzt.

Als Basis zur Berechnung der EVAs der einzelnen Produkte dient der NOPAT (= Net Operating Profit after Taxes oder Nach-Steuer-Betriebsgewinn). Der NOPAT seinerseits entspricht

- dem FCF (= Free Cash Flow) zuzüglich den Nettoinvestitionen, bzw.
- dem EBIT (= Earnings before Interest and Taxes oder Betriebsergebnis vor Zinsen und Steuern) abzüglich einer fiktiven Steuerbelastung bei vollständiger Eigenfinanzierung

3.2.2.1.1 Bildung der Kenngrößen EBIT und FCF

Alle im weiteren angeführten Bezeichnungen beziehen sich stets auf die selbe Periode. Abweichungen werden separat durch Indizierung angezeigt.

[29] Steiner, Manfred, Bruns, Christoph, Wertpapiermanagement, Stuttgart, 2000

EBIT (= Earnings before Interest and Taxes; basierend auf einer Gliederung der Gewinn- und Verlustrechnung (G&V), ohne Berücksichtigung „sonstiger" Posten [30])):

+	Umsatzerlöse
+/-	Bestandsveränderung
+	aktivierte Eigenleistungen
	= Betriebsleistung
-	Materialaufwand und Aufwendungen für bezogene Leistungen
	= Deckungsbeitrag
-	Personalaufwand
-	Abschreibungen auf Sachanlagen
=	**EBIT oder Betriebserfolg oder Betriebsergebnis**
+	Finanzerfolg
=	**EGT (= Ergebnis der gewöhnlichen Geschäftstätigkeit)**

Im EBIT sind somit neben den Herstellungskosten für die verkauften Produkte auch die Verwaltungs- und Vertriebsaufwendungen abgezogen. Des weiteren sind auch die Abschreibungen auf das Anlagevermögen bereits abgezogen. Wird das EBIT einer Periode um die darauf entfallende Gewinnsteuer vermindert, erhält man den NOPAT (auch als NOPLAT (= Net Operating Profit Less Adjusted Taxes)) der Periode.

Bemerkung: In der Break-Even-Analyse wird das EGT unter dem Gesichtspunkt von variablen und fixen Kosten folgendermaßen dargestellt:

+	**Betriebsleistung**
-	variable Kosten
=	**Deckungsbeitrag**
-	Fixkosten
=	**EGT (bzw. allgemein der Gewinn)**

FCF (= Free Cash Flow) [31]):

+	EBIT
-	adaptierte Steuerzahlungen
	= NOPAT oder auch NOPLAT
+	Abschreibungen (1)
+	Erhöhung der langfristigen Rückstellungen (2)
-	Investitionen in das Anlagevermögen (3)
-	Erhöhung des Working Capital (4)
=	**FCF**

[30] Kralicek, Peter, Bilanzen lesen, Wien, 1999

[31] Copeland, T.E., Koller, T., Murrin, J., Valuation: Measuring and Managing the Value of Companies, New York, 1994

wobei (3) + (4) – (1) – (2) unter dem Begriff **Nettoinvestitionen** (abgekürzt als **NI**) zusammengefasst werden. Somit ergibt sich für den FCF die alternative Formulierung:

+	**NOPAT**
-	**Nettoinvestitionen**
=	**FCF**

3.2.2.1.2 Ermittlung der Perioden-EVA

Für die Ermittlung der Perioden-EVA ist von der Ergebnisgröße NOPAT das zu Beginn der Periode gebundene Kapital (KB_{t-1}) in Abzug zu bringen. Als gebundenes Kapital werden jene Finanzierungsmittel verstanden, die zur Erzielung des Betriebsergebnisses benötigt werden. Somit kann die Kapitalbindung jeder Periode t >= 1 als die Kapitalbindung zum Bewertungszeitpunkt t = 0 zuzüglich der Summe der Nettoinvestitionen bis zum Zeitpunkt t formuliert werden.

$KB_t = KB_0 + \text{Summe } (NI_a) \text{ für } _{a = 1 \text{ bis } t}$

Da die NOPAT eine fiktive Ergebnisgröße bei vollständiger Eigenfinanzierung darstellt, ist der Steuereffekt der Fremdfinanzierung bei der Ermittlung der Kapitalkosten zu berücksichtigen. Der Steuervorteil der Fremdkapitalaufnahme wird durch die Bildung eines durchschnittlichen Kapitalkostensatzes berücksichtigt. Dieser auch als WACC (= Weighted Average Cost of Capital) bezeichnete Kostensatz repräsentiert die mittlere Renditeforderung [32]):

$WACC = (EK/GK)*k_{ek} + (FK/GK)*i*(1-s_u)$

mit:	EK	Eigenkapital zu Marktwerten (= Shareholder Value)
	FK	Fremdkapital zu Marktwerten
	GK	Gesamtkapital zu Marktwerten
	k_{ek}	Eigenkapitalkostensatz
		(Ermittlung über CAPM-Modell; siehe unten)
	i	risikoloser Fremdkapitalzinssatz bzw. risikolose Rendite
	s_u	Gewinnsteuersatz

Unter Verwendung des CAPM [33]) gilt für k_{ek} folgender Zusammenhang:

$k_{ek} = i + (E(R_m) - i) * \text{Beta}_{ek}$

mit:	i	risikoloser Fremdkapitalzinssatz bzw. risikolose Rendite
	$E(R_m)$	Erwartungswert der Rendite des Marktportfolios
	Beta_{ek}	Risiko

[32] Mandl, Gerwald, Rabel, Klaus, Unternehmensbewertung, Wien, 1997

[33] Sharpe, W.F., Capital Asset Prices: A Theory of Equilibrium under Conditions of Risk, in: Journal of Finance, Vol. 19, 1964

Das Risiko ist hierbei definiert als der Quotient aus der Kovarianz zwischen dem Wertpapier das das Eigenkapital verbrieft und dem Marktportfolio und der Varianz des Marktportfolios.

Damit lässt sich der EVA der Periode t darstellen als:

$EVA_t = NOPAT_t - (1 + WACC) * KB_{t-1}$ (Capital Charge Form)

Diese Formel ist in der Literatur unter dem Namen „Capital Charge Formel" [34]) bekannt. In dieser Schreibweise kann der EVA als Residualgewinn interpretiert werden.

In der „Value Spread Form" wird der EVA als Überrendite auf das investierte Kapital dargestellt.

$\mathbf{EVA_t = NOPAT_t - WACC * KB_{t-1} = (r_t - WACC) * KB_{t-1}}$ **(Value Spread Form)**

mit: $\mathbf{r_t = NOPAT_t / KB_{t-1}}$ (Investitionsrendite)

Forschungs- und Entwicklungsaufwände sollten nach Stewart [35]) in das gebundene Kapital (KB) miteinbezogen und über die Nutzungsdauer abgeschrieben werden.

Mit der oben angeführten Formel für den EVA kann der aktuelle Unternehmenswert aus der aktuellen Kapitalbindung und den Barwerten der zukünftigen EVAs ermittelt werden.

$U_0 = KB_0 + Summe \{ [E(NOPAT_t) - WACC * KB_{t-1}] / (1 + WACC)^t \}$ für $_{t = 1 bis unendlich} =$
$= KB_0 + Summe \{ EVA_t/(1+WACC)^t \}$ für $_{t = 1 bis T} + 1/(1+WACC)^T * EVA_{T+1}/WACC$

So dargestellt lässt sich der Unternehmenswert in das gebundene Kapital und einen derivativen Firmenwert zerlegen. Dieser derivative Firmenwert wird auch als MVA (= Market Value Added) oder Goodwill bezeichnet, d.h.

$MVA_0 = EK + FK - KB_0 = Summe \{ EVA_t/(1+WACC)^t \}$ für $_{t = 1 bis unendlich}$

Damit ist der Grenzwert des zu erzielenden MVA mit 0 festgelegt, bei der das Portfolio keinen zusätzlichen Nutzen gegenüber dem Durchschnittsmarkt erwirtschaftet.

Als Stellgrößen zur Verbesserung der Überrendite können unter Bezugnahme auf die Value Spread Form $EVA_t = (r_t - WACC) * KB_{t-1}$ folgende Maßnahmen ergriffen werden:

r_t:	Produktivitätssteigerung
WACC:	Optimierung der Kapitalkosten
KB_{t-1}:	Optimierung des Portfolios hinsichtlich der Verteilung des Kapitals

[34] Hostettler, S., Das Konzept des Economic Value Added (EVA), Bern, 1997
[35] Stewart, B., Announcing the Stern Stewart Performance 1000: A New Way of Viewing Corporate America, Journal of Applied Corporate Finance, Vol. 3, 1990

Aufgrund eigener Erfahrungen liegt der größte Nachteil dieser Methode in der schwierigen Abschätzbarkeit des zukünftigen Erfolgs eines Software-Produkts (z.B. in der Abschätzung zukünftiger Umsatzzahlen). Um das Risiko diesbezüglich beziffern zu können, empfiehlt sich eine Sensitivitätsanalyse. Hierbei werden der Abschätzung unterschiedliche Szenarien zugrunde gelegt (z.b. variierende Technologie-Trends) die ihrerseits auf unterschiedliche EVA-Werte führen.

3.2.3 Die Balanced Scorecard

3.2.3.1 Strategie-Implementierung

Die im Rahmen der Strategieplanungen erarbeiteten Konzepte müssen anschließend in operative Maßnahmen, Termin- und Budgetplanungen umgesetzt werden. Hierbei werden in einem ersten Schritt die Strategieformulierungen zu Projekten verdichtet. Im Rahmen der Programmplanung werden die Projekte aufeinander abgestimmt.

In einer langfristigen operativen Planung werden die für die Realisierung der Programme notwendigen Ressourcen – in erster Linie Budgets - Funktionsbereichen zugeordnet. Das eigentliche operative Steuerungsinstrument – die kurzfristige operative Planung – detailliert die Budgetanforderungen und stellt sie in Relation zu Terminplänen.In einer Darstellung von R. Huber [36]) gliedert sich die Programmplanung in folgende Schritte:

1. Festlegung mittelfristiger Ziele
   Ein Hilfsmittel zur Unterstützung des mittelfristigen Zielfindungsprozesses ist die Balanced Scorecard.
2. Entwicklung von Maßnahmen
   Diese sind Schritte auf dem Weg zur stufenweisen Konkretisierung der Strategie.
3. Ermittlung der Reihenfolge und Interdependenzen
   Die Programmplanung macht eine Abstimmung der Maßnahmen erforderlich (z.b. durch die begrenzten Ressourcen).
4. Festlegung des Zeitbedarfs
   Als Hilfsmittel für die Terminplanungen dient die Netzplantechnik.

[36] Huber, R., Überwindung der strategischen Diskrepanz und Operationalisierung der entwickelten Strategie, St. Gallen, 1985

5. Festlegung der Verantwortlichkeiten und Zuständigkeiten
   In diesem Schritt erfolgt die Organisationsplanung, die in enger Beziehung mit dem Anreizsystem steht.

6. Ableitung des Ressourcenbedarfs und der Ressourcenallokation
   In diesem Schritt wird der Grundstein des Projektes gelegt. Darin committet sich das Management zur Verteilung der personellen, sachlichen und finanziellen Ressourcen.

3.2.3.2 Strategie-Visualisierung

In ähnlicher Weise, wie in den vergangenen Jahren und Jahrzehnten die Portfolio-Konzepte im Top-Management Einzug hielten, verbreitete sich der Einsatz der Balanced Scorecard (BSC) im mittleren Management. Das Konzept wurde von R.S. Kaplan und D.P. Norton zusammen mit den Beratungsunternehmen Gemini Consulting und KPMG ausgearbeitet. Motivation war der Mangel an konzeptioneller Verknüpfung zwischen strategischer und operativer Ebene. Insbesondere soll die BSC die primär finanzwirtschaftlich ausgerichteten Controlling-Konzepte um weitere Inhalte ergänzen. Diese Inhalte repräsentieren das Geschäftsgeschehen auf operativer Ebene.

Die BSC ist somit ein formeller „Ansatz zur Umsetzung einer Strategie in spezifische Ziele und Kennzahlen und zur Überwachung der Umsetzung in den folgenden Perioden." [37]) Wie schon durch den Namen unterstrichen, schafft die BSC eine Balance zwischen extern orientierten Messgrößen (z.b. finanzwirtschaftliche Kennzahlen, u.a.) und intern orientierten Messgrößen (z.b. für Geschäftsprozesse, Qualität, Wissenszuwachs, u.a.).

Der Mehrwert der BSC gegenüber einer allgemeinen Diskussion (zur Detaillierung und Konkretisierung einer formulierten Strategie) liegt in der Vorwegnahme von 4 (mit Prioritäten belegten) Erfolgsfaktoren:

1. **Finanzen**: „Wie sollen wir gegenüber Teilhabern auftreten, um finanziellen Erfolg zu haben?"
2. **Kunden**: „Wie sollen wir gegenüber unseren Kunden auftreten, um unsere Vision zu verwirklichen?"
3. **Interne Geschäftsprozesse**: „In welchen Geschäftsprozessen müssen wir die besten sein, um unsere Teilhaber und Kunden zu befriedigen?"

[37] Kaplan, R.S., Norton, D.P., Balanced Scorecard: Strategien erfolgreich umsetzen, Stuttgart, 1997

4. **Lernen und Entwicklung**: „Wie können wir unsere Veränderungs- und Wachstumspotentiale fördern, um unsere Vision zu verwirklichen?"

Im Zuge der Aufstellung der BSC sind für jede dieser 4 Perspektiven

- Ziele
- Kennzahlen
- Vorgaben
- Maßnahmen

zu finden. Hierbei muss berücksichtigt werden, dass die 4 Perspektiven in einer Ursache- und Wirkungsbeziehung stehen. Als Ausgangspunkt der Dekomposition fungieren die aus der Strategieformulierung abgeleiteten finanziellen Ziele.

Als Kritikpunkte wären die Fokussierung auf Kennzahlen, oder die fehlende konzeptuelle Begründung der 4 Perspektiven zu nennen. Als Stärke der BSC ist die Transparentmachung der Unternehmensstrategie bis auf die operative Ebene zu nennen. Damit ist ein durchgängiges Konzept geschaffen, das die Kausalkette von Strategien auf der obersten Unternehmensebene und Arbeitsanweisungen auf der untersten Ebene verknüpft.

3.2.3.2.1 Die finanzwirtschaftliche Perspektive

Zunächst muss jede Kennzahl in einer Zweck-Mittel-Relation zur übergeordneten Strategie stehen. Des weiteren wird für die qualitative Beurteilung der Kennzahlen auf den Lebenszyklus Rücksicht genommen. Dieser wird in die 3 Entwicklungsstadien

- Wachstum (z.b. Umsatzwachstum),
- Reife (z.b. Deckungsbeitrag) und
- Ernte (z.b. Kapitalrentabilität)

untergliedert. In Klammern sind Beispiele für die jeweilige Phase bezüglich Ertragswachstum angeführt. Darüber hinausgehend führen R.S. Kaplan und D.P. Norton auch Kostensenkungs- und Investitionsziele an.

3.2.3.2.2 Die kundenorientierte Perspektive

Darunter werden die klassischen Aufgaben des Marketing mit den entsprechenden Kennzahlen verstanden. Mit den Worten von M. Bruhn [38]) handelt diese Perspektive von der Erfüllung der Erfolgskette: „Kundenorientierung -> Kundennutzen -> Kundenzufriedenheit -> Kundenbindung -> ökonomischer Erfolg". R.S. Kaplan und D.P. Norton ergänzen diese Kernkennzahlen um sogenannte „Differenziatoren", wie z.B. Produkteigenschaften, Qualität, Beratung, u.a.

3.2.3.2.3 Die interne Prozessperspektive

Die Konzentration liegt hierbei bei den für die Stakeholder kritischen Prozessen. Hinzu kommt die Identifikation neuer, wertschöpfender Prozesse. R.S. Kaplan und D.P. Norton nennen als kritische Prozesse mit ihren zugehörigen Kennzahlen:

- Innovationsprozesse (z.b. Neuproduktrate, Entwicklungszeit)
- Betriebsprozesse (z.b. Durchlaufzeit, Fehlerrate)
- Serviceprozesse (z.b. Reklamationslaufzeit)

3.2.3.2.4 Die Lern- und Entwicklungsperspektive

Ziel dieser Perspektive ist eine lernende und wachsende Organisation, die die Voraussetzung bzw. Infrastruktur für alle anderen Perspektiven ist. Im Detail sprechen R.S. Kaplan und D.P. Norton folgende Kategorien an:

- Mitarbeiterpotentiale
- Potentiale der Informationstechnologie
- Arbeitsklima und Kultur

Diese 3 Kategorien haben ihrerseits Auswirkungen auf folgende Kernkennzahlen:

- Mitarbeiterzufriedenheit
- Personaltreue
- Mitarbeiterproduktivität

[38] Bruhn, M. (hrsg.), Balanced Scorecard: Wertorientierte Unternehmensführung, Wiesbaden, 1998

4. Projekt-Management-Leitfaden

Jede Software-Organisation, die über mehr als 2 bis 3 Mitarbeiter verfügt, sollte ihre Aufbau- und Ablauforganisation in schriftlicher Form festlegen. Der Umfang des Dokuments, das diese Organisation beschreibt, ist in vielen Fällen proportional der Personalstärke der Software-Entwicklungs-Organisation. Im Falle eines mehrere hundert Mitarbeiter umfassenden Software-Hauses können diese Dokumente leicht unüberschaubare Größen erreichen. Daher empfiehlt sich im praktischen Einsatz eine auf die notwendigste Information konzentrierte Beschreibung des die Entwicklung begleitenden organisatorischen Rahmens. Hierbei wird vorausgesetzt, dass das Software-Haus im Rahmen eines Projekt-Programm-Managements mehrere Projekte gleichzeitig verfolgt, welche von größtenteils verschiedenen Projekt-Teams realisiert werden.

Ein derartiger Leitfaden wird in den folgenden Abschnitten dieses Kapitels dargestellt. Er wurde zum überwiegenden Teil vom Dissertanten im Rahmen einer Neustrukturierung einer Entwicklungs-Abteilung innerhalb eines Software-Hauses erstellt. Hinsichtlich des Inhalts wurden Namen anonymisiert, der Stil des Leitfadens jedoch weitgehend unverändert beibehalten.

4.1 Konzept

Der Leitfaden dient als Richtlinie für alle Projektleiter innerhalb der angesprochenen Entwicklungs-Abteilung (untergliedert in Dienststellen) und ist als Ergänzung bzw. Präzisierung der eingesetzten Entwicklungsmethode zu sehen. In der vorliegenden Richtlinie werden die 4 wesentlichen Abschnitte eines Projekts aus Sicht des Projektmanagements behandelt:

- Projektinitiierung
- Projektplanung
- Projektkontrolle
- Projektabschluss

Als Standardentwicklungsmethode wird jene des Software-Hauses zugrunde gelegt. Dies gilt nicht für den Fall, dass vom Auftraggeber eine andere Methode vorgegeben wird. Zur Statuserhebung der Projekte wird ein webbasiertes Projekt-Controlling-Tool verwendet.

4.2 Projektinitiierung

Die Phase Initiierung muss bei jedem Projekt laut Entwicklungsmethode durchlaufen werden. Die vorgeschriebenen Dokumente (diese sind im wesentlichen: Grobprojektplan, Grob-Qualitäts-Sicherungs-Plan, Projektentscheid) sind zu erstellen. Bei positivem Projektentscheid wird ein Angebot ausgearbeitet.

4.2.1 Angebote

Für jedes Projekt ist ein verbindliches Angebot zu legen. Dazu sind die von der Kaufmannschaft zur Verfügung gestellten Musterangebote zu verwenden. Es muss bereits in den Angeboten auf ein im Projekt notwendiges Change-Request-Verfahren hingewiesen werden.

Jedes Angebot bekommt eine eindeutige Nummer mit definiertem Aufbau. Die Verwaltung der Angebotsnummern erfolgt in einer Angebotsverwaltungsdatei. Dort wird vom Ersteller des Angebots die nächste freie Nummer abgeholt und die wesentlichen Angebotsdaten laut Tabelle eingetragen. Weiters wird für jedes erstellte Angebot im obigen Verzeichnis ein Unterverzeichnis mit einem eindeutigen Namen angelegt, unter dem alle für das Angebot relevanten Dokumente (z.b. Angebotstext, Anlagen, Kalkulationsgrundlage, Reviewprotokoll) abgelegt werden. Jedes Angebot wird mit einem Mustertext verglichen.

4.2.2 Kick-Off Meeting

Als erste Aktivität nach dem Projektstart (= Auftrag oder Abteilungsbeschluss) ist ein Kick-Off Meeting durchzuführen, an dem alle zu diesem Zeitpunkt benannten Projektmitarbeiter teilnehmen sollten.

Zur Vorbereitung wird die Checkliste aus der Entwicklungsmethode herangezogen. Nach Möglichkeit hat der Projektleiter schon vor Beginn des Kick-Off Meetings die Projektziele mit den Teilprojektleitern vereinbart.

Ziel des Kick-Off-Meetings ist die Entwicklung einer gemeinsamen Projektsicht. Die Tagesordnung hiezu sollte mindestens folgende Punkte umfassen:

- Projektziele
- Projektdefinition
- Projektorganisation

- Projektmanagement
- Strukturierung
- Richtlinien
- Projektstatus
- Ressourcen
- Termine
- Kosten
- Kommunikation
- Configuration Management
- Qualitätssicherung
- Risken

4.3 Projektplanung

Für jedes Projekt wird vom Projektleiter ein Projektplan nach der Entwicklungsmethode erstellt. Wird das Projekt nicht nach dieser abgewickelt und ist eine andere Form des Projektplanes vorgegeben, so ist mindestens die Checkliste aus diesem zu verwenden. Es sind alle dort aufgeführten Punkte zu berücksichtigen. Der folgende Abschnitt enthält Regelungen für folgende Aufgabenbereiche:

- Strukturplanung
- Aufwandsschätzung (Function Points)
- Terminplanung (Tool: MS-Project)
- Ressourcenplanung (Tool: Personaleinsatzplanungstool der Abteilung)
- Kostenplanung
- Berichtswesen

4.3.1 Strukturplanung

4.3.1.1 Projekt-/Produktstruktur

Ein großes Projekt (mehr als 10 Mitarbeiter) wird in Teilprojekte (typisch 7 bis max. 10 Mitarbeiter) mit klarer Verantwortung (auch für Kosten und Termine) zerlegt. Der Blick auf das Gesamtprojekt darf dennoch nicht verloren gehen (Einhaltung der im Kick-Off Meeting festgelegten Strategie!). Im Regelfall setzt sich ein Projekt aus 1 bis n Realisierungsprojekten sowie den Teilprojekten: Integrationstest, Systemtest und Dokumentation zusammen.

Teilprojekte werden in Tätigkeiten (=Arbeitspakete) zerlegt, die aus der Komponentenplanung abgeleitet werden. Ein Arbeitspaket ist die kleinste plan- und verfolgbare Einheit und sollte zur besseren Kontrolle des Projektfortschritts innerhalb eines Monats abgeschlossen werden können und einen Aufwand von 3 MannMonaten nicht überschreiten. Zu diesen kleinen Planungseinheiten kommt man durch schrittweise Verfeinerung der Planung. Spätestens beim Beginn der Arbeiten an einer Komponente muss die Aufsplittung in Arbeitspakete erfolgt sein. Wichtig ist, dass so ein Arbeitspaket mit überprüfbaren Ergebnissen abgeschlossen wird.

4.3.1.2 Kontenstruktur

Aufgrund der Zerlegung des Projekts in Teilprojekte und Arbeitspakete ergibt sich der Kontierungsplan, d.h. den Teilprojekten und Arbeitspaketen werden Auftragsnummern und Teilauftragsnummern zugeordnet.

Grundsätzlich kann die Projektstruktur nahezu beliebig viele Ebenen enthalten. Wir beschränken uns zur Zeit auf 3 Ebenen (Projekt -> Teilprojekt -> Arbeitspaket). Für jedes Projekt sind entsprechend der Projektstrukturierung Konten einzurichten, wobei folgender Prozess einzuhalten ist.

1. Kontonummer ermitteln
2. Auftragsblatt (elektronisch) ausfüllen
3. Auftragsblatt ausdrucken
4. Auftragsblatt unterschreiben (techn.)
5. Bei Vorleistungskonten: Genehmigung
6. Auftragsblatt unterschreiben (kaufm.)und verteilen

Bei Änderungen der Auftragsdaten (Aufwand, Termin, usw.) wird vom zuständigen Dienststellenleiter das Originalauftragsblatt aktualisiert und paraphiert. Wenn durch oftmaliges Ändern die Übersichtlichkeit verloren geht, wird das Auftragsblatt neu erstellt.

4.3.1.3 Berichtswesen

Die Verantwortlichkeiten für das Berichtswesen sind festzulegen. Weiters ist zu definieren Wer Wem Was Wann und Wie zu berichten hat.

4.3.1.4 Projektbuch

Es ist ein Projektbuch zu führen, in dem alle vertragsrelevanten Vereinbarungen in konzentrierter Form abzulegen sind. Das beginnt mit Angebot und Auftrag. Weiters sind alle Schriftstücke, in denen Änderungen des Auftrags (Aufwand, Kosten, Termin, usw.) dokumentiert sind, chronologisch abzulegen (auch beauftragte Change-Requests).

4.3.2 Aufwandsschätzung

Neben der Expertenschätzung ist bei jedem neuen Projekt die Function Point Methode anzuwenden. Eine abteilungsspezifische Umrechnungskurve von Function Points zu Aufwand kann erst nach Abschluss von mehreren Projekten statistisch ermittelt werden. Die Function Points werden im Auftragsfall im Projekt-Controlling-Tool hinterlegt.

4.3.3 Terminplanung

Für die Terminplanung wird ein Netzplan (oder Balkenplan (= Gantt-Diagramm)) herangezogen. Dieser wird für das Gesamtprojekt und für jedes Teilprojekt mittels MS-Project erstellt. Teilprojekte werden auf Arbeitspaketebene geplant. Arbeitspakete werden einem Meilenstein zugeordnet. Somit ist die Erklärung eines Meilensteins durch den Abschluss der hiefür notwendigen Arbeitspakete auch qualitativ bestimmt. Definierte Meilensteine werden ins Projekt-Controlling-Tool übernommen.

4.3.4 Ressourcenplanung

Die Personaleinsatzplanung wird mit dem Personaleinsatzplanungstool (PEP) durchgeführt. Die Planung erfolgt durch die jeweiligen Dienststellenleiter in Abstimmung mit dem jeweiligen Projektleiter. Bei Dienststellen-übergreifenden Projekten sind die notwendigen Absprachen zu treffen. Geplant wird auf Projekt- und Teilprojektebene.

Zu Monatsende sind die zukünftigen Daten zu aktualisieren und gegebenenfalls zu ergänzen. Diese Plandaten (auf Monatsbasis) werden ins Projekt-Controlling-Tool übernommen. Der Betrieb des Tools ist den Standardprozessen zuzurechnen. Wichtig ist die Berücksichtigung der Aufwände für Systems-Engineering, welche den Projekten meist vorgelagert und daher oft nicht eindeutig zurechenbar sind.

4.3.5 Kostenplanung

Die Kostenplanung berücksichtigt die Personal-, Hardware-, Leitungs-, Reise-, Lizenz- und sonstigen Kosten.

4.3.5.1 Tooling

Der Einsatz von Tools ist zu planen, wobei Empfehlungen aus anderen Abteilungen zu berücksichtigen sind.

4.3.5.2 Projekt-Controlling-Tool

Die Planung ist erst dann fertig, wenn die entsprechenden Plandaten auch in Projekt-Controlling-Tool erfasst wurden. Die Verantwortung hiefür liegt beim Projektleiter. Wichtig ist, dass jedem definierten Meilenstein alle geplanten Meilensteinergebnisse zugeordnet werden.

4.4 Projektkontrolle

Ziel der Projektkontrolle ist ein möglichst frühzeitiges Erkennen von Planabweichungen, um steuernde Maßnahmen rechtzeitig einzuleiten. In die Kontrolle werden die quantifizierbaren Größen (Termin, Aufwand und Kosten) sowie der Sachfortschritt einbezogen.

Für die Projektfortschrittskontrolle wird die Funktionalität (Soll/Ist-Vergleich) des Projekt-Controlling-Tools genutzt, wo eine Betrachtung auf Arbeitspaketebene möglich ist. Dadurch kontiert der Mitarbeiter direkt auf das Arbeitspaket. Die Zuordnung der Projektstruktur auf die Auftragsstruktur ist Aufgabe des Projektleiters.

Unabhängig vom Tooling müssen für eine wirksame Kontrolle des Entwicklungsfortschritts die Anforderungen der Überprüfbarkeit und der Kurzfristigkeit gegeben sein:

- Überprüfbarkeit

  Aussagen wie "Programm ist zu 80 % fertig und zu 50% getestet" oder "Noch 8 Tage bis zur Fertigstellung" sind nicht überprüfbar, wodurch ein „fertig“ / “nicht fertig“ - System

auf Arbeitspaketebene empfohlen wird. Relativ einfach kann dies beim Integrations- und Systemtest eingeführt werden.

Für jedes Arbeitspaket ist ein Verantwortlicher zu definieren, der mit dem Projektleiter vereinbart, nach welchen Kriterien ein Arbeitspaket als „fertig" gilt.

Für den Komponententest sind projektspezifisch Testende-Kriterien festzulegen.

- Kurzfristigkeit

  Ein Arbeitspaket muss innerhalb eines Monats erledigt werden können. Nur dadurch können Verzögerungen rechtzeitig erkannt werden.

Bei Einhaltung obiger 2 Anforderungen kommt man neben der Termin- und Aufwandskontrolle, die heute schon stattfindet, zu einer wirksamen Sachfortschrittskontrolle. Anstelle des relativen Fertigstellungsgrades, der durch die Beantwortung der Frage "Zu welchem Prozentsatz ist das Arbeitspaket fertig" bestimmt wird, kommt man zur Bestimmung des absoluten Fertigstellungsgrades. Dieser wird durch die digitale Fragestellung "fertig oder nicht fertig" bestimmt.

z.b.

Ein Teilprojekt hat x Arbeitspakete mit geschätztem Gesamtaufwand
von 90 MannMonaten (MM):
n Arbeitspakete zu geplanten 30 MM fertig
m Arbeitspakete zu geplanten 60 MM nicht fertig (hierzu werden auch jene Arbeitspakete gezählt, die „beinahe fertig" sind)

Teilprojekt ist zu 33 % (= 30 / 90) fertig

Diese Information ist vorwiegend bei Großprojekten wertvoll. Ist ein Arbeitspaket fertig, sind vom Projektleiter die entsprechenden Einträge im Projekt-Controlling-Tool zu veranlassen.

4.4.1 Berichtswesen

Aufgabe des Berichtswesens ist eine aktuelle und übersichtliche Information über den Projekt-Fortschritt. Voraussetzung für ein regelmäßiges Berichtswesen ist die Planung der Verantwortlichkeiten zur Berichterstattung zu Beginn des Projekts. Die Hauptgliederung der Berichte erfolgt entsprechend der Projektgliederung in Projekte, Teilprojekte, Phasen (Zeit) und Arbeitspakete.

4.4.1.1 Projektmappe

Für das Projekt-Management-Berichtswesen ist vom Projektleiter eine monatlich aktuelle Projektmappe zu führen, die sich im wesentlichen auf die Daten des Projekt-Controlling-Tools stützt. Diese Projektmappe wird auch zur Information der Abteilungs-Leitung verwendet und dient als Basis für die monatlichen Projektdurchsprachen. Welche Teile davon für das externe Berichtswesen herangezogen werden, ist projektspezifisch festzulegen. Grundsätzlich sollten in die Projektmappe folgende Dokumente aufgenommen werden:

- Roadmap
  Sie soll den zeitlichen Rahmen des Projekts auf einer Seite zusammengefasst abbilden. Dazu werden die Phasen der Projekte bzw. Teilprojekte in Form eines Balkenplans dargestellt. (MS-Project).
- Teilprojektliste
- Terminpläne der Teilprojekte (MS-Project)
- Projektorganisation
- Aktueller Monatsbericht

4.4.1.2 Monatsbericht

Das Projekt-Controlling-Tool liefert die standardisierten Basisdaten für das Berichtswesen. Darüber hinaus sind vom Projektleiter Folien zu erstellen, die in den monatlichen Projektdurchsprachen zu präsentieren sind und folgende Punkte beinhalten:

- Sachfortschrittsbericht (Soll/Ist-Vergleich der fertiggestellten Arbeitspakete)
- Problembericht
- Abweichungsanalyse mit vorgeschlagenen Steuerungsmaßnahmen
- Qualitätsbericht (wird vom Qualitäts-Sicherungs-Verantwortlichen beigesteuert)
- Top 10 Risiko-Liste
- Auftragsänderungen
- Metrikbericht (darin wird über das Fehleraufkommen berichtet)

4.4.2 Projektbesprechungen

4.4.2.1 Regelmäßige Projektbesprechungen

Für jedes Projekt werden monatliche Projektmanagement-Besprechungen (Projekt - Reviews) abgehalten, bei denen folgende Punkte behandelt werden:

- Darstellung des aktuellen Projektstandes
- Feststellen von Abweichungen (Termin, Kosten, Aufwand, Sachfortschritt)
- Festlegung von erforderliche Steuerungsmaßnahmen
- Überwachung der Top 10 Risikoliste
- Vorschau auf nächsten Monat

Teilnehmer bei diesen Besprechungen sind:
Abteilungsleiter, zuständiger Dienststellenleiter, Projektleiter, Qualitäts-Sicherungs-Verantwortlicher und Abteilungs-Qualitäts-Manager.

Die Besprechungstermine sind so gewählt, dass die Kontierungs-Endauswertung bereits durchgeführt worden ist und die aktuellen Daten (Ist-Aufwände, u.a.) im Projekt-Controlling-Tool vorhanden sind.

4.4.2.2 Ereignisgesteuerte Projektbesprechungen

Tritt ein besonderes, meist unvorhergesehenes, Ereignis im Projektverlauf ein, so ist vom Projektleiter eine Projektbesprechung mit obigem Teilnehmerkreis einzuberufen.

Als derartige Ereignisse gelten u.a. :

- Auftreten einer Projektkrise
- sich abzeichnende erhebliche Planabweichungen (Termin, Kosten, Sachfortschritt)
- sich abzeichnende Qualitätsmängel
- plötzlich eintretende Personalprobleme
- Auftreten von entwicklungstechnischen Sachproblemen
- Probleme bei Zulieferungen und Beistellungen

4.4.3 Risikomanagement

Das Risikomanagement besteht aus 6 Schritten:

- Risikoidentifikation
- Risikoanalyse
- Risikoprioritätenbildung
- Risikomanagementplanung
- Risiko-Überwindung
- Risikoüberwachung

1. Schritt: Risiko-Identifikation

Das Ergebnis einer Risiko-Identifikation ist eine Liste der projektspezifischen Risikoelemente, die den Projekterfolg gefährden. Um die Risiken zu identifizieren, bedienen wir uns Checklisten. Darüber hinaus werden bei Software-Entwicklungen immer wieder vorkommende Risiken betrachtet, wie etwa:

- Personelle Defizite
- Unrealistische Termin- und Kostenvorgaben
- Entwicklung von falschen Funktionen und Eigenschaften
- Entwicklung der falschen Benutzerschnittstelle
- Vergolden („über das Ziel schießen")
- Kontinuierliche Anforderungsänderungen !
- Defizite bei extern gelieferten Komponenten
- Defizite bei extern erledigten Aufträgen
- Defizite in der Echtzeitleistung
- Überforderung der Software-Technik
- uvm.

2. Schritt: Risiko-Analyse

Bei der Risiko-Analyse wird die Schadens-Eintrittswahrscheinlichkeit und das Schadens-Ausmaß für jedes identifizierte Risikoelement geschätzt. Außerdem werden zusammengesetzte Risiken abgeschätzt. Daraus werden Risikofaktoren ermittelt.

Risikofaktor = Wahrscheinlichkeit [0-1] * Schadensausmaß [1-10]

3. Schritt: Risiko-Prioritätenbildung

Um die relevanten Risiken zu identifizieren, werden die Risiken nach Prioritäten geordnet. Eine Möglichkeit besteht darin, nach den Risikofaktoren zu gliedern; es kann aber auch die Eintrittswahrscheinlichkeit oder das Ausmaß herangezogen werden.

4. Schritt: Risikomanagement-Planung

Hierbei geht es um das Etablieren von Risikokontroll-Aktivitäten, die das Ausmaß des Risikos begrenzen sollen. Dazu werden Risikomanagementpläne entwickelt, die die notwendigen Aktivitäten festlegen. Für jedes Risiko ist ein Risikomanagement-Plan zu entwickeln. Stellt z.b. das geforderte Zeitverhalten ein Risiko dar, sollte die Entwicklung eines Prototypen überlegt werden. Die Planung muss Aufschluss auf die Fragen: Warum, Was, Wann, Wer, Wo, Wie und Wie viel geben. Nach Abschluss der Planung sind die entstandenen Pläne in den Projektplan zu integrieren.

5. Schritt: Risikoüberwindung

Nach Abschluss der Risikoplanung werden die dort festgelegten Aktivitäten ausgeführt; z.b. ein Prototyp erstellt oder die Anforderungen gelockert.

6. Schritt: Risiko-Überwachung

Die Fortschritte bei der Risiko-Minimierung werden überwacht. Bei Abweichungen werden korrigierende Maßnahmen vorgenommen. Unter Berücksichtigung der Top-10-Risiken sieht der Prozess folgendermaßen aus:

a) Risikoelemente in eine Rangfolge bringen
b) Durchsprache der Liste in den Projektmanagement-Besprechungen
c) In diesen Besprechungen wird vom Projektleiter über den Fortschritt bei den Top-10-Risikoelementen berichtet. Die Übersicht sollte die Rangordnung jedes Risikoelementes angeben, den Rang bei der letzten Durchsprache und wie oft das Element schon auf der Top-10-Liste stand. Außerdem sollte angegeben werden, wie sich das Risiko seit der letzten Sitzung entwickelt hat.
d) Die Sitzung soll sich darauf konzentrieren, die Risikoelemente zu beseitigen

Eine solche in c) angesprochene Liste hat beispielhaft folgende Struktur (vgl. Tab. 4.1):

Risikoelement	Monatsrang			Fortschritt bei
	Dieser Monat	Letzter Monat	Anzahl Monate	Risikoüberwindung
1	1	4	2	...
2	2	5	3	...
3	3	3	3	...
4	4	2	2	...
5	5	1	3	...
6	6	-	1	...
7	7	8	3	...
8	8	6	3	...
9	-	7	2	Erledigt
10	-	9	3	Erledigt

Tab. 4.1: Top-10-Liste

Diese Top-10-Liste wird im Projektplan laufend fortgeschrieben.

Wenn sich identifizierte allgemeine Risiken im Laufe des Projektes auf mehrere (konkrete) Risiken aufteilen, werden die kritischen Teilrisiken überwacht.

4.5 Projektabschluss

4.5.1 Phasenabschlussbesprechungen

Bei größeren Projekten ist pro Phase (Meilenstein) eine Phasenabschlussbesprechung durchzuführen. Wichtig ist, dass vor dieser Besprechung eine umfassende Phasenabschluss-Vorbereitung gemacht wurde, um dort aufgrund von Tatsachen Entscheidungen zu treffen. Damit wird vermieden, dass in dieser Besprechung unnötige Diskussionen (z.b: über Erreichen oder Nichterreichen von Prozenten bei der Testabdeckung) stattfinden. Bei dieser Besprechung ist ebenfalls der Plan für die nächste Phase zu überprüfen.

4.5.2 Metriken

Aus der Vielzahl von möglichen Metriken konzentrieren wir uns auf ein einfaches Metrikprogramm. Ziel ist es, Fehler schon in den frühen Phasen eines Projektes zu finden. Zur Erreichung dieses Zieles ist in erster Linie das Reviewgeschehen (für z.b. Entwurfsdokumente, Code, usw.) zu verbessern. Um die Erreichung dieses Zieles überprüfen zu können, ist es notwendig, alle gefundenen Fehler quantitativ zu erfassen (im Projekt-Controlling-Tool).

Fehler werden unabhängig von der verwendeten Entwicklungsmethode folgenden Abschnitten zugeordnet:

- Definition
- Entwurf
- Codierung
- Komponententest
- Integrations- und Systemtest
- Einsatz

Als Bezugsgröße werden die ermittelten Function-Points für das Gesamtprojekt verwendet.

Als groben Vergleichswert kann man gemäß allgemeiner Literatur annehmen, dass die Anzahl der Fehler in einem Projekt gleich (Function Points)^1.25 ist. Auch wird ein aus persönlichen Gesprächen bekannter Zusammenhang zwischen CMM-Level (charakterisiert den Reifegrad einer Software-Entwicklungs-Organisation) und der Fehlerverteilung auf die oben angeführten Abschnitte eines Software-Entwicklungs-Projektes angenommen (vgl. hierzu Tab. 4.2).

Fehler nach Phasen in %	**Definition**	**Entwurf**	**Codierung**	**Komponenten-Test**	**Integrations- und System-Test**	**Einsatz**
Level 3	2	5	28	30	30	<5
Level 2	1	2	7	30	50	10
Level 1	0	0	5	15	65	15

Tab. 4.2: Fehlerverteilung auf Phasen in einem Software-Entwicklungs-Projekt

4.5.2.1 Function-Point - Zählung

Am Ende der Phase Definition ist eine neuerliche Function-Point Zählung durchzuführen und das Ergebnis ins Projekt-Controlling-Tool einzutragen.

4.5.2.2 Verbesserungen im Prozess

Um anhand der oben beschriebenen Metriken Verbesserungen im Prozess durchführen zu können, ist es notwendig, Fehler zu klassifizieren. Dies geschieht dadurch, dass Fehler definierten Fehlerklassen zugeordnet werden. Die im Komponententest gefundenen Fehler werden den Fehlerklassen Modulspezifikation und Code zugeordnet. Im Integrationstest erweitert sich die Fehlerklassifizierung auf die Gruppen Modulschnittstellen-Fehler und Architekturfehler. Im Systemtest besteht zusätzlich die Möglichkeit Fehler in die Kategorie der Anforderungsfehler einzuordnen. Im Rahmen der Festlegung des einzusetzenden Fehlermanagement-Tools ist die Erfassung der Fehlerklassifikation zwingend vorgesehen.

Anhand von Analysen der Fehlerklassifizierung werden Rückschlüsse gezogen, inwieweit Fehler nicht schon in früheren Phasen vermieden werden hätten können. Die daraus gewonnenen Ergebnisse werden in konkrete Verbesserungsmaßnahmen umgesetzt, wie z.b. mehr qualitativen Aufwand ins Design-Review verlagern, oder mehr Code-Reviews durchführen, usw.).

4.5.2.3 Prognose

In einem weiteren Schritt wird empfohlen, die Metrikdaten auch für Prognosen und zum Projektcontrolling heranzuziehen (z.b. Überprüfen der Testendekriterien).

4.6 Change Request Verfahren

4.6.1 Ziele

Ziel des Change Request (CR) - Verfahrens ist es, alle in den Projekten anfallenden Änderungsanforderungen nachvollziehbar abzuwickeln. Somit regelt das CR-Verfahren folgende Projektaufgaben zum Teil oder gänzlich:

- Unterscheidung zwischen Fehler und Änderung
- Sicherstellung eines planbaren und überschaubaren Änderungsgeschehens
- planbare Aufwands- und Terminverfolgung
- leistungsgerechte Abrechnung von Zusatzaufträgen und Änderungswünschen

Dabei soll zwischen dem prinzipiellen Prozess und der Nutzung des Toolings, das diesen Prozess unterstützt, unterschieden werden. Während der CR-Gedanke stets Element eines allgemeinen Projektmanagements ist (siehe Standardliteratur), orientiert sich der nachfolgend näher definierte Prozess an den konkreten Gegebenheiten. Die zur Unterstützung des CR-Verfahrens eingesetzten Tools richten sich nach dem konkreten Projekt, dem Auftraggeber, dem Projektleiter, usw.

4.6.1.1 Change Requests

Im Gegensatz zu periodischen Tätigkeiten (z.b. Aufwandsverfolgung, usw.) muss für die Anwendung des CR-Verfahrens ein auslösendes Ereignis – eine Änderungs-Anforderung (Change Request (CR)) - vorliegen. Der früheste Zeitpunkt des Auftretens eines CRs ist zu Beginn der Entwurfsphase – also nach Fertigstellung des Angebots und des Pflichtenhefts. Change Requests sind Auftragsänderungen (z.b. andere Schnittstellen-Anforderungen, andere Normen und Standards, zusätzliche Funktionalität, usw.).

Da es in manchen Fällen nicht sofort möglich ist einen CR von einem Fehler zu unterscheiden, werden Fehlermeldungen und Change Requests zunächst unter dem Begriff Modification Request (MR) zusammengefasst. Fehlermeldungen (Fehler = Abweichungen vom geplanten Verhalten) werden sowohl projektintern (z.b. vom Integrations- und Systemtest), als auch projektextern (z.b. bei der Abnahme oder vom Kunden) generiert.

4.6.2 Prozess

Das Change-Management-Verfahren definiert somit den Prozess vom Auftreten eines Modification Requests bis zu dessen Bearbeitung. Je nachdem, ob es sich um einen Change Request oder einen Fehler handelt (beides sind MRs), müssen unterschiedliche Schritte gesetzt werden. Besondere Bedeutung liegt bei der frühzeitigen Festlegung der Verantwortlichkeiten. Dazu ist schon im Angebot auf die Notwendigkeit hinzuweisen. Im weiteren Projektverlauf sind sämtliche die Change Requests betreffenden Entscheidungen und Vorgänge in einem Projekthandbuch zu dokumentieren.

Für CRs bedarf es schon zu Projektbeginn der Festlegung von CR-Ansprechpartnern beim Auftraggeber. Die genannten Ansprechpartner leiten bei Bedarf die CRs an die im Projekt definierte Clearingstelle weiter, wo sie eindeutig identifiziert und dem CR-Koordinator zur Analyse vorgelegt werden. Sind MRs offensichtlich in die Kategorie der Fehler einzureihen, entfällt der Weg über die CR-Clearingstelle.

4.6.2.1 Analyse

Zur Entscheidungs-Vorbereitung ist jeder CR vom CR-Koordinator zu analysieren. Hierbei werden folgende Eigenschaften geklärt:

- Klassifizierung in: CR oder Fehlermeldung
- Priorität
- prinzipielle Lösungsstrategien
- Auswirkungen auf Schnittstellen, andere Systemteile, Performance, Stabilität, usw.
- geschätzter Aufwand (inklusive Testaufwand)
- objektivierte Bewertung nach Function Points
- Realisierungszeitpunkt
- Kapazitätserfordernisse

Die erste Analyse erfolgt durch die Projektverantwortlichen in Form einer persönlichen ersten Stellungnahme. Kann keine sofortige Auskunft erteilt werden, muss eine Besprechung zu diesem Thema stattfinden.

4.6.2.2 Entscheidung und Eingliederung in den Projektablauf

Wird im Rahmen der Analyse ein CR bestätigt, muss vom Projekt-CR-Koordinator der Auftraggeber-CR-Koordinator kontaktiert werden. Abhängig von der Projektgröße erfolgt die Entscheidung zur Bearbeitung des CRs entweder im Rahmen einer Besprechung zwischen Auftraggeber, Projektleitern und Entwicklern, oder innerhalb eines eigens dazu installierten Change Control Boards. Die Entscheidungen beinhalten die verbindliche Zustimmung des Auftraggebers zu Änderungen der ursprünglich vereinbarten Kosten, Termine und Leistungen.

Um die Übersichtlichkeit zu bewahren werden bis zur Integration in das Projekt eigene Rahmenterminpläne für CR's geführt. Spätestens zum Zeitpunkt der Entscheidung zur Realisierung muss ein CR einer geplanten Produkt-Version zugeordnet werden und unterliegt ab dann dem normalen Projektablauf.

5. Kernprozesse eines Softwarehauses

Die folgenden Ausführungen orientieren sich hinsichtlich der Gliederung an den Produktenstehungsprozessen in der Telekommunikations-Sparte der Siemens AG. Da die Software dieser Sparte sehr komplex und umfangreich ist, besitzen die – auch aufgrund eigener Erfahrungen beschriebenen – Sachverhalte eine generelle Gültigkeit in allen komplexen Umgebungen (wie z.b. in der Luft- und Raumfahrt [1]), für die Software erstellt wird. Die Ausführungen hinsichtlich des Einsatzes von OEM-Produkten (= Original Equipment Manufacturer - Produkt) bzw. COTS-Produkten (= Commercial off-the-shelf - Produkt) basieren weitestgehend auf den eigenen beruflichen Erfahrungen des Dissertanten.

5.1 Einleitung

5.1.1 Produkt versus Projekt

Generell sind im Projektmanagement zwei Betrachtungsgegenstände klar zu unterscheiden [2]):

- Produkt (= Objektsystem, Ergebnis)
- Projekt (= Handlungssystem, Prozess)

Das Produkt ist das Ergebnis eines spezifischen Projekts. Entsprechend der Art der Strukturierung lässt sich das **Produkt** durch Objektstrukturpläne (z.b. Pflichtenheft, Leistungsmerkmalübersicht, u.a.) oder Funktionspläne (z.b. Flussdiagramme, Aktivitätsdiagramme, u.a.) beschreiben. Dieselbe Gliederung – nach statischen und dynamischen Gesichtspunkten – lässt sich auch auf das **Projekt** bzw. den Prozess anwenden. Bei hierarchischer Gliederung spricht man von Projektstrukturplänen; bei Prozessorientierung hingegen von Projektablaufplänen (z.b. Balkenpläne, Meilensteintrendanalysen, u.a.).

Da in der Software-Branche Produkte selten einzeln (ohne Variationen oder Weiterentwicklung) abgesetzt werden, spricht man oft von einer **Produktlinie.** Diese ist zwar einem übergeordneten Thema zugeordnet, besitzt aber trotzdem anwendungsspezifische Komponenten, die nicht immer gleichzeitig installiert werden müssen. Charakteristisch für eine Produktlinie ist des weiteren eine Versionierung, die die Weiterentwicklung kennzeichnet.

---

[1] http://www.stsc.hill.af.mil/crosstalk/ , CrossTalk: The Journal of Defense Software Engineering, 2004
[2] Patzak, Gerold, Rattay, Günter, Projektmanagement, Wien, 1998

5.1.2 Produkt versus Projekt in der Software-Branche

Die Abgrenzung zwischen Produkt und Projekt ist in der Software-Industrie durch folgende Kriterien bestimmt:

- Während ein Software-Produkt für mehrere Kunden gedacht ist, wird ein Software-Projekt spezifisch an den Erfordernissen eines einzigen Kunden ausgerichtet.

- Da ein Software-Produkt (vor allem in der Telekommunikations-Industrie) stets mit anderer Software kommuniziert, hat die Weiterentwicklung (Änderung) eines Produkts auch die Änderung aller mit ihm potentiell in Kontakt stehenden Produkten zur Folge. Im Unterschied dazu, muss im Falle eines Software-Projekts nur die konkrete Umweltsituation in Betracht gezogen werden, wodurch sich der Aufwand erheblich reduziert. Bei Produkten wird in diesem Zusammenhang der Aufwand größtenteils durch die Versionspflege und die Integrationstests verursacht.

- Geht man von der Annahme aus, dass ein Software-Produkt an mehrere Kunden verkauft wird, jedoch keiner Weiterentwicklung unterliegt, besteht der Unterschied zum Software-Projekt im Kopieren der Software und der mehrmaligen Inbetriebnahme. Da der Fertigungsvorgang (Kopiervorgang) keinen nennenswerten Aufwand darstellt, ist der Entwicklungsprozess bis zur Fertigstellung für das Produkt und das Projekt weitestgehend identisch. Dem gegenüber bestehen Unterschiede bei den Prozessen zur Initiierung und der Produkt- bzw. Projektausphasung.

- Ein weiterer, nicht vernachlässigbarer Unterschied besteht hinsichtlich dem wirtschaftlichen Risiko. Im Falle eines Software-Projekts besteht ein Risiko hauptsächlich durch Termin- und Aufwandsüberschreitungen. Im Falle eines Software-Produkts besteht das Hauptrisiko in der richtigen Einschätzung der Absatzmenge. Gleichzeitig liegt darin aber auch eine große Chance, da mit jedem zusätzlich verkauftem Stück der Preis des Produkts sinken kann, was wiederum die Eintrittsbarriere für Konkurrenten erschwert. Ein markantes Beispiel hierfür sind die Produkte der Firma Microsoft.

- Oft ist der Unterschied zwischen einem Software-Produkt und einem Software-Projekt nicht eindeutig. Viele Produkte sind die Weiterentwicklung von anfänglich ähnlichen Projekten. Dadurch lassen sich die Kosten erheblich reduzieren und die Marktpräsenz erhöhen.

5.1.3 Allgemeine Gliederungsgesichtspunkte

Im vorhergehenden Abschnitt wurde der Begriff des Projekts synonym zum Begriff der Systemumgebung (innerhalb der das Produkt entsteht) verwendet. Um das Projekt zu untergliedern können u.a. folgende Kriterien herangezogen werden [3]):

- Objekte / Topographie: Gliederung der Planungsobjekte in abgrenzbare Subsysteme
- Funktionen: Gliederung der Planungsobjekte nach Kriterien der späteren Nutzung
- Leistungen: Gliederung des Projekts nach Leistungsbereichen
- Verantwortung: Gliederung des Projekts nach Verantwortungsbereichen
- Phasen: Gliederung des Projekts nach zeitlichen Abschnitten
- Realisierungsschritte: Gliederung der Planungsobjekte nach der Abfolge der Realisierungsschritte
- Aufwand: Gliederung des Projekts nach Kostenarten

Software-Projekte sind in der Regel nach mehreren Kriterien gleichzeitig gegliedert. Dies ist inhärent mit der Tatsache, dass die Parameter eines Software-Projekts unterschiedliche Ausprägungen besitzen. Im nachfolgenden Abschnitt wird als Kriterium der ersten Gliederungsebene der Prozess (phasenbezogene Leistungen) gewählt.

Unabhängig von der Gliederung sind 2 Prinzipien stets einzuhalten:

- Disjunktheit der Untergliederung: Diese fordert, dass sich die Strukturelemente einer Ebene nicht überlappen dürfen. In der Software-Entwicklung entspricht dies z.b. der Forderung, dass keine Funktionalität mehrmals von verschiedenen Teams entwickelt zu werden braucht.

- Vollständigkeitsregel: Bei Untergliederung eines Strukturelements muss die Summe der Teilmengen mit der Ursprungsmenge übereinstimmen.

[3] Wojda, Franz, Projektorganisation – Projektmanagement, Wien, 2001

5.1.4 Kernprozesse der Software-Produktion

Wie im allgemeinen Teil beschrieben, wird ein Projekt nach mehreren Dimensionen strukturiert. Hierbei haben - abgesehen von der Produktgliederung - im wesentlichen 2 Dimensionen eine ausgezeichnete Stellung:

- Prozessgliederung
- Organisationsgliederung (wird an anderer Stelle diskutiert)

Die Prozessgliederung beschreibt die Ablauforganisation, welche sich anhand der derzeitigen Praxis der Software-Branche in folgende 4 zentrale Prozesse unterteilt (vgl. Abb. 5.1):

- Chancen-Identifizierungs-Prozess (CIP)
- Produkt-Management-Prozess (PMP)
- Produkt-Bereitstellungs-Prozess (PBP)
- Original Equipment Manufacturer-Prozess (OEMP)

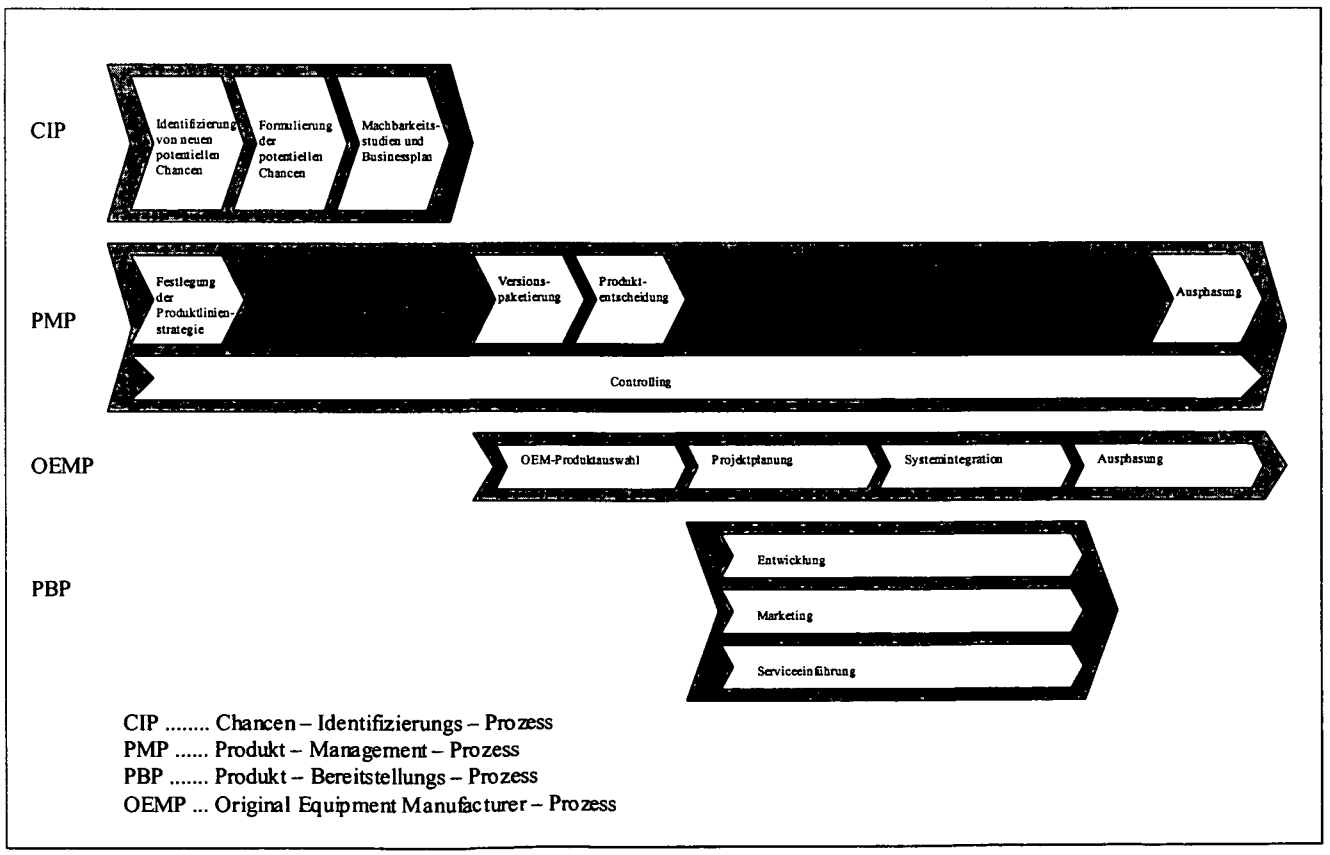


Abb. 5.1: Kernprozesse

Erklärung des Begriffs "Original Equipment Manufacturer": Hierbei handelt es sich um ein Produkt das nicht aus der eigenen Entwicklung stammt, sondern zugekauft wird. Da der Anteil zugekaufter Software am Gesamtprodukt innerhalb der Software-Branche eher zu- als abnimmt, sind die damit verbundenen Konsequenzen von zunehmender Bedeutung.

Während der CIP am Anfang und der PBP in der Mitte eines Produkt-Entstehungs-Zyklus abläuft, bildet der PMP die Klammerfunktion und steuert alle nicht von CIP und PBP notwendigen Aktivitäten. Aufgrund des vielfältigen bereits am Markt existierenden Software-Angebots ist für viele Problemlösungen keine Software-Neuentwicklung notwendig. Stattdessen werden auf Basis von Fremdprodukten Adaptierungen vorgenommen, oder das geplante (Teil-)Vorhaben auf Kooperationsbasis realisiert. Über gemeinsame Synchronisationspunkte werden die Aktivitäten der unterschiedlichen Prozesse, die zudem von separaten Organisations-Einheiten wahrgenommen werden, verknüpft.

5.2 Chancen-Identifizierungs-Prozess (CIP)

Im CIP wird kontinuierlich nach neuen Chancen, d.h. Geschäfts-Möglichkeiten gesucht. Mit der Ausdrucksweise der heute gebräuchlichen Terminologie laufen diese Prozesse in sogenannten „Think Tanks" ab. Dabei sind die Quellen für neue Ideen sehr unterschiedlich. Im wesentlichen stammen sie von folgenden Gruppen:

- Kunden (Nachfragemarkt)
- Konkurrenz (Angebotsmarkt)
- Mitarbeitern des eigenen Hauses

Eine Geschäfts-Möglichkeit wird hierbei durch folgende Charakteristika beschrieben: Sie ist entweder ein Produkt, ein zusätzliches Leistungsmerkmal eines bereits bestehenden Produkts, oder eine Dienstleistung. Zudem existiert ein potentieller Markt auf dem die Idee während eines bestimmten Marktfensters einen bestimmten Preis erzielt.

Die Schritte im CIP sind die folgenden:

5.2.1 Identifizierung von neuen potentiellen Chancen

In dieser Phase wird höchster Wert auf Kreativität gelegt. Das hiermit beauftragte Team ist weitestgehend vom Tagesgeschäft losgelöst, steht aber in aktivem Kontakt mit anderen Organisationseinheiten. Diese wichtigsten Einheiten sind:

- **Sales- und Service-Abteilungen**, die den Kontakt zum Kunden haben. Während die Sales-Mitarbeiter eher mit den kaufmännischen Aspekten konfrontiert sind, besitzen die Service-Mitarbeiter Informationen über technische Realisierungswünsche.
  Die Sales-Abteilungen haben den Zugang zu den Investitionsbudgets der Kunden und kennen deren Absicht in bestimmte Segmente investieren zu wollen. Der Service kann wertvolle Informationen über die Konkurrenzprodukte in Erfahrung bringen, da ein Kunde meist ähnliche Systeme von verschiedenen Anbietern gleichzeitig im Einsatz hat. Dies schafft dem Kunden den Vorteil, verschiedene Anbieter im eigenen Haus vergleichen zu können, und andererseits nicht in eine Angebotsmonopolstellung zu geraten.

- **Marketingabteilungen**, die einen umfassenden Blick über die Marktsegmentierung, die Konkurrenz, die Absatzwege und die erzielbaren Preise haben. Dies ist insbesondere in Industriegüter-Märkten keine leichte Aufgabe, da die Konkurrenzprodukte bzw. –projekte nur bedingt bekannt sind.

  Wenn eine Vorführung im Rahmen einer Präsentation unzureichend ist – was in der Industrie-Software-Branche in der Regel der Fall ist – müsste das Produkt eigens zum Zweck der Analyse im eigenen Haus installiert werden. Dies ist zumeist ein teures Unterfangen. Denn selbst wenn die dafür notwendigen Lizenzen zu Testzwecken gratis zur Verfügung gestellt werden, muss es in eine Systemlandschaft eingebunden werden, die nur mit großem Aufwand realisiert werden kann. Zudem ist eine geeignete Hardware anzuschaffen, die für diese Kategorie von Software teuer ist.

  Als Alternativen bieten sich enge Kontakte zum Kunden an, wie sie zum Beispiel in der Telekommunikations-Industrie durch sogenannte „User Groups" organisiert sind. Dies sind meist quartals- oder halbjährlich stattfindende Konferenzen, an denen einerseits ein Lieferant eines bestimmten Produkts oder einer Produktgruppe, und andererseits deren Kunden, teilnehmen. Darin werden dann in konzentrierter Form die Kunden-Feedbacks eingeholt. Eine andere Möglichkeit wäre der Einstieg ins Kundengeschäft, d.h. dass z.b. ein Telekom-Ausrüster selbst ein Betreibergeschäft (z.b. Siemens hatte eine hohe Beteiligung an MaxMobil) unterhält. Der darin liegende Nachteil der Schaffung einer Konkurrenzsituation zu den eigenen Kunden.

- **Entwicklungsabteilungen**, die aufgrund ihrer Systemkenntnisse neue Kombinationen der Informationsverknüpfung erkennen. Oft sind auch dieselben Technologien in sehr unterschiedlichen Branchen einsetzbar. Ein Beispiel aus der Praxis: Aufgrund eines zufälligen Vergleichs der jeweiligen Software wurde erkannt, dass die Anforderungen zur Überwachung von Fehlern in Telekom-Netzen mit jenen zur Verarbeitung der Signale von Testsprengungen zur Erdölsuche überraschend ähnlich sind.

5.2.2 Formulierung der potentiellen Chancen

Hierin werden die zunächst als Ideen vorhandenen Chancen strukturiert und so dargestellt, dass eine Bewertung aus Kunden-, Anwender- und Geschäftssicht möglich wird. Wesentliche Punkte dieser strukturellen Darstellung sind der Inhalt mit Angaben zu:

- Geschäfts-Idee-Koordinator
- funktionelle Kurzbeschreibung
- Nutzensbeschreibung
- Ideenursprung
- Patentrelevanz
- ungeklärte Probleme und Risiken
- Schnittstellen
- Korrelation zu anderen Geschäftsideen
- Wettbewerber

und ein erster Entwurf eines Business-Plans mit Angaben zu:

- Markt- und Umsatzpotential
- Entwicklungskosten
- EBIT und EVA
- Marktfenster

5.2.3 Machbarkeitsstudien und Business Plan

Ziel dieses Punktes ist die weitere Konkretisierung, sodass eine Entscheidung über die Realisierung getroffen werden kann. In diesem Schritt erfolgen die ersten Untersuchungen zur Einbindung von OEM-Produkten (= Original Equipment Manufacturer), die die Kosten und die Entwicklungszeit zu reduzieren helfen. Des weiteren werden auch Überlegungen zur Versionspaketierung angestellt, auf dessen Basis der Business-Plan kalkuliert wird.

Im Detail werden gegenüber dem vorigen Schritt die Ergebnisse verfeinert und folgende zusätzliche Punkte geklärt:

- Untersuchung von Buy-, Make- und Reuse-Alternativen
- Identifikation von OEM-Produkten
- Architekturprinzipien
- Umwelt- und Aufbaubedingungen
- Abschätzung der Auswirkungen auf andere Systeme
- Teststrategien

- Zielgruppenabschätzung
- Mengenabschätzung
- Preisempfehlung

5.3 Produkt-Management-Prozess (PMP)

Der Produktmanagementprozess ist der zentrale Prozess der geschäftssteuernden Organisationseinheit. Er erstreckt sich über die gesamte Lebensdauer des Produkts. In der Regel handelt es sich beim Eigentümer dieses Prozesses um ein Profit-Center dem eine Person mit Prokura vorsteht. Zu den wichtigsten Aufgaben dieses Prozesses zählen:

5.3.1 Verantwortung für die Produktlinien-Strategie

Hierzu zählen sämtliche strategischen Aufgaben, die auch die Organisation der Umwelt beinhalten. Mit der Umwelt ist sowohl die firmeninterne als auch die externe angesprochen. Firmenintern sind die Anforderungen an die Organisations-Struktur anzupassen. Dies bedeutet im wesentlichen die Verteilung der Kompetenzen auf die entsprechend des Entwicklungs-Fortschritts betroffenen Organisations-Einheiten (wie z.b. Entwicklung, Service, Vertrieb, u.a.)

Die Strategie muss konform zum Portfolio und der Marktsegmentierung ausgerichtet sein. Aufgrund der hohen Personalintensität in der Software-Entwicklung ist die Personal-Ressourcen-Planung stets einer der wichtigsten Eckpfeiler. Des weiteren ist die Absatzsteuerung mit Planungen hinsichtlich Pilotkunden, Umsatz-, Preis- und Kostenzielen immer in der Verantwortung des PMP-Eigentümers. Für die Markterschließung ist insbesondere Wert auf die Erfolgsfaktoren im Vergleich zur eigenen Situation zu legen. Zur Darstellung eignen sich vorzüglich Portfolio-Konzepte, wie sie von namhaften Consulting-Firmen beworben werden. Für die Ableitung der Anforderungen auf operativer Ebene eignet sich z.b. die Business Score Card, die um detaillierte Darstellungen ergänzt wird.

5.3.2 Versionspaketierung

Die Aufgabe der Versionspaketierung ist die schrittweise Einführung des Produkts am Markt. Da Software-Produkte im Regelfall einer kontinuierlichen Verbesserung unterliegen, und der Ressourceneinsatz in einem angepassten Verhältnis zum Risiko stehen muss, enthält die erste Ausbaustufe des Produkts nur die für den ersten Marktauftritt wesentlichen Leistungsmerkmale.

Selbst im Falle eines einmaligen Projekts werden umfangreiche Vorhaben in mehreren Schritten realisiert. Dadurch wird auch das Risiko des Kunden minimiert, der meist parallel mit der Einführung der neuen Software interne Umstrukturierungen vornehmen muss. Zudem kommt es nicht selten zu Projektabbrüchen bzw. Teilfertigstellungen wenn sich zwischenzeitlich die Umweltbedingungen für den Kunden verändert haben.

Die Darstellung dieser stufenweisen Realisierung erfolgt in einer Roadmap. Darüber hinaus werden die Ergebnisse in Business Plänen (z.b. basierend auf dem EVA-Konzept), in Ressourcenplänen und Rahmenterminplänen dargestellt. Die Zusammenfassung dieser Detailergebnisse geschieht im Projekthandbuch.

5.3.3 Controlling

Zentrale Aufgabe des Controlling ist die Überwachung der durch die Strategie vorgegebenen Aktivitäten und Planwerte. In der Software-Branche liegt der Fokus auf der Fortschrittskontrolle. Dies beinhaltet sowohl eine kontinuierliche Aktualisierung der Reporte, als auch die gezielte Intervention bei den für die Abweichung Verantwortlichen. Auf der anderen Seite ist mit dem Vertrieb ein regelmäßiger Kontakt zu halten, um Nachfrageverschiebungen rechtzeitig zu erkennen. Dies beinhaltet auch eine Aufwands- und Ertragskontrolle, um die Einhaltung der finanziellen Ziele zu gewährleisten.

Um ein umfassendes Controlling zu gewährleisen, das zudem mit geringem Aufwand realisierbar ist, empfiehlt sich der Einsatz von erfahrenem Personal, das die Risiken von Abweichungen und Schwachstellen adäquat einschätzen kann.

5.3.4 Ausphasung

Die Ausphasung hat in einer Weise zu geschehen, dass der Kunde ungehindert seine Geschäfte fortführen kann. Im Falle von einmaligen Projekten, ist der Zeitpunkt der Ausphasung vertraglich festgelegt. Im Falle von Produkten, die durch verbesserte Versionen ersetzt werden, muss eine Update-Strategie eine kontinuierliche Anwendung der Software gestatten. In diesem Zusammenhang muss die Weiterverwendung von alten Kunden-Datenbeständen – meist als Datenmigration bezeichnet – gewährleistet sein.

Zudem ist auch auf die Systemumwelt Rücksicht zu nehmen, sodass die von Kundenseite eventuell vorgenommenen Adaptionen auch bei Einsatz der nächsten Version funktionsfähig bleiben. Ein weiteres, speziell in der Softwarebranche anzutreffendes Kriterium, ist die Abstimmung des Lebenszyklus einer Softwareversion auf jenen der erforderlichen Hardware.

5.4 Produkt-Bereitstellungs-Prozess (PBP)

Ziel dieses Prozesses ist die schnellstmögliche Markteinführung eines Produkts bzw. die Fertigstellung eines Projekts, wobei vorausgesetzt werden darf, dass der Chancen-Identifizierungs-Prozess (CIP) abgeschlossen ist. Um die Zeitspanne bis zur Produktlieferung zu minimieren, werden alle Arbeitspakete bestmöglich parallelisiert. Dabei lassen sich folgende drei parallel ablaufende Teilprozesse identifizieren:

- Entwicklung
- Markteinführung
- Serviceeinführung

Während unter der Entwicklung der klassische Schritt der Realisierung eines Produkts gesehen wird, ist der Erfolg vieler Unternehmen der Software-Branche immer stärker durch eine gelungene Markteinführung und kundenfreundliche Service-Gestaltung geprägt.

5.4.1 Entwicklungs-Prozess

Im Entwicklungsprozess steckt der größte Teil des Know-hows eines „Software House". Für die Diskussion einzelner Details sei auf die speziellen Abschnitte hingewiesen. Im folgenden soll ein Überblick gegeben werden, der den Blick auf die anderen notwendigen Prozesse nicht verdeckt.

Generell wird der Entwicklungsprozess zunächst in Phasen untergliedert, die ganz allgemein in der Softwarebranche wie folgt bezeichnet werden:

1. Analyse
2. Entwurf
3. Implementierung
4. Integration
5. Systemtest
6. Kundenabnahme

5.4.1.1 Analyse-Phase

Die Analyse-Phase geht davon aus, dass die Funktion des Produkts hinreichend definiert wurde, um die Teilfunktionen – auch Leistungsmerkmale genannt – im nächsten Schritt formal zu erfassen. Dies bedeutet auch, dass die Leistungsmerkmale in diesem Schritt noch weitgehend in Form graphischer Skizzen und beschreibendem Text gehalten sind. Auch darüber hinausgehend ist diese Phase noch wenig formalisiert und wird stark durch die Organisationsstruktur und die Mitarbeiter bestimmt. In dieser Stufe finden häufig Besprechungen, Workshops und Review-Meetings statt. Ziel ist es ein gemeinsames Verständnis über die Funktionalität und Realisierungsmöglichkeiten der Leistungsmerkmale zu gewinnen.

Als Ergebnis liegt am Ende der Analyse-Phase folgendes vor:

- Rahmenterminplan
- umfassende Leistungsmerkmal-Spezifikation
- Zusammenfassung von Leistungsmerkmalen zu Subsystemen
- Zuordnung der Verantwortung für die Subsysteme und Leistungsmerkmale
- Definition der Schnittstellen zwischen den Subsystemen
- Festlegung der Synchronisationspunkte im Falle einer versionierten Entwicklung
- Planung der Entwicklungshilfswerkzeuge
- Planung der Teststrategie und des Testbedarfes
- Berücksichtigung von Zertifikaten und Exportbeschränkungen
- Zielwert für Fehler im Feld

Zusammengefasst sind diese Informationen in einem Projekthandbuch.

5.4.1.2 Entwurfs-Phase

Zu Entwurfsbeginn sollte ein vollständiges Verständnis der Funktionalität vorliegen. Nun wird – häufig unter dem Einsatz von Software-Tools – mit der Formulierung der Lösung begonnen. Dazu werden die Schnittstellen der Subsysteme und der darunter liegenden Module formuliert, wobei die Module die kleinsten Arbeitspakete darstellen. Diese hierarchische Gliederung der Software wird insbesondere durch die objekt-orientierten Sprachen sehr erleichtert. Die Qualität des Entwurfs muss einen Grad erreichen, der die Erzeugung des vollständigen Codes in der nächsten Phase gestattet.

Im Rahmen einer „Produktion" werden die Module anschließend auf ihre gegenseitige Kompatibilität hin getestet. Unter der Produktion versteht man das Verknüpfen der zum

jeweiligen Zeitpunkt vorhandenen Software – ausgehend von den Schnittstellen-Definitionen. Anhand der Schnittstellen-Definition werden die Kommunikationspartner eindeutig festgelegt. Durch den Test wird sichergestellt, dass jedes Modul die für seine Teilaufgabe notwendigen Parameter bereitgestellt bekommt und seine Ergebnisse auch zur Verfügung stellen kann.

Unter den erwähnten Tools ist generell Software zu verstehen, die die Erzeugung von Software unterstützt. Dies kann in der Entwurfs-Phase durch graphische Visualisierungen der Funktionalität, oder, in der Implementierungs-Phase durch automatische Generierung von Code, erfolgen.

5.4.1.3 Implementierungs-Phase

Dies ist der ureigenste Kern der Software-Erstellung. Hier wird der Sourcecode – teilweise mit Hilfe von Tools – geschrieben. Die in der Entwurfs-Phase definierte Funktionalität wird auf Modulebene realisiert. Parallel zur Codierung der Module werden auch Testsequenzen geschrieben, die der Überprüfung der einzelnen Module dienen. Im allgemeinen Sprachgebrauch wird dieser Modultest auch Komponententest genannt. Da die Module zum Zeitpunkt dieses Tests noch nicht Teil des Gesamtsystems sind, gehören diese Tests ins Umfeld der Offline-Tests (im Gegensatz zu den Online-Tests).

Am Ende dieser Phase liegt der komplette Sourcecode mit den komponenten-getesteten Modulen vor. Des weiteren existiert ein erster Entwurf einer Kundendokumentation und Planungsergebnisse für die Online-Tests. Die Online-Tests müssen alle Leistungsmerkmale des Gesamtsystems zu überprüfen imstande sein.

5.4.1.4 Integrations-Phase

In diesem Schritt werden alle Komponenten zusammengebaut und als Gesamtsystem getestet. Die Systemumwelt wird dazu teilweise durch Simulationen ersetzt. Am Ende dieser Phase liegen die technischen und kaufmännischen Grunddaten für den Vertrieb vor. Dazu zählen die Konfigurationsregeln, Update- und Upgrade-Regeln, sowie sonstige für das Marketing notwendige Informationen.

5.4.1.5 Systemtest-Phase

Dies ist die letzte Phase der Tests. Je nach Möglichkeit wird das System in der eigenen Organisation oder beim Kunden auf volle Funktionalität hin überprüft. In vielen Projekten ist

dieser Online-Test sehr kostenintensiv und verlangt eine präzise Einteilung der Ressourcen. Im Rahmen dieser Tests werden auch die Kundendokumente auf Konsistenz mit der Software überprüft, da sie die Grundlage für etwaige spätere Ansprüche seitens des Kunden darstellen.

Systemtests kommen auch häufig in Form von Regressionstests vor. Dies ist dann der Fall, wenn nur Teile des Gesamtsystems erneuert werden, und nicht alle Komponenten von der Software-Änderung betroffen sind. Für die Überprüfung von Fehler-Korrekturen genügen oft simulierte Umweltbedingungen, da die Auswirkungen des jeweils betroffenen Moduls mit ziemlicher Sicherheit vorhergesagt werden können.

5.4.1.6 Kundenabnahme

Mit der Abnahme durch den Kunden geht die Betreuung der Software auf den Kunden oder/und die Service-Einrichtungen über. Kommt es zu einer Verweigerung der Abnahme, hat dies zumeist zwei Gründe:

- der Kunde hat von der Software andere Erwartungen
- die Stabilität der Software ist unzureichend

Der häufigste Grund bei Großprojekten ist die zu geringe Stabilität. Die Ursache hierfür liegt meist in verkürzten Testphasen und verbleibenden Restfehlern, die aufgrund von Terminnöten zustande kamen. Aus diesem Grund ist eine kontinuierliche Fortschrittskontrolle, an der auch der Kunde Interesse haben sollte, unentbehrlich. Die Ausübung von Druck auf die Einhaltung der Termine geht in der Praxis stets zu Lasten der Stabilität, sollte keine Streichung der Leistungsmerkmale möglich sein.

Nach der Abnahme durch den Kunden sind die Aufgaben der Entwicklungsabteilung hinsichtlich der jeweiligen Produkt(version) beendet (mit Ausnahme von Aktivitäten zur Unterstützung des Service).

5.4.2 Marketing-Prozess

Der Marketing-Prozess kann ebenso in Phasen unterteilt werden (vgl. folgende Abschnitte) und läuft – so weit möglich – parallel zur Produkt-Entwicklung ab. Ziel ist die Information aller zentralen und lokalen Vertriebs- und Serviceeinheiten über die Vermarktbarkeit eines neuen Produkts. Für einmalige Projekte ist er nicht relevant.

5.4.2.1 Zusammenstellung der Marketing-Grunddaten

Schon zum Zeitpunkt des Beginns der Analyse im Entwicklungs-Prozess startet der Marketing-Prozess mit der Zusammenstellung vermarktbarer Pakete. Dies erfolgt in Absprache mit der Entwicklung, die gegebenenfalls die Systemgestaltung anpassen muss. Weiters werden Pläne zur Markteinführung und Ausphasung gemeinsam mit der Entwicklung abgesprochen. Dies ist vor allem in Hinblick auf die Substitutions- und Update-Lösungen unbedingt erforderlich. Am besten werden hierzu gemeinsam mit dem Service Pilotkunden identifiziert, die als Schlüsselkunden oder technologische Vorreiter gelten. Für den Breiteneinsatz wird eine Marktkommunikation erarbeitet. Hierzu zählt:

- Ausgabe einer Produktinformation (z.b. Internet, Folder, u.a.)
- Vorbereitung von Messe-Präsentationen
- Vorbereitung von Kunden-Präsentationen
- Erarbeitung von Schulungskonzepten für Vertriebe und Kunden

In der Praxis der Software-Industrie übernimmt die Marketing-Abteilung auch die wichtige Aufgabe der Kundenbetreuung im Falle der nicht selten vorkommenden Terminverzögerungen. Mit Fertigstellung oben genannter Tätigkeiten beginnt die Akquisition.

5.4.2.2 Vorbereitung von Angeboten

Auf Basis der im vorigen Schritt erarbeiteten Daten werden die ersten Akquisitions-Aktivitäten gestartet. Hierzu zählen Präsentationen, Messeauftritte und Kunden-Gespräche. Die lokalen Vertriebsorganisationen liefern die ersten Kundenfeedbacks. Zu diesem Zeitpunkt werden auch die ersten Trials vereinbart, um eine rasche Verbreitung des Produkts zu erreichen (ein Trial ist ein kostenloser Probebetrieb der neuen Software inklusive aller Service-Aktivitäten). Am Ende dieser Phase sollte die Entwicklung so weit fortgeschritten sein, um dem Kunden verlässliche Informationen geben zu können, auf deren Grundlage dann Angebote abgegeben werden.

5.4.2.3 Einrichten des Logistik-Prozesses

Ab dem Zeitpunkt des Vertragsabschlusses muss der Bestellvorgang mit all seinen nachgelagerten Teilaufgaben sicher und rasch abgewickelt werden. Dazu müssen zum Beispiel alle Pakete mit Artikelnummern versorgt und in einem Computersystem zum allgemeinen Zugriff durch den Vertrieb eingespeichert sein. Zudem muss auch der ordnungsgemäße Einkauf von OEM-Produkten und die Abwicklung des Zahlungsverkehrs eingerichtet sein.

Mit Aufnahme der Bestellungen ist der Marketing-Prozess durchlaufen. Die weiteren Kunden-Aktivitäten sind durch die Vertriebs-Prozesse abgedeckt.

5.4.3 Serviceeinführungs-Prozess

Unabhängig vom Tagesgeschäft muss im Fall des Vertriebs eines neuen Produkts der Service Vorarbeiten leisten, um, beginnend mit der Installation beim Kunden bis zum Ersatz durch ein anderes Produkt bzw. Version, vorbereitet zu sein. Hierzu zählt in einem ersten Schritt das Kennenlernen des neuen Produkts. Durch Schulungsmaßnahmen werden die Kenntnisse an die diversen Service-Zentren weitergegeben.

Vielfach existiert eine hierarchische Kompetenzverteilung, sodass kleinere Aufgaben von einer Service-Organisation in der unmittelbaren geographischen Nähe des Kunden erbracht werden können. Dadurch werden auch kulturelle Hürden abgebaut, auf die im Falle eines zentralen Service nur schwer Rücksicht genommen werden kann.

Von besonderer Bedeutung sind im Zusammenhang mit dem Produktangebot die Service-Angebote. Diese werden zusammen mit dem Produkt vertrieben und sind als Teil des Komplettangebots zu kalkulieren. Unter Umständen können durch leicht erhöhte Servicekosten kostenintensive Tests ausgeglichen und hierbei noch die Kundenbindung erhöht werden.

5.5 Original Equipment Manufacturer - Prozess (OEMP)

Unter einem „OEM-Produkt“ (= Original Equipment Manufacturer - Produkt) bzw. „COTS-Produkt“ (= Commercial off-the-shelf - Produkt) versteht man in der Software-Industrie ein Produkt, welches nicht von der eigenen Organisation entwickelt wird. Die Gründe für ein Outsourcing von Software-Entwicklungsaufgaben sind vielfältig.

5.5.1 Entscheidungsgrundlage für den Einsatz von OEM-Produkten

Durch den Zukauf von Software sollen grundsätzlich 3 Vorteile genutzt werden:

- Zum ersten wird das Risiko gegenüber einer Eigenentwicklung stark reduziert (es verbleibt nach wie vor das Risiko mangelnder Funktionalität, insbesondere hinsichtlich der Anforderungen),
- zum zweiten werden durch den Economies-of-Scale-Effekt die Kosten gesenkt, und

- zum dritten kann im Idealfall die Entwicklungszeit auf Null reduziert werden (d.h., dass das Produkt bereits existiert).

Ausgangspunkt ist wie im Fall der Eigenentwicklung das Vorliegen der Anforderungen. Dabei können nach der Evaluierung potentieller OEM-Produkte vier (sich nicht ausschließende) Situationen auftreten [4]):

1. die OEM-Software entspricht den Anforderungen zu 100% (effizienteste Variante)
2. die OEM-Software-Funktionalität muss abgeändert werden (erfordert wegen einer dauerhaften engen Kooperation mit dem OEM-Lieferanten eine genauere Risiko- und Kostenbetrachtung über die Lebenszyklen der Software-Versionen)
3. die OEM-Software muss hinsichtlich zusätzlicher Anforderungen erweitert werden (siehe hierzu Anmerkung zu Punkt 2.)
4. die OEM-Software besitzt Funktionen die über die Anforderungen hinausgehen (erfordert die Evaluierung ungewünschter Nebeneffekte, wie z.b. eine überdimensionale Bedienungskomplexität oder erhöhte Anforderungen an die Hardware)

In jedem Fall müssen für eine Evaluierung von OEM-Software folgende Parameter berücksichtigt werden, die in Gegenüberstellung mit einer Eigenentwicklung die Vorteile des Zukaufs verdeutlichen sollen:

- **Kosten**: Eine Eigenentwicklung führt u.U. zu höheren Entwicklungskosten (gegenüber einer Nutzung von OEM-Software), da das Know-how innerhalb der eigenen Organisation nicht in ausreichendem Maße vorhanden ist und/oder die für den Eigengebrauch notwendigen Stückzahlen die Eigenentwicklung nicht rechtfertigen. Abgesehen von den eigenen Entwicklungskosten sind auch die Folgekosten (z.b. für Fehlerkorrekturen, Erweiterungen, usw.) in beiden Fällen (Eigenentwicklung und OEM-Software-Zukauf) zu berücksichtigen. Im Falle des beabsichtigten OEM-Software-Zukaufs sind auch die Evaluierungskosten von OEM-Software zu berücksichtigen.

- Für manche Kunden muss der Zulieferer bestimmte **Zertifikate** vorweisen (z.b. ISO 9000, CMMI Level x, u.a.).

- Nutzung von **Marktpositionierungen** – z.b. durch:
  - Imagegewinn des Endprodukts durch den Zukauf und die Integration von OEM-Software mit hoher Marktakzeptanz.
  - Durch die Integration von Software mit hohem Marktanteil wird ein höherer Grad an Kompatibilität zur restlichen eigenentwickelten Software bzw. zu anderer OEM-Software erreicht (dadurch begründet sich u.a. die starke Marktposition der ORACLE®-Datenbank oder des MICROSOFT®-Windows®-Betriebssystems.

[4] Holmes, Lori A., Evaluating COTS Using Function Fit Analysis, in: CrossTalk, Vol. 13 No. 2, Febr. 2000, p. 26-30

Diese Produkte haben den Status eine Quasi-Standards erreicht, wodurch eigenentwickelte Software, die mit alternativer OEM-Software integriert wird (unabhängig von der Funktionalität) Nachteile hinsichtlich Marketing hätte).

Sehr aufschlussreich ist diesbezüglich eine Analyse des Kundenstocks der OEM-Software-Zulieferer.

- Minimierung des **Investitions-Risikos**, wodurch auch ohne großen Kapitaleinsatz umfangreiche Software entwickelbar wird.

- Der Zukauf von OEM-Software ermöglicht größere Flexibilität und kürzeres **Time-to-Marketing**. *Damit rückt die Software-Entwicklung näher zum Kunden. Seine* Anforderungen können in kürzerer Zeit realisiert werden.

- Durch den OEM-Software-Zukauf wird die **Personal-Thematik** vom Software-Produkt getrennt. Diese Entkopplung bezieht sich insbesondere auf Kapazitätsanpassungen einerseits und dem Arbeitsrecht andererseits.

- Die Gestaltung der Verteilung zwischen Eigenentwicklung und OEM-Software-Zukauf hat in technischer Hinsicht wesentlichen Einfluss auf die **Verteilung des Know-hows**. In diesem Zusammenhang sollte auch die Verflechtung des Zulieferers mit Konkurrenten beachtet werden. Aufgrund von langfristigen Kooperationen (z.b. bedingt durch Wartungs- und Upgrade-Verträge) werden dem OEM-Software-Zulieferer auch die eigenen Strategien, Stärken und Schwächen bekannt gegeben, welche Firmengeheimnisse darstellen. Oft erhält man aus der Studie über die Historie des OEM-Lieferanten Informationen zu dessen Firmenverflechtungen (sehr hilfreich sind diesbezüglich Internetrecherchen; z.b. über die Suchmaschine Google®).

- Ein (Wieder-)Aufbau von Know-how ist abgesehen von langfristigen Strategien meist nur durch Firmenzukäufe möglich. **Mergers & Acquisitions** spielen daher in der Software-Branche eine große Rolle. Es lässt sich leicht beobachten wie ein und dasselbe Software-Produkt (und mit ihm die Entwickler) alle paar Jahre den Eigentümer wechselt. Manchmal wird dadurch auch ein Konkurrenzprodukt liquidiert, falls die Kosten der M&A-Transaktion die bessere Marktpositionierung und Kostenreduktionen rechtfertigen. Das darin verborgene Risiko ist vor allem in der Schwierigkeit begründet ähnliche Software zu einem neuen einheitlichen Produkt zu integrieren. Andererseits birgt eine zu rasche Abkündigung (abgesehen von der Pflicht zur Einhaltung bestehender Verträgen) das Risiko, dass Kunden einen neuen Evaluierungsprozess starten und somit der Marktanteil nicht zwangsläufig steigen muss. Umgekehrt lassen sich bei Weiterführung von zwei unterschiedlichen Produkten oder Produktfamilien kaum Synergien in den Entwicklungsabteilungen realisieren. Die potentiellen Synergieeffekte hinsichtlich Logistik und Vertrieb sind durch die modernen weltumspannenden Kommunikationsdienste ebenfalls bescheiden, da sich die meisten

kleinen Software-Firmen nur wenige Repräsentanzen weltweit leisten – und dann vor allem in der Nähe der gerade wichtigsten Industrie-Kunden, oder im Fall eines geographisch eher gleichverteilten Absatzes (im unteren Preissegment) überhaupt nur eine zentrale Informations-, Vertriebs- und Service-Niederlassung.

- Die langfristige Gestaltung der **Zusammenarbeit mit dem OEM-Software-Lieferanten** wird durch eigene vertragliche Zusagen gegenüber dem Kunden (z.b. in Wartungs- und Upgrade-Verträgen) zur Notwendigkeit. Dieser Aspekt spielt bei Nutzung von gratis zur Verfügung gestellter Software eine besondere Rolle. Sind die rechtlichen Details geklärt ist die Integration derartiger Software nur dann von Vorteil, wenn die eigene Entwicklungsabteilung im Besitz des Sourcecodes ist und diesen versteht (ein Reverse-Engineering ist in fast allen Fällen teurer als eine Software-Eigenentwicklung). Eine Analyse der Kapitalstruktur gibt Aufschluss über die finanzielle Stabilität, die sich vor allem bei kleineren Software-Firmen rasch verändern kann.

- Risken beim Ausfall des Lieferanten (durch die Auswahl möglichst standardisierter Produkte kann ein **Zweitlieferant** mit geringerem Aufwand identifiziert werden).

- Für ein weltweit zu vermarktendes Software-Produkt sind die **Export-Beschränkungen** (und damit Absatz-Beschränkungen) in einige bestimmte Länder zu beachten. Ohne auf Abstufungen und Details einzugehen ist derzeit (Stand: April 2004) der Export von Hochtechnologie in folgende Länder stark eingeschränkt: Kuba, Iran, Irak, Libyen, Nordkorea, Syrien, Sudan. Grundlage hierfür sind vor allem US-amerikanische Bestimmungen gegen „Schurkenstaaten". Ein Verstoß wird strafrechtlich und mit außerordentlich hohen Bußgeldern (u.a. auch der Firmen-Liquidation) geahndet. Da oftmals Produkte mit ähnlicher Funktionalität mit Herkunftsland USA strengeren Ausfuhrbeschränkungen unterliegen als andere ist deren Einsatz als OEM-Software im Endprodukt kritisch zu hinterfragen.

- Der OEM-Zulieferer darf nicht aufgrund eines Vergehens auf einer **„Black-List"** vermerkt sein, wobei sowohl Firmen-internen als auch -externe Listen zu beachten sind (z.b. aufgrund von Exportkontroll-Vergehen, u.a.).

5.5.2 Phasenabschnitte

Ausgangspunkt dieses Prozesses ist der CIP, bzw. die Beschlussfassung zur Realisierung eines bestimmten Produkts (vgl. hierzu Abb. 5.1). Je nach Integrationsgrad der OEM-Software in die eigene Software wird einzelnen Phasen-Abschnitten eine unterschiedlich hohe Bedeutung beigemessen. Die Erfordernisse an den Software-Prozess (OEMP) variieren hierbei mit folgenden Stufen:

- OEM-Software wird ohne Integration weitervermarktet,
- OEM-Software ist auf triviale Weise Teil eines Gesamtsystems,
- OEM-Software wird in ein Gesamtsystem tiefgehend integriert,
- OEM-Software wird zu einem sehr frühen Zeitpunkt integriert, wobei der eigenständige Charakter des OEM-Produkts verloren geht und das Gesamtsystem ab dem Zeitpunkt der Realisierung als Eigenentwicklung betrachtet werden muss

Dabei ist grundsätzlich auch die Möglichkeit in Betracht zu ziehen, dass abgesehen von Änderungen bzw. Erweiterungen der Funktionalität der zugekauften Software auch Änderungen in den Anforderungen möglich sind. Z.b kann es gegebenenfalls günstiger sein Geschäftsprozesse an die Software anzupassen als umgekehrt.

Die Phasenabschnitte beim Einsatz von OEM-Software sind im Detail:

5.5.2.1 OEM-Produktauswahl

Zu Beginn der Auswahl-Phase wird auf firmeninterne Quellen zurückgegriffen. Oft sind die in Frage kommenden Produkte schon bekannt, sei es durch vergangene Projekte oder von Fachmessen. Danach erfolgt auf Basis von Non-Disclosure-Agreements ein intensiver Austausch der Produktinformationen. Hierbei handelt es sich um gegenseitige Verschwiegenheits-Verpflichtungen gegenüber Dritten.

Als Ergänzung zu den funktionalen (technischen und qualitativen) Anforderungen sind folgende Punkte vertraglich zu regeln (siehe hierzu auch die Ausführungen am Beginn dieses Abschnittes zum Thema: Entscheidungsgrundlage für den Einsatz von OEM-Produkten):

- Preise bzw. Kosten, die sich im Detail aus folgenden Positionen zusammensetzen können (bei Software sind der hohe jährliche Preisverfall und die typischerweise großen Differenzen zwischen Listenpreis und schlussendlich verhandeltem Preis zu beachten [aus Erfahrung oftmals bis zu einem Verhältnis von 10:1]):
    - Lizenzen für ein Entwicklungstool, die die Integration der OEM-Software in die eigene Software unterstützen
    - Laufzeit-Lizenzen, die für den Betrieb der Software (beim Kunden) notwendig sind
    - Kosten für den Bezug des Sourcecodes (ohne Zugang zum Sourcecode sind Änderungen an der OEM-Software unmöglich)
    - Kosten für Updates (kleinere Ergänzungen) und Upgrades (größere Ergänzungen)
    - Wartungskosten zur Laufzeit (beim Kunden)

- Kosten, die beim OEM-Lieferanten und in der eigenen Organisation durch Adaptionen bzw. die Integration der OEM-Software anfallen
- Zahlungsziel (im Idealfall unter 30 Tagen; zumindest aber unter 90 Tagen)

- Roadmap der OEM-Software-Lieferungen und Weiterentwicklungs-Verpflichtung der OEM-Software. Hierbei werden auch die Prozesse definiert, die durch die Integration der OEM-Software mit der eigenen Software für Upgrades bzw. Updates nötig sind.

- Antwortzeiten beim Auftreten von Mängeln (diese müssen kompatibel zu den eigenen Kundenverträgen sein). Ergänzend sollte ein Prozess vereinbart werden der die Schritte definiert falls ein Fehler während des Systemtests oder im Betrieb beim Kunden nicht eindeutig der OEM-Software oder eigenen zugewiesen werden kann

- Prozess beim Eintreten von Änderungswünschen („Change Requests") während der Software-Entwicklung

- Insbesondere ist bei Adaptionsarbeiten seitens des OEM-Lieferanten die Durchführung der Tests zu spezifizieren. Im Industriesektor kann die Testumgebung beim Lieferanten meist ohne erhebliche finanzielle Investitionen nicht nachgebildet werden. Selbst bei kleineren Adaptionen kann die dem Software-Test zugrundeliegende Hardware einen beachtlichen Anteil der Adaptionskosten ausmachen, wodurch sie u.U. vom Auftraggeber bereitgestellt werden muss.

- Juristische Details, wie:
  - Gestaltung von Gewährleistungen und Garantien
  - Patentrechtliche Absicherungen
  - Gerichtsstand (abgesehen von der geografischen Entfernung kann die Anwendung von anderem Recht zu gravierenden Nachteilen führen)
  - Exportkontroll-Kennzeichnung (eine nachträglich vom Lieferanten eingeforderte Bekanntgabe der Kennzeichen kann u.U. zeitaufwendig sein)
  - Klauseln zur Sourcecode-Hinterlegung im Falle des Bankrotts des OEM-Software-Zulieferers
  - Klauseln zur Vertragskündigung

- Definition der Abnahmekriterien (insbesondere von Leistungsaussagen bei Volllast)

Je nach finanziellem Volumen wird der Vertrag implizit durch Bestellung oder über ein eigens aufgesetztes Schriftstück gestaltet. Dieses wird meist in einen Hauptteil, einen Zusatz über Details zu bestimmen Software-Produkten und –Versionen (Vertragsgegenstand) und in einen Teil betreffend Service-Leistungen (z.b. Wartung) gegliedert.

5.5.2.2 Projektplanung, Integration, Service, Ausphasung

In diesem Schritt stellt der OEM-Lieferant meist einen Prototyp inklusive Dokumentation und allfälliger Lizenzen bereit, der ersten Tests und Integrationsversuchen unterzogen wird. Parallel dazu werden die Preis-/Leistungsverhandlungen zu einem Ergebnis gebracht, wobei aus folgenden Tätigkeiten weitere Vertragsdetails abgeleitet werden:

- Stückzahlplanungen der OEM-Software
- Leistungsmerkmal-Spezifikation des Komplettsystems
- Erstellung eines Projekthandbuchs
- Erstellung eines Offline-Testreports der OEM-Software
- Erstellung der Spezifikationen für den Online-Test (Gesamtsystemtest)
- Erstellung eines Service-Konzepts
- Erstellung eines Trainings-Konzepts
- Bereitstellung der Konfigurations- und Upgrade-Information
- Zurverfügungstellung der notwendigen Produktzertifikate

Am Ende dieser Phase ist das unter Verwendung von OEM-Software gebundene Gesamtsystem fertig integriert und getestet und zur Installation beim ersten Kunden bereit. Des weiteren sind die logistischen Prozesse etabliert, die eine Vermarktung in größerem Rahmen gestatten (sofern es sich nicht um ein einmaliges Projekt handelt). Mit der Abnahme durch den Kunden übernimmt die Service-Organisation die Betreuung des Systems und wartet es bis zum Zeitpunkt der Außerdienststellung.

Die in dem Projekt gewonnenen Informationen werden in einer firmeninternen Datenbank abgelegt. Ein wichtiger Punkt, der ebenfalls zum Zeitpunkt der ersten Kundenabnahme geklärt sein muss, ist die Ausphasungs- bzw. Upgrade- und Update-Strategie. Upgrades bezeichnen größere, Updates kleinere Funktionshübe.

Aufgrund von geänderten Kundenanforderungen oder der schrittweisen Einschränkung von Service-Leistungen muss Software erneuert oder überhaupt außer Betrieb gesetzt werden. Hierbei ist Rücksicht auf die Systemumgebung zu nehmen.

6. Software-System-Modellierung im Entwicklungs-Prozess

Die Software-System-Modellierung im Entwicklungs-Prozess ist in zweidimensionaler Ausprägung zu betrachten. Einerseits hinsichtlich des Prozesses und andererseits hinsichtlich des Inhalts. Während die Prozess-Thematik schon seit Anfang der 70' Jahre wissenschaftlich erörtert wird, entstanden die ersten Arbeiten zur graphischen Darstellung des Inhalts von Software (vgl. Unified Modeling Language (= UML)) erst im Zusammenhang mit der kommerziellen Nutzung objektorientierter Sprachen in den 90' Jahren. Die Visualisierungstechniken zur Formulierung des Inhalts richten sich sowohl an den Anwender als auch an den Software-Entwickler und erleichtern gegenüber der Beschreibung in ausschließlich schriftlicher Form das Verständnis für die Funktionsweise der Software. Dies ist die wichtigste Voraussetzung zur Entwicklung der „richtigen", d.h. gewünschten Software.

6.1 Prozess-Modelle

Grundbaustein der Ablauf-Organisation von Entwicklungs-Abteilungen innerhalb eines Softwarehauses sind deren Software-Erstellungs-Prozesse. Darüber hinaus sind definierte Prozesse die Voraussetzung für das Erlangen von Qualitäts-Zertifikaten (z.b. ISO900x).

Die im folgenden diskutierten Entwicklungs-Prozesse spiegeln den allgemeinen Fortschritt in der Software-Technologie wider, der u.a. durch immer umfangreichere Softwarepakete und neue Programmierkonzepte (z.b. Objektorientierung) gekennzeichnet ist. Aus einer anderen Perspektive betrachtet behalten die zentralen Aussagen der unterschiedlichen Modelle weiterhin ihre Gültigkeit – abhängig von den konkreten Projekt-Gegebenheiten.

Zu Beginn der Entwicklung von Software (in Maschinen- und Assemblercode) galt lediglich das Prinzip „code & fix" – zu Deutsch „Codiere und behebe die Fehler". Aus diesen Anfängen haben sich mittlerweile komplexe und umfassende Prozesse inklusive dazugehörender Tools entwickelt – wie z.b. jener der Firma Rational [1]), bzw. mittlerweile IBM (im Dezember 2003 übernahm IBM die Firma Rational Software Corp. für US$ 2,1 Milliarden [2])). Über diesen Link erhält man Zugang zu einer Reihe von Dokumenten im Zusammenhang mit dem Rational Unified Process, welcher auf den Überlegungen zur UML aufbaut.

---

[1] http://www-306.ibm.com/software/rational/info/literature/lifecycle.jsp , IBM's Rational Unified Process, UML, 2004

[2] http://www.butlergroup.com , IBM's Übernahme der Firma Rational Software Corp., 2004

6.1.1 Wasserfall-Modell

Eines der ersten Modelle, das weiterhin Gültigkeit besitzt, ist das Wasserfall-Modell aus dem Jahre 1970 [3]). Es unterteilt die Arbeitspakete der Software-Entwicklung in 7 Phasen, die nacheinander durchlaufen werden. Diese sind:

1. Identifikation der System-Anforderungen
2. Identifikation der Software-Anforderungen
3. Analyse
4. Entwurf
5. Codierung
6. Test
7. Betrieb

Nach Beginn eines neuen Phasenabschnitts ist lediglich die Rückkehr zur vorhergehenden Phase vorgesehen. Weitere Charakteristika sind:

- Das Modell unterstützt die Einführung eines disziplinierten und kontrollierbaren Entwicklungs-Fortschritts.
- Die Aktivitäten jeder Phase müssen abgeschlossen sein, bevor mit der nächsten begonnen wird.
- Das Ende jeder Phase wird durch die Fertigstellung eines Dokuments charakterisiert, wodurch der Eindruck einer Dokument-getriebenen Entwicklung entsteht.
- Der Auftraggeber wird nur zu Beginn in die Entwicklung miteinbezogen.
- Durch die klare Top-Down-Struktur erfordert dieses Modell geringen Managementaufwand.

Typische Anwendungen sind:

- Projekte mit klaren Anforderungen
- Projekte kurzer bis mittlerer Laufzeit

Um die Risken auf ein akzeptables Niveau zu begrenzen sind folgende Einschränkungen bei Anwendung des Wasserfall-Modells zu berücksichtigen:

- Die Anforderungen sollen zu Beginn der Implementierungs-Phase bekannt sein.
- Die Anforderungen beinhalten keine Risken hinsichtlich:
    - OEM-Komponenten
    - Kosten
    - Terminplänen
    - Performance (insbesondere Mengengerüstprobleme und System-Antwortzeiten)

[3] Royce, W.W., Managing the development of large software systems, in: IEEE, 1970, p. 1-9

- Sicherheit
- GUIs (= Graphical User Interfaces)
- organisatorische Risken

- Die Natur der Anforderungen ändert sich nur geringfügig während der Entwicklungszeit.
- Die Anforderungen basieren auf einer Übereinstimmung der Erwartungen von Benutzern, Kunden, Entwicklern, Wartungspersonal und Investoren.
- Die Architektur ist bekannt.
- Für die sequentielle Entwicklung steht genügend Zeit zur Verfügung.

Wird eines der Kriterien nicht erfüllt, handelt es sich nicht mehr um eine Entwicklung nach dem Wasserfall-Modell im eigentlichen Sinn.

6.1.2 V-Modell

Ein auf dem Wasserfall-Modell basierendes Entwicklungs-Modell ist das V-Modell. Besondere Aufmerksamkeit wird auf den Software-Test gelegt, sodass die Qualitätssicherung wichtiger Bestandteil des Prozesses wird. Hierdurch wird sowohl die Validation als auch die Verifikation der Teilprodukte sichergestellt [4]). Da dieses Modell seit 1992 die Grundlage für die Software-Entwicklung sowohl in der deutschen Bundesverwaltung als auch bei der Bundeswehr ist, existieren hierzu umfassende Ausarbeitungen [5]). Ziel ist eine Standardisierung und die damit einhergehende Vergleichbarkeit der Projekte.

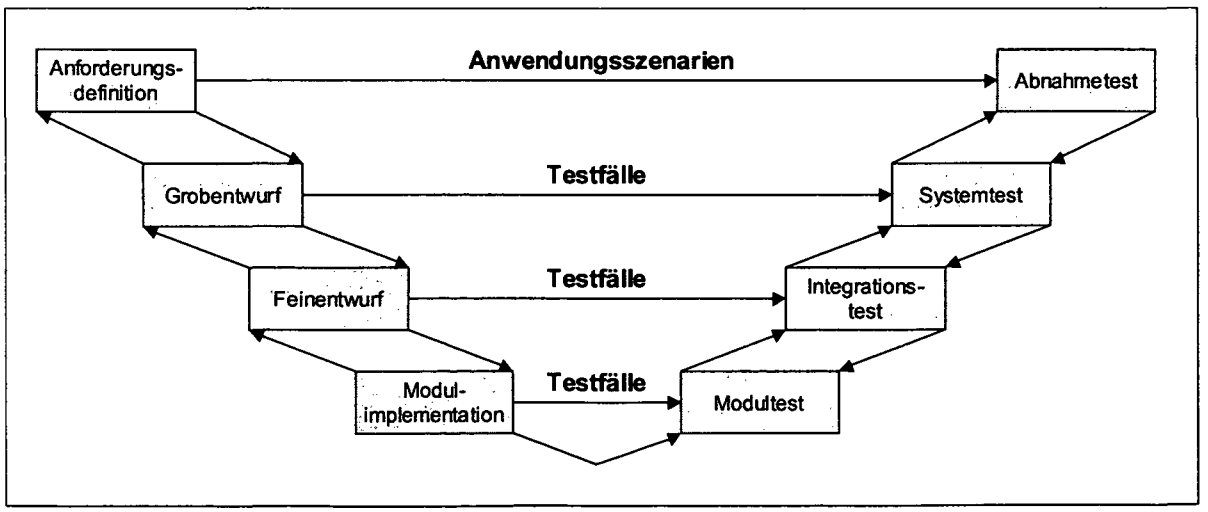


Abb. 6.1: V-Modell

[4] Boehm, B.W., Verifying and Validating Software Requirements and Design Specifications, in: IEEE Software, 1984, p. 75-88

[5] Bröhl, A.-P., Dröschel, W. (hrsg.), Das V-Modell, München, 1993

In Abb. 6.1 werden die Zusammenhänge zwischen den Tests (Qualitätssicherungs-Maßnahmen) und den ihnen zugrunde liegenden Meilenstein-Ergebnissen (z.b. Feinentwurf) veranschaulicht. Es zeigt zudem den Zeitpunkt bzw. Meilenstein, zu dem die Testfälle erstellt werden können (jene des Abnahmetests werden z.b. schon nach Fertigstellung der Anforderungsdefinition erstellt).

Im Gegensatz zum Wasserfall-Modell sind die Systemgrenzen relativ weiträumig gezogen. Als System wird ein IT-System, bestehend aus Software und Hardware, bezeichnet. Die Basisbausteine des V-Modells sind **Produkte** und **Aktivitäten**. Unter einem Produkt wird (im V-Modell) das Ergebnis bzw. der Bearbeitungs-Gegenstand einer Aktivität verstanden. Die Aktivität beschreibt eine Tätigkeit mit dem Ziel ein Produkt zu erstellen oder den Zustand bzw. den Inhalt eines Produkts zu ändern. Folglich dokumentiert eine Aktivitätenbeschreibung die Arbeitsanleitung und eine Produktbeschreibung den Inhalt des Produkts.

Nicht-IT-Systeme können ebenfalls Teil der Betrachtungen innerhalb des V-Modells sein. Typisches Beispiel hierfür ist die Organisationsentwicklung, die in enger Relation zur IT-Planung erfolgen sollte. Die sinnvolle Ausstattung von Arbeitsplätzen mit Terminals, Druckern, Netzsoftware, aber vor allem Anwendungssoftware, kann nur in Übereinstimmung mit einer entsprechenden Aufbau- und Ablauforganisation erfolgen.

Um den Zustand eines Subsystems zu kennzeichnen ist im V-Modell ein Zustandsübergangs-Diagramm mit 4 Zuständen definiert (vgl. Abb. 6.2). Im Zustand „geplant" sind die Subsystem-Grenzen des Produkts noch unvollständig definiert.

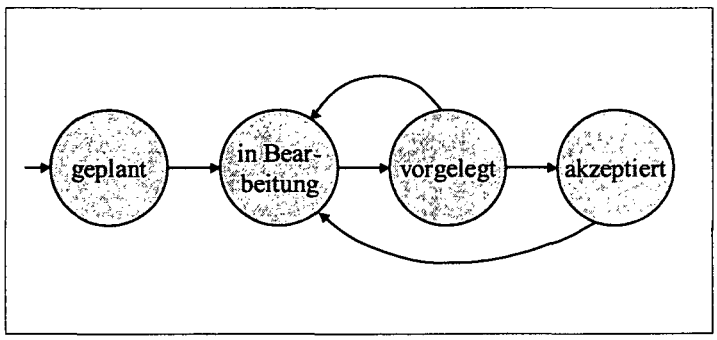


Abb. 6.2: Zustandsübergangs-Diagramm

Im Zustand „in Bearbeitung" befindet sich das Produkt noch in der privaten Arbeitsumgebung eines Entwicklers. Nach Fertigstellung wird das Produkt dem Konfigurations-Management übergegeben und einer Qualitäts-Kontrolle unterzogen. Nach bestandenem Test wird das Produkt schließlich als akzeptiert abgelegt und in weiterer Folge in die übrige Systemlandschaft integriert.

Ein Charakteristikum komplexer Modelle ist die Einarbeitung von **Rollen**. Diese stehen stellvertretend für die notwendigen Erfahrungen, Kenntnisse und Fähigkeiten um bestimmte Aktivitäten durchführen zu können. Im V-Modell werden zunächst die Aktivitäten (und damit die Produkte) auf 4 Submodelle verteilt. Diese sind:

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

Für jedes dieser Submodelle sind Anforderungsprofile vordefiniert, und zwar für

- einen Manager, der die Rahmenbedingungen des Submodells festlegt und oberste Entscheidungsinstanz ist,
- einen Verantwortlichen, der die Aktivitäten des Submodells plant, steuert und kontrolliert, und
- Durchführende, die die Aktivitäten ausführen.

Des weiteren sind für jedes Subsystem auch die Teil-Produkte, wie z.b. Systemarchitektur, Schnittstellenübersicht, usw. vordefiniert. Der Anspruch des V-Modells auf Allgemeingültigkeit legt den Nachteil der Unübersichtlichkeit nahe. Diesem Umstand Rechnung tragend kann durch ein „Tailoring" die Vielfalt an Produkten und Aktivitäten eingeschränkt werden, sodass das Modell auch auf kleinere Projekte anwendbar ist. Dieses Maßschneidern des Modells erfolgt in 2 Stufen - dem Vortailoring (vor Entwicklungsbeginn) und dem Technischen Tailoring (vor Beginn jeder Hauptaktivität während des Entwicklungs-Prozesses).

Das Vortailoring wird durch vorgefertigte Matrizen erleichtert, wobei für typische Anwendungen pro Submodell die notwendigen Aktivitäten, Produkte und Rollen genannt werden. Auf Rollenebene wird durch eine Einteilung in „verantwortlich", „mitwirkend" und „beratend" das organisatorische Zusammenwirken auch zwischen den Rollen definiert.

Zusammengefasst stehen den unübersehbaren Vorteilen der bis ins Detail ausgearbeiteten Submodelle trotz aller Bemühungen des Tailorings die Nachteile der unnötigen Aktivitäten-, Produkt- und Rollenvielfalt bei kleinen Projekten gegenüber. So schlägt z.b. das Submodell SE für ein Projekt mit einem Aufwand von 6 MannMonaten 11 Teilprodukte und 18 Aktivitäten vor [6]). Somit ist dieses Modell der Anwendung in mittleren und großen Projekten vorbehalten.

6.1.3 Modell-Philosophien

Zusätzlich zu den in den vorigen und nachfolgenden Abschnitten erörterten Modellen findet man in der Literatur noch weitere, die einiger Bemerkungen bedürfen:

- Prototyping-Modell

[6] Balzert, Helmut, Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Heidelberg, 1998

- Evolutions-Modell
- Inkrementelles Modell
- Objektorientiertes Modell

Sie sind dadurch gekennzeichnet, dass sie jeweils eine charakteristische Eigenschaft in den Mittelpunkt stellen und die weiteren Details undefiniert lassen. Somit sind sie nur in Kombination mit einem anderen Modell sinnvoll einsetzbar. Dies ist der Grund, warum sie eher als Philosophien, denn als Modelle zu bezeichnen sind.

6.1.3.1 Prototyping-Modell

Dieses Modell resultiert aus der Erkenntnis, dass eine frühzeitige vollständige Definition der Anforderungen in den meisten Software-Projekten unmöglich ist. Gründe dafür sind, dass

- der Auftraggeber selbst über seine Anforderungen nur unvollständig informiert ist,
- sich während der Entwicklung die Anforderungen ändern bzw. konkretisieren,
- erst während der Entwicklung Lösungsstrategien beurteilt werden können (Performance-Anforderungen), oder aber
- der Vertrieb bereits in der Akquisitionsphase vorführbare „Software-Muster" benötigt.

Um diesen unterschiedlichen Anforderungen an einen Software-Prototyp gerecht zu werden, sind folgende Typen zu unterscheiden (siehe auch [7]):

- Labor-Prototyp:
  Dieser dient der Evaluierung technischer Aspekte. Er realisiert nur die zur Diskussion stehenden Eigenschaften und dient nicht der Modellierung des Systems als Ganzes.

- Prototyp i.e.S.:
  Soll hingegen der Anwendungsbereich analysiert werden, liegt der Fokus auf der Benutzer-Schnittstelle (GUI) und der Funktionalität der zu realisierenden Software. Die optimale Gestaltung der Ergonomie ist zumeist nur anhand eines Prototyps beurteilbar.

- Pilotsystem:
  Im Gegensatz zu den anderen Prototypen ist für ein Pilotsystem die Anwendung eines definierten Entwicklungs-Prozesses notwendig, da dieses bereits die erste Version des gewünschten Systems vorweg nimmt. Es muss infolgedessen so konstruiert sein, dass eine evolutionäre oder inkrementelle Weiterentwicklung möglich ist.

[7] Kieback, A., et al., Prototyping in industriellen Software-Produkten, in: Informatik-Spektrum 15, 1992, p. 65-77

- Demo-Prototyp:
  Diese Art von Prototyp gestattet den größten Spielraum zur Realisierung. In vielen Fällen ist nur die Benutzeroberfläche dem Original ähnlich. Screen-Shots können hierbei auf vielfältigste Weise erstellt werden. Für Demo-GUIs empfiehlt sich der Einsatz entsprechender Tools, die eine interaktive Gestaltung der GUI unterstützen. Zumeist ist die Funktionalität auf die Verarbeitung einer vorgegebenen Menge von Parametern beschränkt, oder zur Gänze nur für einen bis ins kleinste Detail vordefinierten Use-Case darstellbar.

Im konkreten Fall sollte schon vor Anfertigung des Prototyps bekannt sein, in welcher Weise der Prototyp mit dem eigentlich zu erstellenden System in Beziehung steht. Gegebenenfalls sind dann bei der Erstellung des Prototyps bereits eine Menge von Richtlinien zu berücksichtigen. Dies gilt vor allem dann, wenn Teile des Prototyps in das endgültige System übernommen werden sollen.

6.1.3.2 Evolutions-Modell

Beim Evolutions-Modell verhält es sich ähnlich wie beim Prototyping-Modell. Ohne zusätzliche Modell-Elemente lässt dieses Modell zu viele Freiheitsgrade offen, wodurch von einem Modell im eigentlichen Sinn nicht gesprochen werden kann. Kernphilosophie des evolutionären Gedankens ist die schrittweise Entwicklung eines Software-Systems, wobei jede Version beim Kunden zum Einsatz gelangt. Dieses Vorgehensmodell wird durch folgende Sachverhalte bekräftigt:

- Oft sind die Anforderungen dem Kunden zu Beginn des Software-Projekts nur unzureichend bekannt. Und mit jeder neuen Version hat der Kunde Einfluss auf die weitere Produkt-Gestaltung.

- Der Kunde erhält in kürzeren Zeitabständen einsatzfähige Produkte.

- Da mit dem Einsatz kommerzieller Software auch in Organisationsstrukturen eingegriffen wird, kann diese durch die fortlaufenden Versionen schrittweise adaptiert werden.

- Durch die kleinere Anzahl an Arbeitspaketen innerhalb einer Version ist das Projekt besser überschaubar und damit einfacher zu managen.

- Nachteilig wirkt sich der kurze Planungshorizont aus, wodurch in nachfolgenden Versionen eine Architekturüberarbeitung notwendig werden kann.

Wie aus der Charakterisierung erkennbar, kommt der Gedanke der Versionierung in vielen Produkten für den anonymen Markt zum Tragen, wo Produkte kontinuierlich durch neue Versionen verbessert werden. Da bei Neuentwicklungen die ersten Versionen nur eingeschränkt am Markt angeboten werden, spricht man in diesem Zusammenhang auch von evolutionärem Prototyping. Über detailliertere Ordnungsstrukturen werden im Zusammenhang mit der Bezeichnung „Evolutions-Modell" generell (in der Literatur) keine Aussagen gemacht.

6.1.3.3 Inkrementelles Modell

Dieses baut zur Gänze auf dem evolutionären Modell auf, jedoch mit dem Unterschied, dass größerer Aufwand in die Anforderungs-Analyse investiert wird. Dadurch wird gewährleistet, dass die von Beginn an geplante Systemarchitektur auch in den folgenden Versionen einsetzbar ist. Mit fortschreitenden Versionen werden immer mehr – von Anfang an geplante – Funktionalitäten realisiert. Zudem wird durch dieses Vorgehen auch eine Risiko-Minimierung erreicht.

6.1.3.4 Objektorientiertes Modell

Dieses Modell verkörpert eine grundlegende Sichtweise hinsichtlich der Struktur von Programm-Code. Als Weiterentwicklung des prozeduralen Ansatzes unterstützt der objektorientierte Ansatz die Bewältigung zunehmender Komplexität von Software (vgl. entsprechende Ausführungen in diesem Kapitel zur Abbildung techno-sozialer Komplexität).

Die wichtigsten Charakteristika dieses Modells, bzw. dieser Philosophie, sind die Wiederverwendbarkeit und die durchgehende Einhaltung des objektorientierten Paradigmas während des Entwicklungs-Prozesses. Durch die Unterstützung der Wiederverwendbarkeit wird zum ersten Mal auch dem Bottom-Up-Ansatz eine grundlegende Bedeutung beigemessen. Ausgehend von bereits vorhandenen Elementen bzw. zugekauften Software-Komponenten wird zunächst eine objektorientierte Analyse (OOA), anschließend ein objektorientiertes Design (OOD), und schlussendlich die objektorientierte Programmierung (OOP) durchgeführt. Durch den fehlenden Paradigmabruch im Entwicklungsprozess wird eine durchgehende Tool-Unterstützung erleichtert bzw. sogar erst ermöglicht.

Um den Vorteil der Wiederverwendbarkeit überhaupt erst zu realisieren muss entweder schon während, oder spätestens nach der Fertigstellung des Produkts, eine entsprechende Archivierung der OOA, OOD und/oder OOP Entwürfe sichergestellt werden. Dies verursacht Kosten, die bei der Wiederverwendung in Rechnung gestellt werden müssen. Andernfalls ist die Einführung einer adäquaten Archivierung eher theoretischer Natur. Eine weitergehende

Diskussion der Hilfswerkzeuge zu OOA und OOD wird im Abschnitt 6.4 vorgestellt. Dies sind bedeutende Konzepte die weit über die Software-Entwicklung hinaus Anwendung finden.

6.1.4 Extreme Programming - Modell

Ein wesentliches Unterscheidungsmerkmal von „Extreme Programming" (= XP) im Vergleich zu anderen Modellen und Philosophien liegt in der Minimierung des Risikos (hinsichtlich der Erfüllung aller Requirements: Funktionalität, Qualität, Kosten und Termine) durch möglichst viele einsatzfähige Zwischen-Versionen des Endprodukts. Zu diesem Zweck zielt das Modell auf eine optimale Teamarbeit zwischen Kunde, Manager und Entwickler ab, wodurch eine hohe Dynamik in der Berücksichtigung von Änderungswünschen erreicht wird. Das Einbringen von Requirements während der gesamten Projektlaufzeit ist in vielen Software-Projekten ein häufig anzutreffendes Charakteristikum, da der Kunde mit jedem Entwicklungsschritt eine bessere Vorstellung über das Endprodukt bekommt.

Der Ursprung von „Extreme Programming" ist auf ein Projekt bei der Firma Daimler-Chrysler (Projektbeginn war 1996) zurückzuführen, in dem Kent Beck [8]) neue Methoden der Software-Entwicklung untersuchte. Dabei ging es in erster Linie um die Anpassung der Entwicklungsprozesse an die alltägliche Praxis. Im Gegensatz zu früheren Überlegungen wurden die Defizite nicht durch genauere Planungen oder aufwendigere neue Konzepte auszugleichen versucht, sondern durch eine radikale Vereinfachung des Entwicklungsprozesses und einer Konzentration auf die wesentlichsten Erfolgsfaktoren:

- „Communication":
  Darunter wird die Optimierung der Kommunikation zwischen dem Kunden, den Managern und den Entwicklern verstanden. Hinsichtlich des Kunden wird dies durch dessen kontinuierliche Einbindung (zumeist täglich) in den Entwicklungsprozess erreicht. Zwischen den Entwicklern wird die Kommunikation u.a. durch einen übersichtlich gestalteten Sourcecode verbessert, wodurch prinzipiell jeder Entwickler in jeder Code-Passage Änderungen vornehmen kann.

- „Simplicity":
  Diese Anforderung kann treffend mit der Phrase: „Do the simplest thing that could possibly work" [9]) beschrieben werden. Die Software wird in möglichst kurzen Iterationsschritten entwickelt, wobei auf eine einfache Architektur und übersichtliche Codierung besonderer Wert gelegt wird. Da die Hardware- im Vergleich zu den Softwarekosten zumeist nur einen kleinen Bruchteil betragen (oft weniger als 10%),

[8] http://www.extremeprogramming.org/ , 2004
[9] Jeffries, Ronald E., The Four Project Values, http://www.xprogramming.com/Practices/PracValues.html , 2004

wird die Systemleistung (resultiert in verminderter Hardware-Anforderung) erst gegen Projektende hin optimiert.

- „Feedback":
  Feedback wird hinsichtlich der Validierung der Anforderungen durch die enge Einbindung des Kunden (aufgrund der kurzen Release-Zyklen wird die Software dem Kunden schrittweise präsentiert) und hinsichtlich der technischen Qualität durch umfangreich gestaltete, möglichst weitgehend automatisierte, Tests eingeholt.

- „Courage":
  Dieser Faktor adressiert eine für XP wünschenswerte persönliche Einstellung der Teammitglieder: „Courage exists within the context of the other three values. They all support each other. It takes courage to trust that concrete feedback along the way is better than trying to know everything up front. It takes courage to talk to others on the team when that might expose some of your own ignorance. It takes courage to keep the system simple, deferring tomorrow's decisions until tomorrow." [10])

Die philosophische Grundlage bildeten verschiedene Patterns (vgl. Kapitel 11), die sich in der Praxis als besonders wertvoll erwiesen um Software pünktlich und innerhalb der Budgetgrenzen zu realisieren. Die im Rahmen der XP-Philosophie angewandten Patterns bilden (zusammengesetzt) eine „leichtgewichtige Methodik", die sich durch eine geringe Anzahl an leicht anzuwendenden Regeln und Praktiken auszeichnet. Dadurch steht dieses Modell im Gegensatz zu den sich in den 80er Jahren unter dem Einfluss von TQM entwickelten umfangreichen Entwicklungsmethoden, die u.a. in komplexen CASE-Tools kumulierten und die Lernkurve immer flacher werden ließen.

Die vier Hauptentwicklungsschritte (Planung, Entwurf, Codierung, Test) sind bei Anwendung des XP-Entwicklungsmodells durch folgende Regeln und Praktiken charakterisiert:

1. **Planung:**
   - In „User Stories" beschreibt der Kunde die Funktionalität der zu erstellenden Software (ohne Rücksichtnahme auf irgendeine Syntax – im Gegensatz zu den „Use Cases" von UML – vgl. Abschnitt 6.4 in diesem Kapitel). Der Umfang der User Stories sollte so gewählt sein, dass sie in ein bis drei Wochen realisierbar sind (vgl. hierzu das Verfahren der kleinsten Arbeitspakete nach Exl im Kapitel 7). Auf den User Stories basierend werden die Abnahmetests entwickelt. Mehrere User Stories sollten innerhalb eines Iterationsschritts realisierbar sein.

[10] Miller, Roy W., http://www-106.ibm.com/developerworks/java/library/j-xp0813/?loc=dwmain , 2004

- In der Release-Planung werden die User Stories auf die einzelnen Iterationen entsprechend dem Kundenwunsch verteilt. Hierbei sollte darauf geachtet werden, dass der Kunde in regelmäßigen Abständen (zwischen ein und drei Wochen) neue Releases (nach jedem Iterationsschritt steht ein neues Release zur Verfügung) zum Zweck der Einholung von Feedback bekommt.

- Um den Aufwand auf einfache Weise abzuschätzen, sollte die Anzahl der User Stories pro Iterationsschritt und der Aufwand pro User Story konstant sein. Während eines Iterationsschritts müssen neben neuen User Stories auch Fehler aus den nach jedem Iterationsschritt durchzuführenden Abnahmetests korrigiert werden (die Fehlerkorrekturen werden wie User Stories behandelt).

- Um das Know-how innerhalb des Teams möglichst gleichmäßig zu streuen, sollten die Projekt-Mitarbeiter in Form einer Job-Rotation möglichst viele unterschiedliche Tätigkeiten im Projektverlauf durchführen. Dadurch werden die Mitarbeiter in die verschiedensten Aufgaben eingeschult und das Risiko bei Engpässen und Kündigung minimiert.

- Die vom Team zu befolgenden Regeln sind einzuhalten, jedoch sollte bei Bedarf eine Anpassung in kürzest möglicher Zeit erfolgen.

- Die Teamgröße für ein nach dem XP-Modell aufgesetztes Projekt sollte 10 Personen nicht übersteigen [11]), andernfalls besteht das Risiko, die Projektübersicht zu verlieren.

2. Entwurf:

- Die wichtigste Regel beim Software-Entwurf ist die Konzentration auf eine möglichst einfache Lösung.

- Durch Metaphern werden User Stories, Objekte und andere Bezeichner für alle Teammitglieder gleichermaßen verständlich gemacht.

- Um den Entwurf als Teamarbeit zu gestalten, können CRC-Karten (= Class-Responsibilities-Collaboration - Karten) als Hilfsmittel benutzt werden. Es handelt sich hierbei um Papierkärtchen, die an einer Pinwand geordnet befestigt werden. Voraussetzung für deren Einsatz ist die Anwendung einer objektorientierten Programmiersprache. Unter dem Begriff: Responsibilities werden die Attribute und Operationen der Klasse, unter dem Begriff: Collaboration die Beziehungen zu anderen Klassen verstanden. Die Anordnung der Kärtchen bzw. Klassen auf der Pinwand folgt entweder der statischen Semantik (d.h. den Generalisierungen,

[11] http://dev.panopticsearch.com/xp-notes.html , 2004

Spezialisierungen, Assoziationen und Aggregationen) oder der dynamischen Semantik (d.h. dem Nachrichtenfluss).

- Bei neuartigen bzw. komplexen Softwareteilen (in XP als Spikes bezeichnet) sollten durch separate Experimente (in Form von zu testenden Software-Fragmenten) Lösungen im Vorfeld gefunden werden.

- Um den Terminplan nicht zu gefährden empfiehlt sich die Realisierung von Software-Verbesserungsmaßnahmen ausschließlich zu Projektende.

- Die während des Projekts überflüssig gewordenen Code-Passagen oder Teile davon (z.b. unbenutzte Variable) sollten kontinuierlich aus dem Code entfernt werden (in XP als Refactoring bezeichnet), um ihn nicht unnütz aufzublähen und damit die Qualität negativ zu beeinflussen.

3. **Codierung:**
   - Im Gegensatz zu anderen Modellen und Philosophien sollte der Kunde im XP-Modell ständig erreichbar sein. Zu favorisieren ist eine lokale Präsenz beim Entwicklungsteam.

   - Codier-Standards gewährleisten die Lesbarkeit des Codes für jedes Team-Mitglied.

   - Zu Beginn der Codierung werden die Komponententests programmiert. Dies erleichtert die Codierung der Ziel-Software und schränkt gleichzeitig ihre Funktionalität auf jene Leistungsmerkmale ein, die von der Test-Software überprüft werden [12]): „You create one test to define some small aspect of the problem at hand. Then you create the simplest code that will make that test pass."

   - In manchen Fällen empfiehlt sich „Pair Programming", d.h. dass sich zwei Entwickler einen Computer teilen. Während der eine über Tastatur und Maus codiert, reflektiert der andere über den geschriebenen Code. Dieser Arbeitsstil („Zweier-Teams") fördert zudem das Training und steigert die Qualität der Software. Als Nebeneffekt können dadurch u.U. auch teure Lizenzkosten eingespart werden, wobei dieser Aspekt keinesfalls der Grund für diesen Arbeitsstil sein darf!

   - Durch eine in kurzen Zeitabständen stattfindende Integration der Software-Teile wird der kontinuierliche Projektfortschritt am besten dokumentiert. Um das Repository möglichst aktuell zu halten sollten die Zeitabstände, in denen die Entwickler neuen Code zur Verfügung stellen, maximal einen Tag betragen.

[12] http://www.extremeprogramming.org/rules/testfirst.html , 2004

- Überstunden erwiesen sich in der Mehrzahl der Projekte als ein ungeeignetes Mittel zur Beschleunigung des Projektfortschritts.

4. **Test:**
   - Die Codierung der Komponententests hat in XP immer Vorrang vor jener der eigentlichen Ziel-Software. Um die Tests nach Modifikationen der Ziel-Software in einfacher Weise zu wiederholen empfiehlt sich die Nutzung von Test-Werkzeugen, die die Einzelschritte weitestgehend automatisieren.

   - Da der Test zum Zeitpunkt der Codierung der Ziel-Software schon existiert, können insbesondere Komponententests auf einfache Weise schon während der Fertigstellung der Ziel-Software angewandt werden.

   - Die Abnahmetests (engl. Acceptance Tests) werden in jeder Iteration entsprechend den User Stories geschrieben. Hierbei definiert der Kunde die Szenarien, die innerhalb der Abnahmetests zu überprüfen sind.

   - Gefundene Fehler (im Englischen werden Fehler als Bugs bezeichnet) werden entsprechend den User Stories im nächsten Iterationsschritt bzw. schon früher (anschließend an die Komponententests) korrigiert.

In Abb. 6.3 [13]) werden diese Regeln und Praktiken im XP-Prozess zusammengefasst.

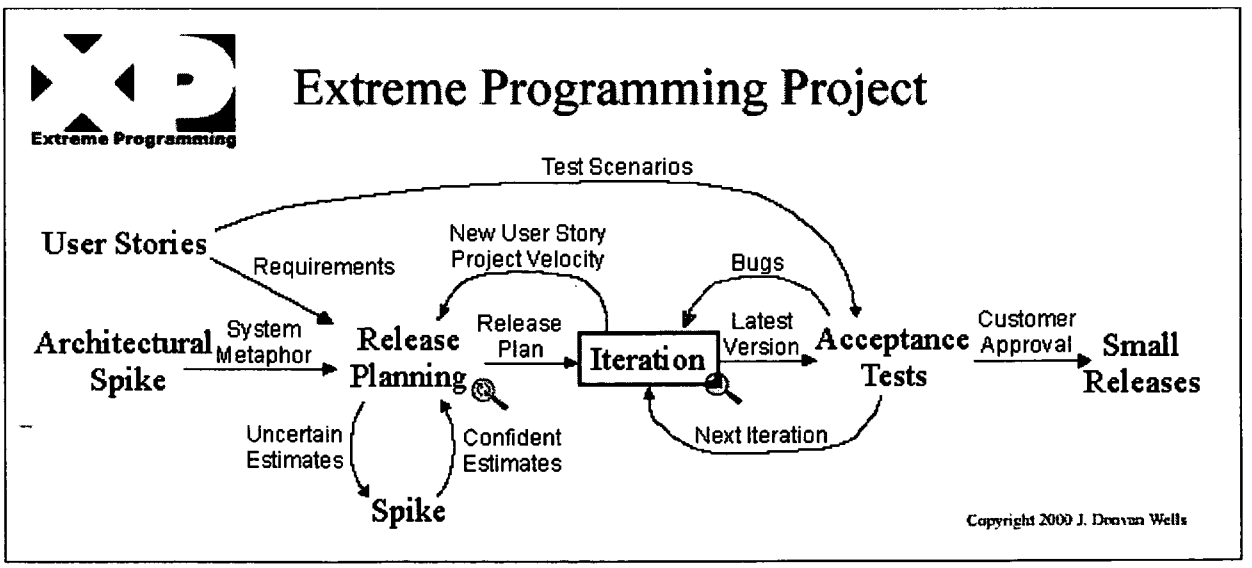


Abb. 6.3: XP-Prozess

[13] http://www.extremeprogramming.org/map/project.html , 2004

Abschließend sei angemerkt, dass dieses Verfahren insbesondere die Anforderungen der Entwickler berücksichtigt, die kontinuierlich mit neuen bzw. sich ändernden Requirements seitens des Kunden konfrontiert werden [14]).

6.2 Spiral-(Prozess)-Modell

Als eines der fortschrittlichsten Modelle der Software-Entwicklung gilt das Spiral-Modell. Dies ist der Grund, dass ihm ein eigener Hauptabschnitt gewidmet ist. Hierin wird einerseits versucht, diverse erprobte Entwicklungsmodelle in einen übergeordneten Rahmen einzubetten, und andererseits, die unbekannten Größen des Software-Projekts durch einen geeigneten Risiko-Management-Prozess zu regeln. Zentrale Motivation ist hierbei der effiziente und stufenweise Mitteleinsatz.

Gegenüber anderen Prozessen ist der Spiral-Prozess vielseitig anwendbar:

- Entwicklung umfangreicher Software-Systeme
- „Legacy System Replacement" (= Austausch bestehender Software-Systeme)
- Integration von Standard-Software-Komponenten
- Hardware-Entwicklung und -Integration

In Abb. 6.4 [15]) werden die wesentlichsten Eigenschaften wiedergegeben:

- zyklisches und gleichzeitiges Engineering
- Risiko-determinierte Auswahl von Prozessen und Produkten
- Risiko-determinierte System-Weiterentwicklung
- frühzeitige Eliminierung von Alternativen

Barry Boehm definiert das Spiral-Modell folgendermaßen [16]): „The spiral development model is a **risk-driven process model generator**, in which different risk patterns can lead to choosing incremental, waterfall, evolutionary prototyping, or other subsets of the process elements. It is used to guide multi-stakeholder concurrent engineering of software-intensive systems. It has two main distinguishing features. One is a cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions. ..."

[14] Litke, Hans-Dieter, Projektmanagement, München, 2004
[15] Boehm, B.W., A Spiral Model of Software Development and Enhancement, in: Computer, May 1988, p. 61-72
[16] Hansen, Wilfried J. (hrsg.), Spiral Development: Experience, Principles, and Refinements, Special Report, CMU/SEI-2000-SR-008, 2000

Die Risken haben in diesem Modell den höchsten Stellenwert – durch sie wird selbst der Entwicklungs-Prozess zur abhängigen Größe: „... At the start of a cycle, all of the project's success-critical stakeholders must participate concurrently in reviewing risks and choosing the project's process model accordingly." Mit Hilfe der "Anchor point milestones" wird der Entwicklungs-Fortschritt auf Basis des Spiral-Modells bezeichnet.

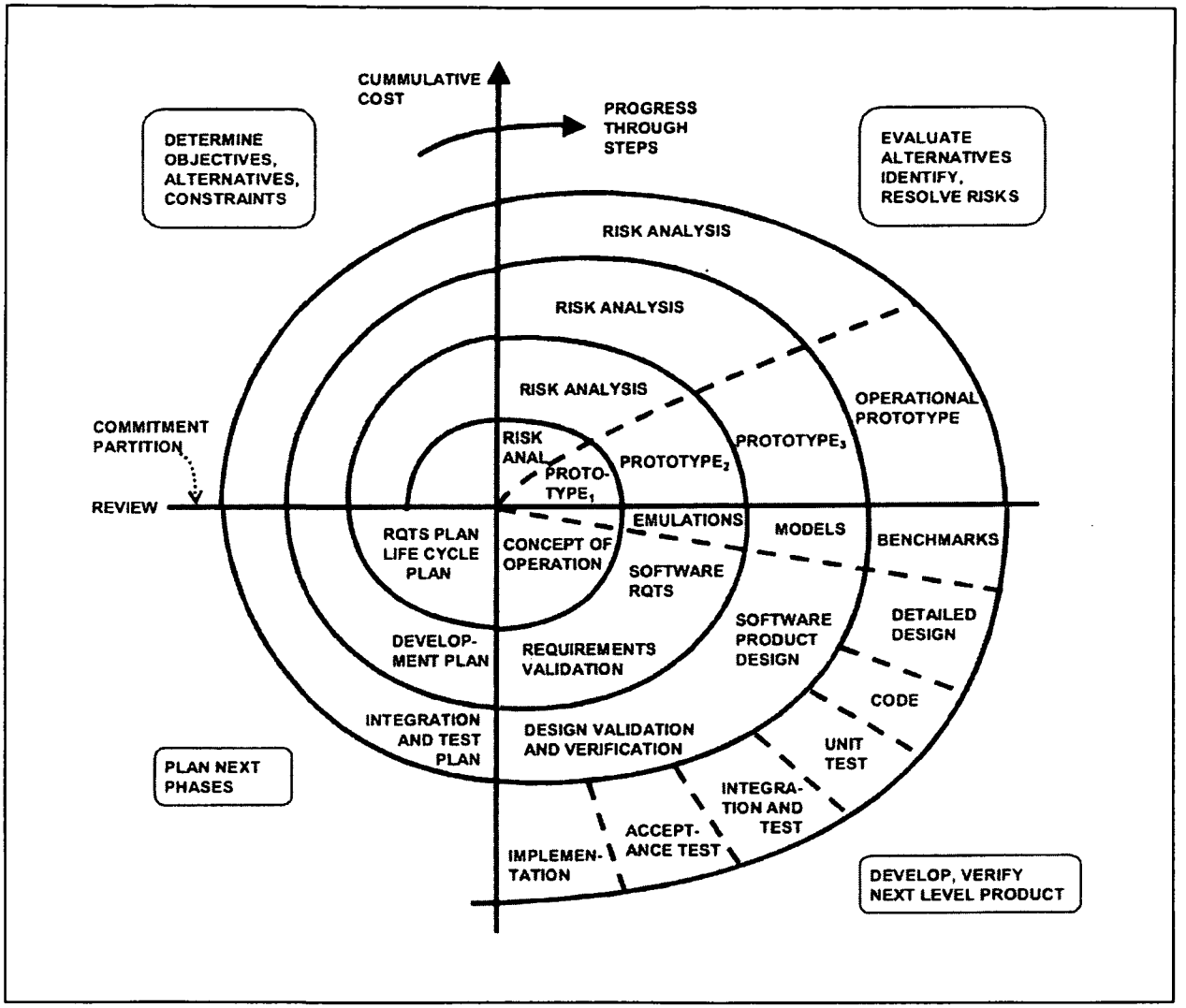


Abb. 6.4: Ursprüngliches Spiral-Modell

6.2.1 Invariante und variante Prozess-Größen

Unabhängig von der Spiral-Modell-Eigenschaft, sich an die unterschiedlichsten Randbedingungen anzupassen, zeichnet es sich durch einige charakteristische Merkmale

(Invarianten) aus [17]), die zudem die erfolgskritischen Eigenschaften des Prozesses sind. Gleichzeitig werden zur Unterscheidung auch variante Prozess-Größen angeführt. Um die durch die Invarianten geforderten Prozesseigenschaften zu unterstreichen, sind beispielhaft Modelle angeführt, die der jeweiligen Invariante widersprechen. Im folgenden werden die einzelnen Invarianten des Spiralmodells detailliert beschrieben.

6.2.1.1 Gleichzeitige Bestimmung der Software-Schlüssel-Parameter

Hierdurch wird gewährleistet, dass alle Requirements bzw. Anforderungen an die Software innerhalb der Rand- und Nebenbedingungen erfüllbar sind. Zu den gleichzeitig zu bestimmenden Schlüsselparameter zählen:

- Operations-Konzept
- System- und Software-Requirements
- System- und Software-Architektur
- wichtigste Code-Komponenten
    - COTS (= commercial-off-the-shelf) Komponenten, bzw. OEM-Produkte
    - Prototypen
    - Algorithmen

Als Negativ-Beispiele können die zu frühzeitige Plattform-Auswahl, inkompatible COTS Komponenten, oder die Verwendung von Komponenten mit unbestimmten Eigenschaften genannt werden. Der gleichzeitigen Bestimmung der wesentlichsten Komponenten steht das Wasserfall-Model diametral gegenüber. Dort müssen alle Requirements zu Beginn bekannt sein. Hierdurch wird z.b. vorausgesetzt, dass es zum vereinbarten Preis eine Lösung gibt, die durch die Requirements (mehr oder weniger) eindeutig bestimmt ist.

Durch die wechselseitigen Abhängigkeiten der Schlüsselparameter kann die Anzahl der Iterationen bis zur Erfüllung der Requirements aller Stakeholder variieren und nicht im vorhinein bestimmt werden.

6.2.1.2 Zyklische Berücksichtigung der Interessen aller Stakeholder

Grundvoraussetzung für jeden Entwicklungs-Zyklus ist die allgemeine Zustimmung zu den geplanten Anfangs- und Randbedingungen (oberer linker Quadrant). Diese werden in den

[17] Hansen, Wilfried J. (hrsg.), Spiral Development: Experience, Principles, and Refinements, Special Report, CMU/SEI-2000-SR-008, 2000

Schritten davor in einem Maß ermittelt, um innerhalb dieser Phase sinnvolle Entscheidungen treffen zu können. Hierzu zählen:

- Berücksichtigung der Stakeholder
- Berücksichtigung der kritischen Ziele und Einschränkungen (z.b. durch Reviews)
- Evaluierung aller Projekt- und Prozessalternativen
- Identifikation und Begrenzung aller mit den Alternativen verbundenen Risken
- Zustimmung aller Stakeholder zur Projektfortführung

Um ein Commitment zu erlangen, müssen die Stakeholder einige Eigenschaften erfüllen [18]). Der Stakeholder

- repräsentiert die Organisation (im Gegensatz zur Verfolgung ausschließlich persönlicher Ziele),
- hat die Ermächtigung um Entscheidungen treffen zu dürfen,
- kennt die kritischen Erfolgsfaktoren der repräsentierten Organisation, und
- hat Interesse am Fortschritt des Projekts.

Als variable Prozessgröße kann die Ermittlung (z.b. durch Prototyping, Simulation, Benchmarking, usw.) und Behandlung der Risken betrachtet werden.

6.2.1.3 Aufwands-Bestimmung aufgrund von Risken

Zur Bestimmung der Aufwände der einzelnen Arbeitspakete wird von der Prämisse ausgegangen, dass jede Software-Komponente fehlerhaft ist. Des weiteren wird angenommen, dass sowohl zu viele Fehler als auch ein zu später Markteintritt (durch überhöhte Testaufwände) zu suboptimaler Effizienz hinsichtlich der eingesetzten Ressourcen führt.

Als Varianten bleiben trotzdem die zur Risiko-Bestimmung eingesetzten Methoden und der Grad der Risiko-Elimination in jedem Zyklus. Je weniger Aufwände in einen einzelnen Zyklus investiert werden, umso mehr Zyklen sind bis zur Fertigstellung der Software nötig. Damit sind all jene Entwicklungsmodelle nicht anwendbar, die die Aufwände unabhängig von einer Risiko-Zahl festlegen oder Risken im Prozess nicht berücksichtigen.

[18] Boehm, B.W., Using the Win Win Spiral Model: A Case Study, IEEE Computer, July 1998, p. 33-44

6.2.1.4 Gestaltung des Detaillierungsgrades von Entwicklungskomponenten in Abhängigkeit vom Entwicklungs-Fortschritt

Die im vorhergehenden Punkt getroffene Aussage wird dahingehend verallgemeinert, als dass nicht immer davon ausgegangen werden darf, dass mit zusätzlichem Aufwand das Risiko stets minimiert wird. Wird z.b. eine GUI (= Graphical User Interface) schon zu Beginn zu detailliert spezifiziert, ist die Wahrscheinlichkeit groß, dass zu einem späteren Zeitpunkt eine nochmalige Spezifikation aufgrund neuer Erkenntnisse notwendig ist. Das heißt, wenn das Risiko durch eine genauere Spezifikation wächst, sollte diese zu einem späteren Iterationsschritt erfolgen.

6.2.1.5 Stufenweises Commitment zur Finanzierung des nächsten Zykluses

Aufgrund der großen Flexibilität hinsichtlich der Modell-Wahlmöglichkeit für den Entwicklungs-Prozess stellt sich die Frage nach einer sinnvollen Unterteilung in Meilensteine (oder „Anchor Points"). In einem Workshop [19]) unter Mitwirkung der University of Southern California und Rational Inc. wurden 3 wesentliche Meilensteine identifiziert:

- LCO (Life Cycle Objectives)
- LCA (Life Cycle Architecture)
- IOC (Initial Operational Capability)

Zu jedem der LCO- und LCA-Meilensteine kontrollieren die Stakeholder 6 Artefakte (= Spiralelemente oder allgemein: Arbeitspakete):

- Operationskonzept
- Prototyp-Ergebnisse
- Anforderungs-Spezifikation
- Architektur-Design
- Lebenszyklus-Plan
- Durchführbarkeit

Das Ergebnis zum Zeitpunkt des LCO-Meilensteins muss die Identifikation zumindest einer Realisierungs-Variante aus Geschäftssicht sein. Fokus des LCA-Meilensteins ist die Festlegung auf eine näher spezifizierte Lösungs-Strategie. Dazu gehört auch ein von allen Stakeholdern akzeptierter Risiko-Management-Plan. In dem oben erwähnten Workshop wurden in diesem Zusammenhang folgende Analogien gezogen:

- LCO = Verlobung

- LCA = Heirat
- IOC = 1.Kind

Zum Zeitpunkt der Erreichung des LCA-Meilensteins müssen alle wesentlichen Risken erfasst und „Notfallpläne" aufgestellt sein. Dies ist meistens auch der Zeitpunkt zu dem das Personal und andere Ressourcen aufgestockt werden.

Die Schlüssel-Elemente des IOC-Meilensteins sind:

- Software ist unter Berücksichtigung der Integration von OEM-Produkten fertiggestellt
- allfällige Daten-Konvertierungen sind abgeschlossen
- Software ist entsprechend Abnahmekriterien getestet
- Logistik zur vollständigen Lieferung aller Teile (inkl. Dokumentation, Lizenzen, usw.) ist vorbereitet
- Training der Endbenutzer, Administratoren und des Wartungspersonals ist vorbereitet

Die Finanzierungs-Entscheidung in der folgenden Runde des Entwicklungs-Zykluses soll von den Chancen und Risken bestimmt werden, was eine zugrunde liegende inkrementelle Strukturierung des Projekts voraussetzt.

6.2.1.6 Der System-Lebenszyklus als Gestaltungsrahmen

Diese Invariante betont die Rolle der „Unternehmerschaft" für ein Software-Projekt. Objektorientierte Methodiken müssen demnach auch

- Finanzierungs-Aktivitäten
- Kostenminimierungsaspekte
- Controlling-Aktivitäten
- Personal-Management
- System-Requirements: Performance, Safety, Security, Distribution, Localization, usw.
- usw.

inkludieren. Dies wird durch das Spiralmodell insofern ermöglicht, als in jedem Zyklus der Hardware/Software-Mix, die Software-Leistungsmerkmale, usw. an die Erfordernisse angepasst werden können.

[19] Clark, B., Boehm, B.W. (hrsg.), Knowledge Summary: Focused Workshop on COCOMO 2.0, USC-CSE, 1995

6.2.2 Mit dem Spiralmodell verwandte Modelle

Ein dem Spiralmodell ähnliches Modell ist das Evolutionsmodell. Dieses stimmt bezüglich einer stufenweisen Verfeinerung des gewünschten Ergebnisses mit dem Spiralmodell überein. Die Nachteile dieses Modells liegen einerseits in einer mangelnden Weiterentwickelbarkeit auf Basis der ursprünglichen Software-Architektur und andererseits in einer nicht auf die Ziele der Stakeholder ausgerichteten Priorisierung der Leistungsmerkmale. Dem Nachteil der mangelnden zeitlichen Korrelation mit Legacy-Systemen kann durch eine zusätzliche inkrementelle Modell-Komponente begegnet werden.

Auf Basis des Spiralmodells haben Institute und Firmen CASE-Werkzeuge entwickelt. Die University of Southern California entwickelte beispielsweise das Model-Based Architecting and Software Engineering (MBASE) Electronic Process Guide [20]). Der Prozess wird auf HTML-Basis dargestellt, wobei Templates einen Formalismus für die Anwendung vorgeben. Ein anderer, von der Firma Rational Software Corporation entwickelter Prozess (Rational Unified Process, = RUP) strukturiert anhand der oben erwähnten Meilensteine (LCO, LCA, IOC) die jeweiligen Erreichungsgrade von Requirements, Design, Implementation, Deployment und Management [21]).

Eine Weiterentwicklung des Spiralmodells von Barry Boehm ist das „WinWin Spiral Model" [22]). Darin wurde die Sektion über die Bestimmung der Ziele, Einschränkungen und Alternativen um die prozess-gesteuerte Einbindung der Stakeholder ergänzt. Der erste Quadrant der oben gezeigten Grafik (vgl. Abb. 6.4) wird im WinWin Spiralmodell in folgende 3 Schritte unterteilt:

"

1. Determine Objectives. Identify the system life-cycle stakeholders and their win conditions. Establish initial system boundaries and external interfaces.
2. Determine Constraints. Determine the conditions under which the system would produce win-lose or lose-lose outcomes for some stakeholders.
3. Identify and Evaluate Alternatives. Solicit suggestions from stakeholders. Evaluate them with respect to stakeholders' win conditions. Synthesize and negotiate candidate win-win alternatives. Analyze, assess, and resolve win-lose or lose-lose risks.

"

Damit wird die Bedeutung der Stakeholder für den Projekterfolg unterstrichen und explizit formuliert.

---

[20] Metha, N., MBASE EPG, USC-CSE, LA, CA, http://sunset.usc.edu/research/MBASE/EPG , 1999
[21] Kruchten, P., The Rational Unified Process, MA, 1998
[22] Boehm, B.W. & Bose, P., A Collaborative Spiral Software Process Model Based on Theory W, Proceedings, ICSP 3, IEEE, October 1994

6.3 Abbildung techno-sozialer Komplexität

6.3.1 Einleitung

Aufgrund des ständig wachsenden Umfangs software-gesteuerter Prozesse nimmt auch die Komplexität technischer Realisierungen zu. Bei hinreichender Analyse der Problemstellungen tritt allerdings ein zunächst eher unterbewerteter Faktor in den Vordergrund: Die soziale Komplexität. - Software wird von Menschen für Menschen geschaffen.

Auf Auftraggeberseite ist das Vorhandensein einer umfassenden Gesamt-Strategie für den Geschäftsbereich, in dem die Software zum Einsatz kommen soll, die Voraussetzung für die Auswahl bzw. Gestaltung derselben. Das Fehlen einer Strategie spiegelt sich in der Praxis häufig durch zwei Muster wider. Im ersten Fall ist keine ausreichend entwickelte Strategie vorhanden, was dazu führt, dass mit der Software-Auswahl die Strategie mitbestimmt wird. Ein typisches Beispiel hierfür ist die Einführung des weit verbreiteten Software-Produkts SAP. Als Vorteil ist die günstigere Anschaffung und Wartung im Vergleich zu individuellen Lösungen zu nennen. Kommt es (als Alternative) zum Einsatz individuell zu entwickelnder Software, ohne dass genügend Überlegungen zur Strategie des die Software nutzenden Geschäftsbereiches angestellt wurden, sollte mit umfangreichen Aufwendungen in der Analyse-Phase gerechnet werden. Des weiteren ist zu beachten, inwieweit die innerhalb des Bereiches tätigen Personen auch im Sinne einer (eventuell) existierenden Strategie handeln.

Die Bewältigung dieser Situationen erfordert bereits vor Beginn der Software-Entwicklung eine ausführliche Kommunikation zwischen allen Beteiligten. In erster Linie ist aber ein gemeinsames Verständnis der späteren Nutzer der Software über deren Zweckmäßigkeit unabdinglich. Bernd Oesterreich [23]) formuliert diesen „menschlichen Charakter" von Software mit den Worten: „Da Software mehr als andere technische Systeme mit menschlichen Abstraktionen durchsetzt ist, gleicht sie in ihrer Komplexität, ihren Tücken und Eigenheiten auch eher menschlichen Organisationsstrukturen als typisch technischen."

Im Umfeld der strategischen Unternehmungsgestaltung erläutert Herbert Götz [24]) die Zusammenhänge zwischen Unternehmensstrategie, Aufbau- und Ablauforganisation, sowie des Informations- und Kommunikationsmanagements. Dazu wird auf das Modell der ganzheitlichen Informations- und Kommunikationssystem-Architektur von H. Krcmar [25]) Bezug genommen. Darin werden die Anwendungs-, Daten- und Kommunikations-Architektur eines Unternehmens als Bindeglied zwischen dessen Infrastruktur einerseits, und der Prozess- und Aufbauorganisations-Architektur andererseits, interpretiert.

---

[23] Oesterreich, Bernd, Objektorientierte Softwareentwicklung, München, 1998
[24] Götz, Herbert, Entwicklung eines Modells eines konvergenten Informations- und Kommunikationssystems an Hand eines internationalen Unternehmens der Telekommunikationsindustrie, Dissertation an der TU-Wien, 2002
[25] Krcmar, H., Informationsmanagement, Heidelberg, 2000

Das Management der Informationswirtschaft konzentriert sich daher unter Berücksichtigung des eben Gesagten auf folgende Aktivitäten:

- Management der Informationsquellen
- Management der Informationsressourcen
- Management des Informationsangebotes
- Management der Informationsnachfrage
- Management der Infrastrukturen der Informationsverarbeitung und Kommunikation

6.3.2 Objektorientierung

Die Objektorientiertheit ist das vorläufige Optimum für eine metaphysische Repräsentation der „Wirklichkeit" – sieht man vom derzeit in Ansätzen befindlichen Trend zu Mustern hin - auch Patterns genannt - ab. Neben der objektorientierten Wahrnehmung, wie sie heute z.b. den Programmiersprachen C++ und JAVA zugrunde liegt, existieren grundsätzlich noch folgende Sichtweisen:

- Deklarative Sicht, die in der Programmiersprache PROLOG ihre bedeutendste Anwendung fand.
- Funktionale Sicht, die in der Programmiersprache LISP ihre bedeutendste Anwendung fand.
- Prozedurale Sicht, die, gemessen an den noch heute in Betrieb befindlichen Software-Systemen, den derzeit größten Einfluss hat. Die bedeutendsten Anwendungen sind die Programmiersprachen: ADA, BASIC, C, COBOL, FORTRAN, und PASCAL. In den letzten Jahren konnte beobachtet werden, wie diese Sprachen kontinuierlich um Charakteristika der objektorientierten Sicht ergänzt wurden, bzw. weitestgehend dem objektorientierten Paradigma angepasst wurden (z.b. C++, JAVA).

Zusammengefasst kann festgehalten werden, dass Programmiersprachen zwar stets Anteile aller Sichtweisen beinhalten, dem Trend zufolge jedoch eine Verlagerung des Schwerpunktes in Richtung objektorientierter Repräsentation unverkennbar ist. Als Gründe für diesen Trend sind folgende Charakteristika anführbar:

- einfachere Realisierungsmöglichkeit umfangreicher und komplexer Software
- der evolutionäre Grundgedanke des objektorientierten Designs erleichtert die nachträgliche Integration von zusätzlichen Software-Modulen
- notwendige Korrekturen sind besser lokalisierbar, wodurch die Systeme auch stabiler werden

- der anthropomorphe Grundgedanke der objektorientierten Struktur erleichtert die Kommunikation aller an der Software-Entwicklung Beteiligten
- die ganzheitliche Sichtweise während der Modellierung (mit ihren Strukturen und Abhängigkeiten) kommt der „Wirklichkeit" sehr nahe
- Unterstützung der Wiederverwendung

6.3.2.1 Die objektorientierte Strukturierung

Ausgangspunkt der objektorientierten Strukturierung ist der Wunsch, die „Wirklichkeit" – die reale Welt, wie wir sie mit unseren Sinnesorganen wahrnehmen können – möglichst 1:1 in die Sprache der Metaphysik – der abstrakten Welt – zu übertragen. Ebenso wie die natürlichen Sprachen im wesentlichen auf Substantiven, Verben und Adjektiven aufbauen, sind die Eckpfeiler der objektorientierten Struktur bzw. Sprache: Objekte, Methoden und Attribute.

Anhand eines Beispiels soll dies kurz erläutert werden: Als Objekt soll ein Auto betrachtet werden. Zu seinen Methoden zählen: Beschleunigen, Bremsen, Lichter anschalten, usw.; zu seinen Attributen zählen: Type, Farbe, Motorleistung, usw. Ebenso wie die Objekte der realen Welt einzigartig sind, gilt dieses Prinzip auch in der abstrakten Welt. Die Methoden und Attribute eines Objekts sind ihm eigen, was in der Sprache der Informatik als Datenkapselung bezeichnet wird. Aktiviert werden die einzelnen Methoden eines Objekts über Nachrichten, die an ein spezifisches Objekt gesandt werden.

Um die Struktur ähnlicher Objekte wiederverwenden zu können, bedient man sich dem Klassen-Konzept. Auf diese Weise kann ein komplexes Objekt modular aufgebaut und zudem durch seine konkreten Parameter zu einem „individuellen" Objekt instantiiert werden. Anders formuliert ist jedes Objekt Exemplar einer Klasse. Die Art des modularen Aufbaus komplexer Klassen bzw. Objekte wird durch Vererbungsbeziehungen und anderer Arten von Assoziationen (z.b. Aggregation, Komposition, usw.) beschrieben. Der durch die Betonung auf die Struktur in den Hintergrund tretende Algorithmus, der die Methoden der Objekte in einer bestimmten Sequenz aufruft, ist implizit auf die Methoden unterschiedlicher Objekte verteilt.

6.3.3 Entwicklung einer visuellen Kommunikationsform

Eine schon seit den Anfängen menschlicher Ingenieurskunst bewährte Methodik ist das Anfertigen von Plänen vor Beginn der Realisierung. In den Domänen der Architektur oder des Maschinenbaus wurde diese Technik schon vor langer Zeit zur Perfektion gebracht. Die Einfachheit dieser Zeichnungen im Vergleich zu jenen Sachverhalten die der Software-Entwicklung zugrunde liegen, lässt sich mit der vergleichsweise geringen Anzahl von

Relationen der einzelnen Elemente untereinander erklären. So bestehen zwischen Maschinenteilen nur an den Berührungspunkten Relationen (sieht man von dynamischen Eigenschaften – wie dem Schwingungsverhalten – einmal ab).

In der objektorientierten Softwareentwicklung ist dieser Sachverhalt bedeutend komplexer. Waren noch in den 60er Jahren fünf der sechs am häufigsten benutzten Programmiersprachen prozedural, werden heute kaum noch Software-Projekte in nicht-objektorientierter Sprache zu entwickeln begonnen. Da aber ein Großteil der Software-Entwicklung eine Ergänzung bereits existierender Software ist, findet der Paradigmawechsel zur objektorientierten Strukturierung nur langsam statt.

Obwohl der erste Einsatz objektorientierter Sprachen bereits Mitte der 80er Jahre stattfand, existieren entsprechende Methoden zu objektorientierter Analyse und Design erst seit Mitte der 90er Jahre. 1997 konsolidierten Grady Booch, Ivar Jacobson und James Rumbaugh ihre Methoden zur „Unified Modeling Language (UML)", die im November 1997 in der Version UML1.1 von der Object Management Group als Standard akzeptiert wurde.

Der dadurch erreichte Übergang von der technozentrischen Sicht der Struktogramme und Flussdiagramme zur anthropozentrischen Sicht, an der sich UML orientiert, zeigt sich vor allem in:

- Objektorientierte Gliederung: Durch das Klassenkonzept werden Eigenschaften und Operationen zusammengefasst
- Umfassendere Problemdarstellung: Die Aufmerksamkeit wird vom Lösungsraum (Algorithmen) in den Problemraum verschoben (Relationen zwischen Objekten)
- Evolutionäre Entwicklung: Die Möglichkeit für Erweiterungen wird offen gehalten

An dieser Stelle sei jedoch vermerkt, dass durch die UML keine Entwicklungsmethodik beschrieben wird.

Zentrale Motivation der UML ist die Schaffung einer **einheitlichen Notation und Semantik zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Software**. Dazu gehört vor allem ein einfaches System graphischer Modell-Elemente und Diagramme zur Repräsentation unterschiedlicher Aussagen. Als Diagramm-Typen existieren in UML (vgl. [26])):

- Use-Case-Diagramme (zeigen die Beziehungen zwischen den Akteuren)

- Klassen-Diagramme (zeigen die statische Modell-Struktur der involvierten Objekte mit ihren Attributen und Methoden)

- Verhaltens-Diagramme (zeigen die Abläufe in unterschiedlichster Darstellung)

- Aktivitäts-Diagramme
- Kollaborations-Diagramme
- Sequenz-Diagramme
- Zustands-Diagramme

- Implementierungs-Diagramme
  - Komponenten-Diagramme
  - Verteilungs-Diagramme

Roman Severin, UML-Experte bei Siemens, charakterisiert UML folgendermaßen: „The Unified Modeling Language (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artefacts of software systems. It simplifies the complex process of software design, making a "blueprint" for construction. The UML definition was led by Rational Software's industry-leading methodologists: Grady Booch, Ivar Jacobson and James Rumbaugh."

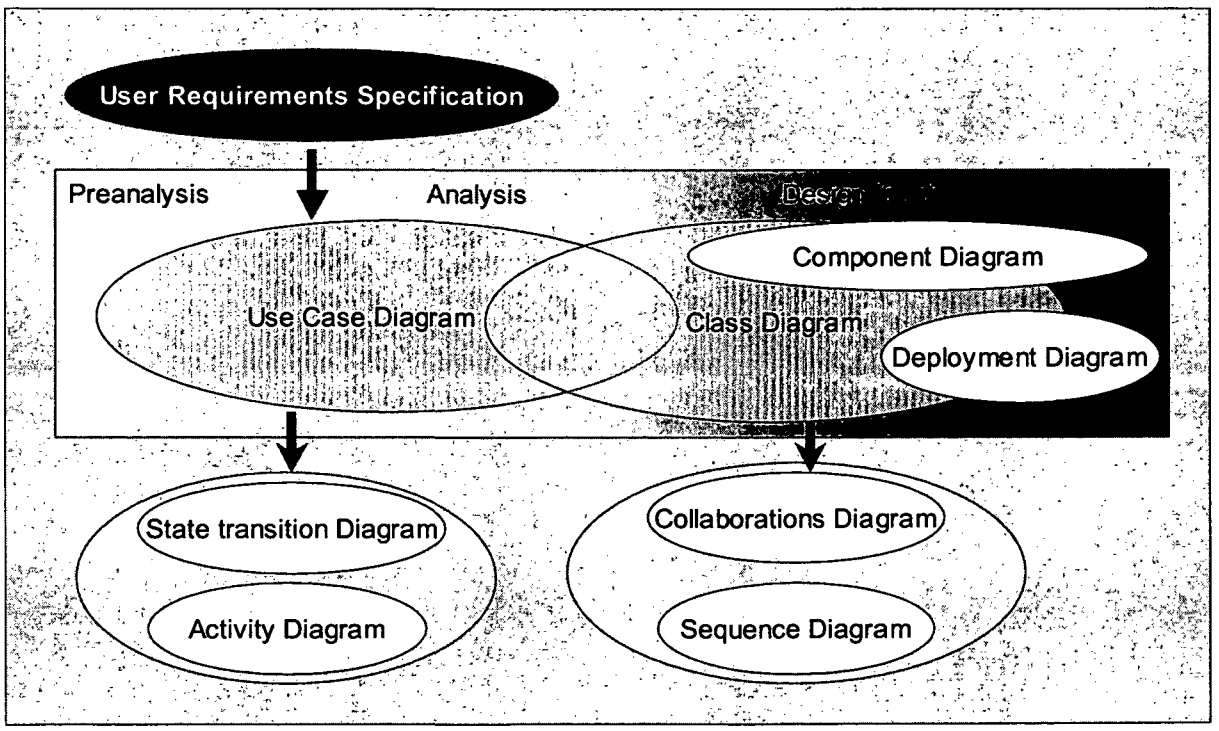


Abb. 6.5: UML diagrams

Da der Fokus dieses Kapitels auf der Abbildung techno-sozialer Komplexität liegt – ebenso wie der Grundgedanke von UML – soll der Rahmen für die nachfolgenden Diskussionen auf die Entwicklung kommerzieller Software gelegt werden (im Gegensatz zu Systemen mit wenigen Interaktionen mit Menschen, wie z.b. Betriebssysteme). Mit dieser Fokussierung wird zudem

[26] Oesterreich, Bernd, Objektorientierte Softwareentwicklung, München, 1998

die Auswahl auf jene Software-Entwicklungs-Prozesse eingeschränkt, die durch folgende Kriterien charakterisierbar sind:

1. Use-Case Orientierung
2. Architektur- und Komponenten Zentriertheit
3. Iteratives Vorgehen (Entwicklung vom Groben zum Detail)
4. Inkrementelles Vorgehen (Gesamtfunktionalität wächst mit jedem Schritt der Detaillierung)

Erläuterung zu 1:
Ein Use-Case beschreibt eine überschaubare Menge von Aktivitäten eines Systems aus der Sicht seiner Akteure. Diese Aktivitäten führen zu einem wahrnehmbaren Ergebnis für die Akteure, die stets einen Use-Case initiieren. Ein Beispiel für einen Use-Case ist das Freischalten eines neuen Kunden in einem Mobilnetz durch telefonische Meldung in einem Call-Center. Der Akteur ist in diesem Fall die Person, die die Daten des Kunden in eine Bildschirm-Maske eines dafür geeigneten Computer-Programms einfügt.

Erläuterung zu 2:
Das Vorgehen in der Anwendungsentwicklung muss die Eigenschaften einer vorgegebenen Architektur berücksichtigen, wodurch die Freiheitsgrade der Software-Entwicklung eingeschränkt werden. Dies kann z.B. durch eine Gliederung von Systemen in Subsysteme (Hierarchienbildung) oder durch die Vorgabe eines Schichtenmodells (z.B. Unterteilung in: Vorgangssteuerung, Dialogsteuerung, Fachklassen) geschehen. Architekturkonzentriertheit bedeutet somit die Erarbeitung der durch die Architektur vorgegebenen Artefakte.

Je nach Entwicklungs-Fortschritt (User Requirements Specification, Preanalysis, Analysis, Design/Coding) der Software eignen sich unterschiedliche Konstrukte von UML zur Visualisierung des erarbeiteten Sachfortschritts (vgl. Abb. 6.5 [27]).

Die in den weiteren Abschnitten näher beschriebenen Elemente von UML gestatten auf einfache Weise die Darstellung von Aufbau- und Ablauforganisationen und sind somit schon im Vorfeld einer allfälligen Software-Entwicklung anwendbar!

[27] Severin, Roman, Vortrag bei Siemens in Wien, Mai, 2002

6.4 Modellierung des Inhalts von Software (UML)

Wie bereits in den oben angeführten einleitenden Abschnitten beschrieben, ist die UML (= Unified Modeling Language) keine Methode zur Entwicklung von Software, sondern eine Sprache und Notation zur Modellierung von Software-Systemen. Dazu stellt sie eine definierte Menge von Modellierungskonstrukten mit einheitlicher Notation und Semantik bereit. Die UML (als Sprache) selbst ist durch ein Metamodell definiert, das im weiteren nicht behandelt wird. Stattdessen wird ein Überblick über die Sprache (im wesentlichen verschiedene Diagrammtypen) gegeben, wie sie für die Realisierung von Software für betriebliche Informationssysteme notwendig ist. Die nachfolgenden Abschnitte sind nach Diagrammtypen geordnet, in denen auch die dort vorzugsweise anzutreffenden Modell-Elemente nähert beschrieben sind. - Grundsätzlich können einige Modellelemente auch in anderen Diagrammtypen vorkommen.

6.4.1 Anwendungsfall-Diagramm (Use-Case diagram)

Dieses Diagramm zeigt die Beziehungen zwischen ein oder mehreren Akteuren und ein oder mehreren Anwendungsfällen. Ein Anwendungsfall setzt sich aus einer Menge von Aktivitäten zusammen, die, in ihrer Summe betrachtet, für einen Akteur ein wahrnehmbares Ereignis beschreiben. In der Sprache der Systemtheorie beschreibt ein Anwendungsfall das über die Systemgrenzen hinaus erkennbare Verhalten des Systems (Input + Output). Sowohl der Input, als auch der Output, ist durch eine Menge von Faktorkombinationen definiert – die jeweils ein Szenario bilden. In der Abb. 6.6 [28]) wird ein Beispiel für ein Anwendungsfall-Diagramm gezeigt.

Der durch eine Ellipse dargestellte Anwendungsfall, welcher in letzter Konsequenz informations-technisch weitestgehend automatisiert werden soll, wird durch einen Text mit folgenden Inhalten (kontextabhängig) näher beschrieben:

- am Anwendungsfall beteiligte Akteure
- Vorbedingungen für den Anwendungsfall
- Nachbedingungen des Anwendungsfalls
- Invarianten
- nicht-funktionale Anforderungen (Plattformvoraussetzungen, Antwortzeitverhalten, usw.)
- Ablaufbeschreibung, welche die Szenarien beschreibt
- Ausnahmen und Fehlersituationen
- Regeln (Geschäftsregeln, Gültigkeits- und Validierungsregeln, usw.)

[28] Severin, Roman, Vortrag bei Siemens in Wien, Mai, 2002

- Protokollnotizen, die auf die im Rahmen der Erarbeitung des jeweiligen Anwendungsfalls beteiligten Personen verweisen
- Verweise auf Dokumente, Handbücher, Referenzen, Bildschirmdialoge und -formulare, die den Anwendungsfall näher dokumentieren

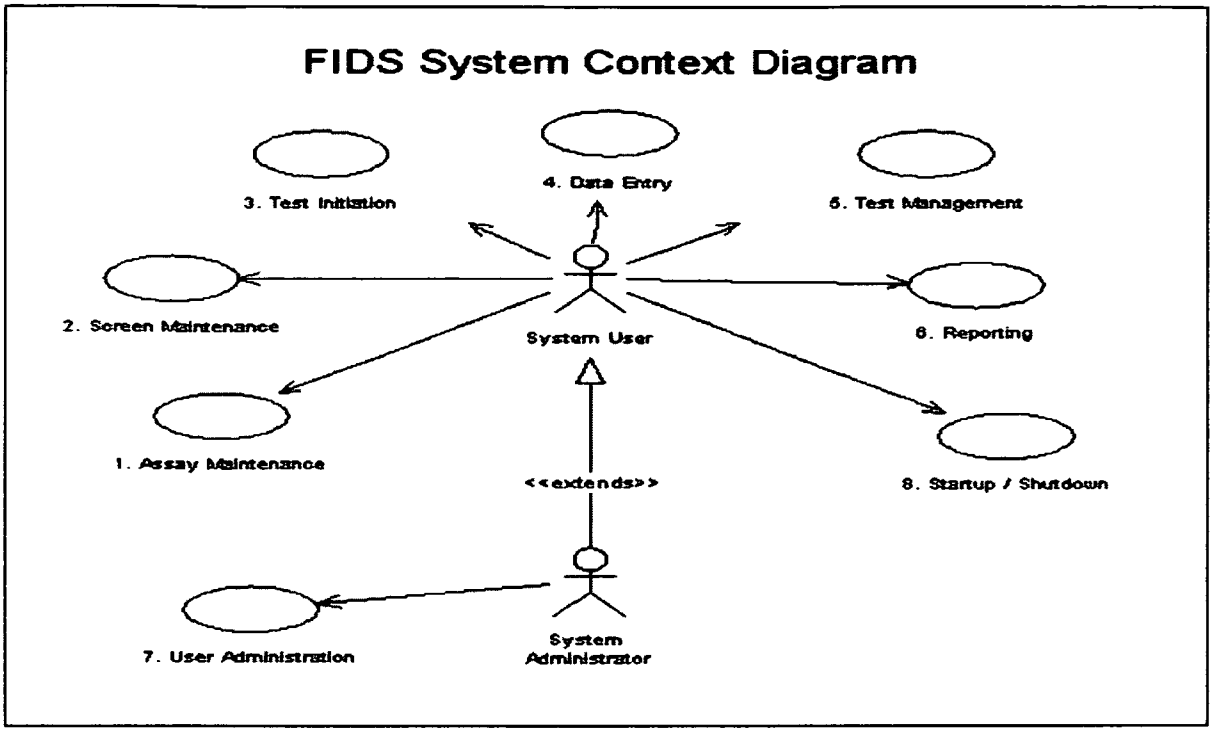


Abb. 6.6: Use-Case diagram

Ein Akteur ist eine außerhalb des Systems liegende Klasse, die an der in einem Anwendungsfall beschriebenen Interaktion beteiligt ist. Akteure sind z.b. Anwender der Software, können aber auch - in einer Verallgemeinerung - externe Systeme oder zeitliche Ereignisse sein. Hierbei wird nicht ein bestimmtes Objekt, sondern wie die Bezeichnung: „Klasse" nahe legt, eine bestimmte Rolle bezeichnet (mit der das System kommuniziert).

Akteure und Anwendungsfälle stellen gemeinsam die Funktionalität des zur Diskussion stehenden Gesamtsystems dar. Der Nutzen, der mit dem Anwendungsfall-Diagramm geschaffen wurde, ist einerseits die Auflistung aller „subjektiv" bedeutenden Handlungen (= Anwendungsfälle), und andererseits deren für den Menschen angepasste Form einer graphischen Repräsentation. Mit dem Adjektiv „subjektiv" soll nochmals unterstrichen werden, dass die identifizierten Anwendungsfälle keiner mathematisch-logischen Formulierung genügen, sondern das Ergebnis einer iterativ geführten Diskussion aller Beteiligten sind. In weiterer Folge wird aufbauend auf dem Anwendungsfall-Diagramm die Analyse weiter vertieft.

6.4.2 Klassen-Diagramm (Class diagram)

Im Gegensatz zum Anwendungsfall-Diagramm ist das Klassen-Diagramm eine statische Beschreibung der zu erstellenden Software. Die Strukturierung in Klassen nimmt den Gestaltungsansatz der Objektorientierung bereits vorweg. Die Definition der Klasse setzt sich aus Attributen und Methoden zusammen. Das potentielle Verhaltensmuster wird durch die Summe der Methoden festgelegt. Diese werden durch von außerhalb der instantiierten Klasse erfolgende Methodenaufrufe aktiviert und abgearbeitet. Das Exemplar einer Klasse wird als Objekt bezeichnet.

Arnold Hickersberger erklärt den Begriff der Klasse mit folgenden Worten [29]): „Die Dinge der Welt erkennen wir auch anhand ihrer Zugehörigkeit zu Gruppen, die wir durch Klassifizierung erhalten. Alle Objekte um uns sind Spezialisierungen allgemeiner Klassen: eine Katze ist Mitglied der Klasse Säugetiere, diese sind Mitglieder der Klasse Tiere, diese der Klasse Lebewesen usw. Die Klassifizierung erfolgt über gemeinsame Merkmale, wobei die spezielleren Klassen die Eigenschaften der allgemeineren übernehmen."

Von besonderer Bedeutung in Klassendiagrammen sind jedoch die Verknüpfungen zwischen den Klassen untereinander. Diese können durch folgende Basis-Mechanismen erfolgen:

1. Vererbung (Generalisierung und Spezialisierung)
2. Assoziation (in den Ausprägungen: rekursive A., attributierte A., Assoziations-Zusicherung, qualifizierte A., abgeleitete A., mehrgliedrige A., gerichtete A.)
3. Aggregation (Komposition)
4. Abhängigkeitsbeziehung

Zu 1: Vererbung (Generalisierung und Spezialisierung):
Die Vererbung ist ein Programmiersprachen-Konzept für jene Relation zwischen Klassen, derzufolge die Attribute und Methoden der Oberklasse auch den von ihr abgeleiteten Unterklassen zugänglich sind. Generalisierung und Spezialisierung bezeichnen die Taxonomie, wonach die Oberklasse das allgemeinere (Generalisierung) und die Unterklasse das speziellere Element (Spezialisierung) darstellt. Die Unterscheidung – der Diskriminator – zwischen Ober- und Unterklasse ist das Ergebnis der Verteilung des semantischen Gehalts zwischen den Klassen.

Zu 2: Assoziation (in den Ausprägungen: rekursive A., attributierte A., Assoziations-Zusicherung, qualifizierte A., abgeleitete A., mehrgliedrige A., gerichtete A.):
Entsprechend dem Diskriminator in der Vererbungsbeziehung wird durch eine Assoziation eine allgemeine semantische Beziehung zwischen Klassen beschrieben. Z.b. bezeichnet die Assoziation „malt" die Beziehung zwischen der Klasse/Objekt „Maler" und der gemalten

[29] Hickersberger, Arnold, Der Weg zur objektorientierten Software, Heidelberg, 1994

Klasse/Objekt „Baum". Da der Begriff der Assoziation viele Ausprägungen haben kann, existieren für die gebräuchlichsten Beziehungen eigene Namen, die in den unterschiedlichen Programmiersprachen durch eigene Konstrukte realisiert werden. In diesem Zusammenhang kann auch von Mustern gesprochen werden.

Zu 3: Aggregation (Komposition):
Die Aggregation ist eine wichtige Variante der Assoziation, die zwischen den beteiligten Klassen eine Ganzes-Teile-Hierarchie darstellt. Die Komposition ist eine Aggregation, bei der mit Beendigung der Existenz des Ganzen auch die Teile ihre Existenz verlieren. Z.b. macht eine Aussage über eine Abteilung einer Firma keinen Sinn, wenn die Firma nicht mehr existiert.

Zu 4: Abhängigkeitsbeziehung:
Damit kann ausgedrückt werden, dass Korrekturen in einem Modellelement (Klasse) Änderungen in anderen Modellelementen nach sich ziehen.

Abb. 6.7 zeigt ein Beispiel für ein Klassen-Diagramm, das zwei gerichtete Assoziationen enthält [30]).

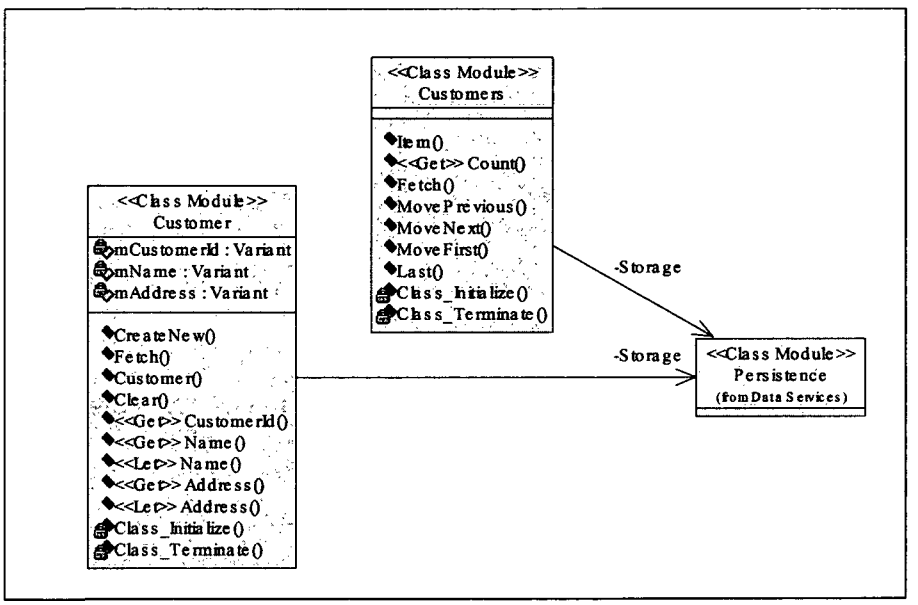


Abb. 6.7: Class diagram

6.4.3 Verhaltens-Diagramme (Dynamic diagrams)

Verhaltens-Diagramme gestatten die Darstellung des dynamischen Verhaltens der zu erstellenden Software. In einem ersten Ansatz wird das Verhalten in einem Anwendungsfall-Diagramm dargestellt. Danach werden die Anwendungsfälle in ihre Teilaktivitäten untergliedert und deren Dynamik in Aktivitäts-Diagrammen, Kollaborations-Diagrammen, Sequenz-Diagrammen, und/oder Zustands-Diagrammen visualisiert.

[30] Severin, Roman, Vortrag bei Siemens in Wien, Mai, 2002

6.4.3.1 Aktivitäts-Diagramm (activity diagram)

Aktivitäts-Diagramme beschreiben innerhalb der jeweils gesteckten Systemgrenzen alle potentiellen Ablaufmöglichkeiten eines (Teil-)Systems. In der Welt der prozeduralen Programmiersprachen entspricht das Aktivitäts-Diagramm (vgl. Abb. 6.8) dem Fluss-Diagramm. Allerdings gilt für die Aktivitäten im Rahmen des objektorientierten Paradigmas, dass sie einer Operation, einer Klasse, oder zumindest einem Anwendungsfall zugeordnet werden. Unter einer Aktivität versteht man in UML definitionsgemäß eine interne Aktion mit einer oder mehreren ausgehenden Transitionen (= Zustandsübergang), die den Abschluss der internen Aktion charakterisieren.

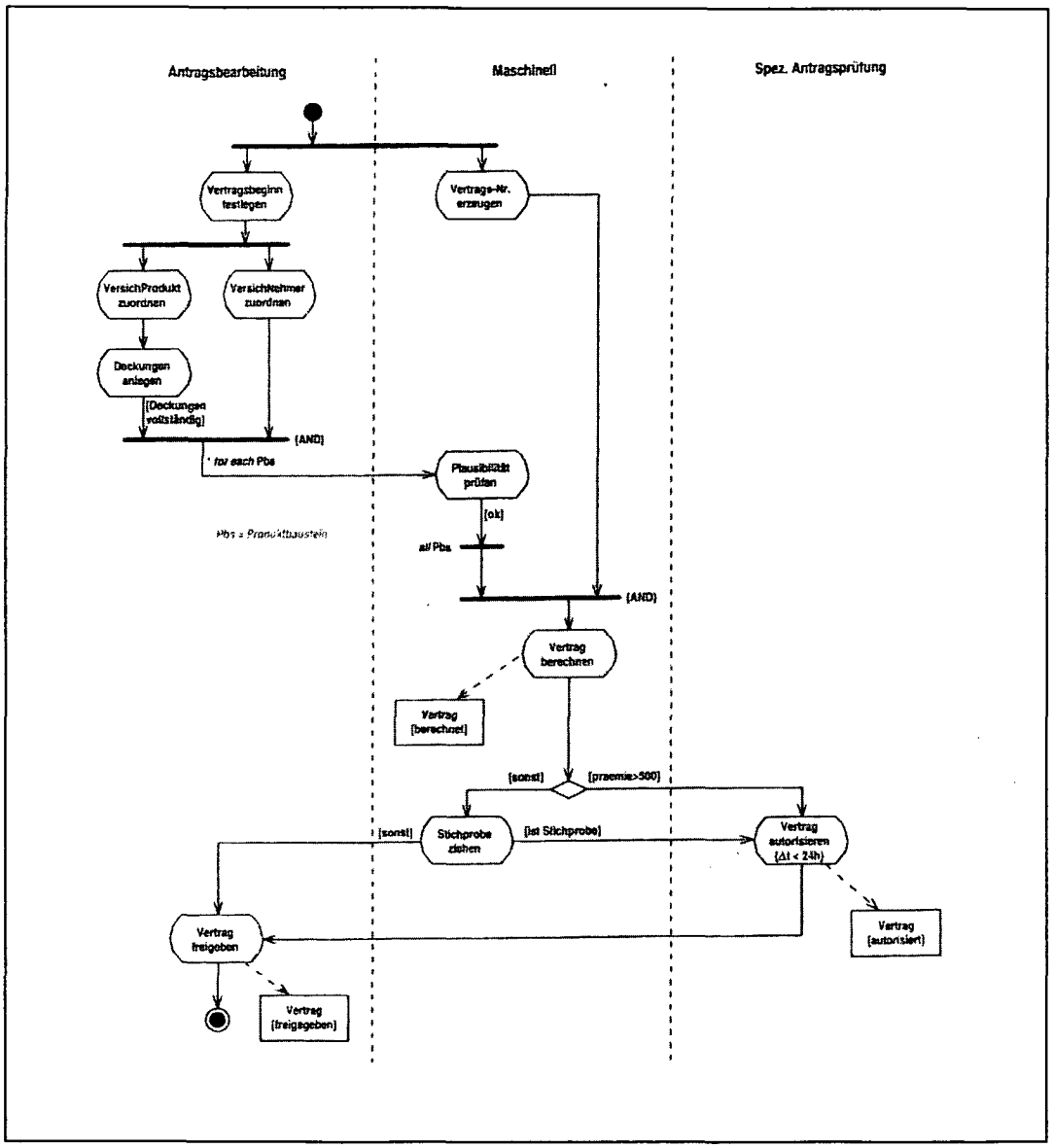


Abb. 6.8: Activity diagram

Ein wichtiges Charakteristikum dieses Diagramms ist die Schwerpunktsetzung auf die vollständige Modellierbarkeit von fachlichen Zusammenhängen und Bedingungen. Durch die Möglichkeit zusammengehörige Aktivitäten von anderen graphisch abzugrenzen, eignen sich Aktivitäts-Diagramme vorzüglich zur hierarchischen Gruppierung. Dies ist vorteilhaft für die Modellierung von Geschäftsprozessen, bei denen im allgemeinen Aktivitäten einzelnen Verantwortungsbereichen zugeordnet werden. Zudem ermöglicht das Instrumentarium von UML auch die Einbindung von Geschäftsregeln und Entscheidungslogiken. Durch eine entsprechende graphische Strukturierung ist zusätzlich eine chronologische Ordnung implementierbar.

6.4.3.2 Kollaborations-Diagramm (collaboration diagram)

Dieses Diagramm zeigt ein sowohl hinsichtlich der beteiligten Objekte, als auch hinsichtlich der Menge an Interaktionen einen begrenzten Ausschnitt eines (Teil-) Systems. Damit fokussiert es im Vergleich zum Aktivitäts-Diagramm stärker auf die Topographie der Objekte bei gleichzeitigem Verzicht auf eine – wie auch immer definierte – Vollständigkeit. Im Mittelpunkt steht immer ein bestimmtes Szenario mit den darin beteiligten Objekten und ihrem Nachrichtenaustausch.

In der Abb. 6.9 [31]) ist ein Beispiel für ein derartiges Diagramm gezeigt. Es enthält Objekte und die an sie gesandten Nachrichten. Die Zahlen bezeichnen die Sende-Reihenfolge der Nachrichten.

Aufgrund der Darstellbarkeit von Objektzuständen in Aktivitäts-Diagrammen (dieser Diagrammtyp ist innerhalb von UML relativ neu) und die in diesen Diagrammen im Gegensatz zu Kollaborations-Diagrammen einhaltbare Vollständigkeit, wird dieser Diagrammtyp (vgl. Abb. 6.9) nur für ausgewählte – besonders diskussionswürdige – Situationen eingesetzt.

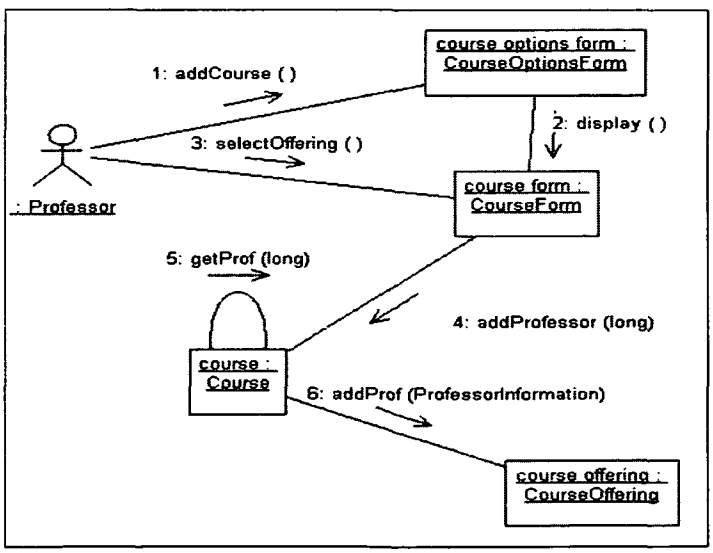


Abb. 6.9: Collaboration diagram

[31] Severin, Roman, Vortrag bei Siemens in Wien, Mai, 2002

6.4.3.3 Sequenz-Diagramm (sequence diagram)

Bezugnehmend auf den Inhalt zeigt das Sequenz-Diagramm die gleiche Information wie das Kollaborations-Diagramm – allerdings mit einer anderen graphischen Schwerpunktsetzung. Im Vordergrund steht hier der zeitliche Ablauf der Nachrichten und die Lebensdauer der Objekte. Abb. 6.10 [32]) zeigt ein Beispiel für ein Sequenzdiagramm. Es entstammt dem gleichen Kontext wie das zuvor gezeigte Beispiel eines Kollaborations-Diagramms. Im Vergleich zum Aktivitäts-Diagramm, das auch ohne Bezugnahme auf eine objektorientierte Programmiersprache für Modellierungs-Zwecke sinnvoll einsetzbar ist, setzt dieser Diagrammtyp – so wie das Kollaborations-Diagramm – die Definition von Objekten voraus.

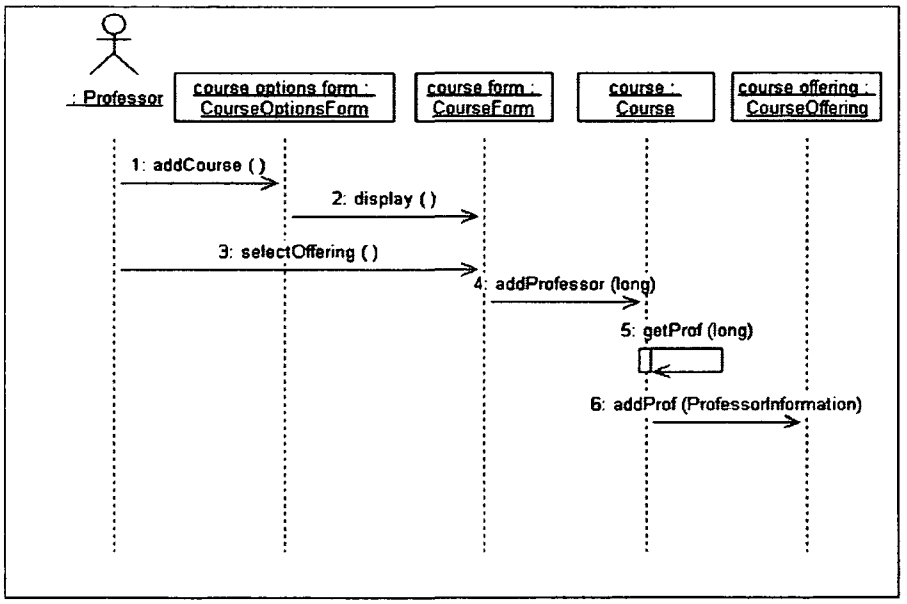


Abb. 6.10: Sequence diagram

6.4.3.4 Zustands-Diagramm (state diagram)

Dieser Diagrammtyp existiert ebenso wie das Flussdiagramm schon seit den Anfängen der Computer-Technologie. Die formale Basis hierzu liefert die Automaten-Theorie. Ziel des Zustands-Diagramms ist die Visualisierung der Folge von Zuständen, die ein Objekt einnehmen kann. Des weiteren sind jene Ereignisse erfasst die bestimmte Aktionen auslösen, d.h. den Aufruf zugeordneter Methoden.

[32] Severin, Roman, Vortrag bei Siemens in Wien, Mai, 2002

6.4.4 Implementierungs-Diagramme (Implementation diagrams)

Entsprechend den vorherigen Abschnitten beginnt die Analyse mit Anwendungsfall-Diagrammen, die den gegebenen Sachverhalt in anschaulicher Weise und unter Verwendung vordefinierter Symbole graphisch präsentieren. In weiterer Folge werden die Analysen detaillierter, die Teile spezifiziert, und in Zusammenhang mit anderen Teilen gebracht. Nachdem nun alle Details erarbeitet wurden, müssen diese wieder in größeren Einheiten zusammengefasst werden. – Und diesem Zweck dienen die Implementierungs-Diagramme.

6.4.4.1 Pakete-Diagramm (package diagram)

Dieser Diagramm-Typ dient der logischen Zusammenfassung von programmtechnischen Komponenten (z.b. Klassen) und zeigt überdies die Abhängigkeiten der so gebildeten Pakete untereinander. Ein Beispiel für Pakete und deren Abhängigkeiten zeigt Abb. 6.11 [33]).

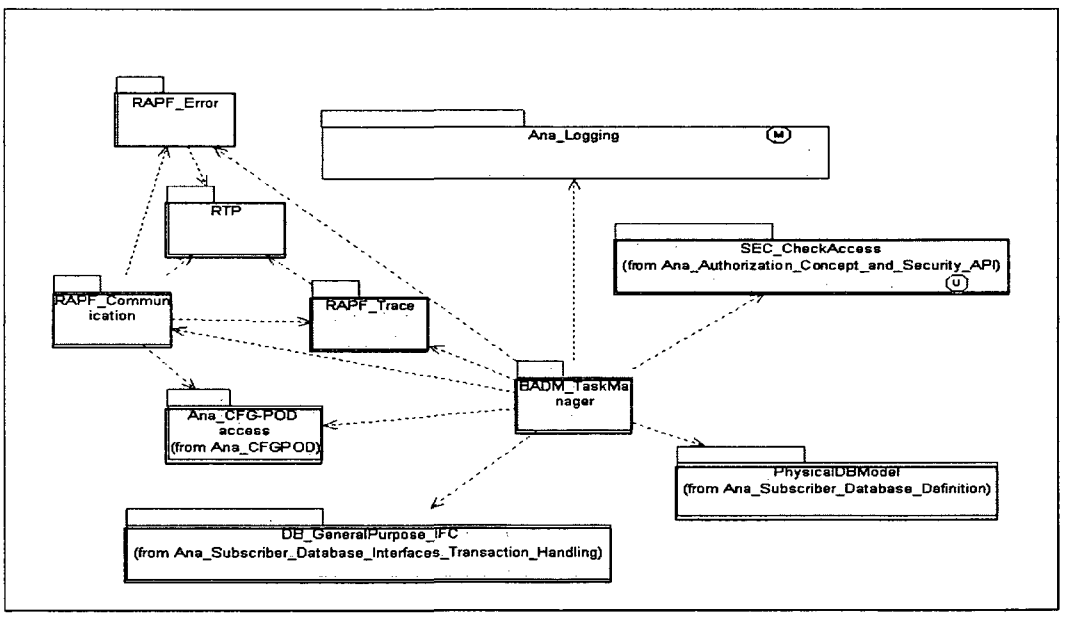


Abb. 6.11: Package diagram

[33] Severin, Roman, Vortrag bei Siemens in Wien, Mai, 2002

6.4.4.2 Komponenten-Diagramm (Component diagram)

Unter einer Komponente versteht man in diesem Diagramm-Typ ein physisches Stück Programmcode. Im Grunde dient das Komponenten-Diagramm ebenso wie das Pakete-Diagramm einer Zusammenfassung zusammengehöriger Elemente. Der Unterschied besteht in der zusätzlichen Darstellbarkeit von Schnittstellen sowie von Compiler- und Laufzeitabhängigkeiten. Die Schnittstellen einer Komponente geben die Sichtbarkeit der Elemente, oder Teilen davon, nach außen hin an. Ein Beispiel für ein Komponenten-Diagramm zeigt Abb. 6.12 [34]).

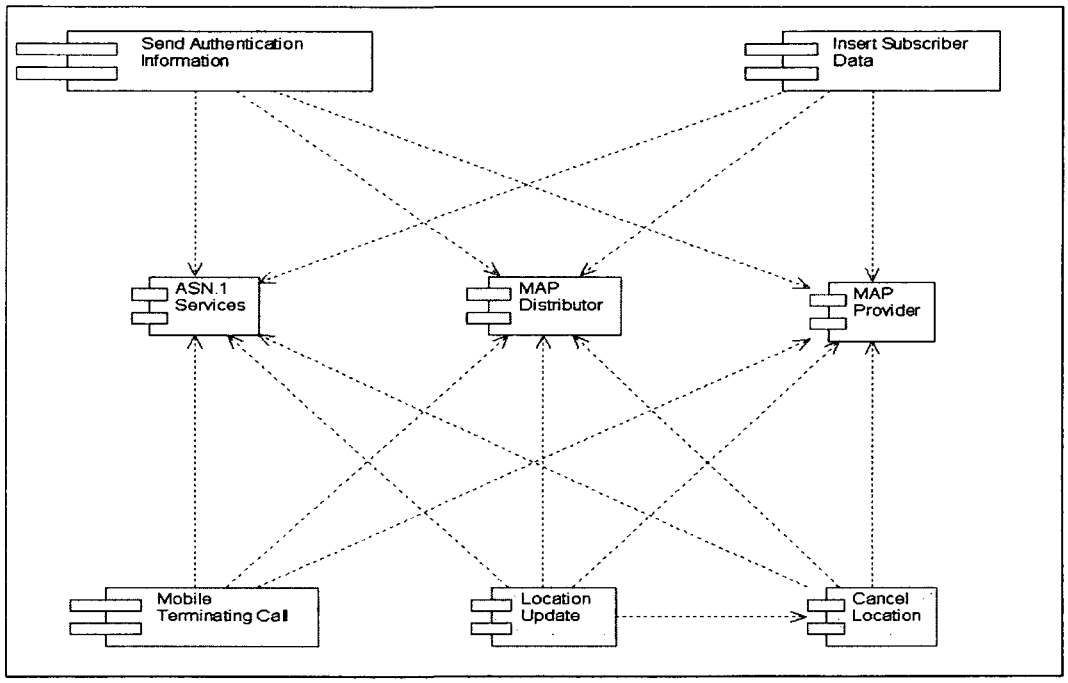


Abb. 6.12: Component diagram

6.4.4.3 Verteilungs-Diagramm (Deployment diagram)

Oftmals läuft Software verteilt auf mehreren Computern (z.b. Server-Client-Architekturen). In vielen Fällen werden darüber hinaus auch andere Geräte (z.b. Drucker, Router, Alarmmelder, usw.) integriert. Um diese Knoten optimal zu nutzen bedarf es einer geeigneten Verteilung der Software-Komponenten auf die jeweiligen Hardware-Komponenten, wobei zur graphischen Darstellung Verteilungs-Diagramme benutzt werden.

[34] Severin, Roman, Vortrag bei Siemens in Wien, Mai, 2002

7. Aufwandsschätzung und Produktivität

Neben der Projektplanung bzw. Ablaufplanung ist die Aufwandsplanung das zweite wesentliche Element der Planungsaktivitäten eines Software-Projekts. Die Ablaufplanung hat die logische Anordnung der detaillierten Arbeitspakete zur Aufgabe. Als Ausgangspunkt hierfür dienen die Phasenablaufmodelle.

Unter der Aufwandsplanung wird in erster Linie die Planung der Personal-Kapazitäten für die einzelnen Arbeitspakete verstanden. Die Aufwände für den Zukauf von Hardware und Software sind in den meisten Fällen vernachlässigbar. Zumindest ist aber ihr Ausmaß gut kalkulierbar, was für die Bestimmung des Arbeitsaufwands nicht zutrifft. Aus den unterschiedlichen Berichten über Software-Projekte geht hervor, dass die tatsächlichen Arbeits-Aufwände bis zu 400% von den Planwerten zu Projektbeginn abweichen.

Eine anschauliche Darstellung dieser Problematik wird anhand der Beurteilung des Leistungsstandes eines Software-Projekts gegeben. Hierbei wird aufbauend auf eigenen Erfahrungen ein einfach zu realisierendes Verfahren („Verfahren der kleinsten Arbeitspakete") dargestellt.

7.1 Aufwandsschätzung

Grundlage jedweder Aufwandsplanung und darauf aufbauend der Terminplanung ist die Schätzung des Aufwands zu Beginn eines Software-Projekts. Obwohl wesentlicher Bestandteil einer Auftragsabwicklung genießt sie in der Praxis nicht jene Beachtung die ihr zukommen sollte. Die Gründe dafür sind u.a.:

- Mangel an Know-how über Schätzmethoden
- Unzureichende Motivation seitens der Firmenleitung
- Allgemeiner Akzeptanzmangel durch beschränkte Ergebnisqualität der Schätzung

Hierzu anzutreffende Aussagen aus der Praxis sind:

„Mehrmals über kleine Verzögerungen zu berichten ist schlechter als eine einmalige größere Verzögerung am Projektende bekannt zu geben. Und zu Beginn kann keine Schätzung genaue Werte liefern."

„Schätzverfahren sind sowieso ungenau – warum also dafür einen so großen Aufwand spendieren."

„Wir wissen über die derzeitigen Verzögerungen Bescheid, holen den Verzug aber bis Projektende wieder auf."

In diesem Zusammenhang muss auf die Gestaltung des Anreizschemas hingewiesen werden, das in den meisten Fällen den ausschlaggebenden Beitrag zum richtigen Einsatz von Schätzverfahren liefert. Hinzu kommen noch folgende Anforderungen, denen das Verfahren genügen muss:

- Genauigkeit innerhalb einer akzeptablen Grenze
- Eindeutigkeit (Reproduzierbarkeit)
- Anwendbarkeit in frühen Phasen des Projekts
- Stabilität der Schätzfunktion
- Benutzerfreundlichkeit
- Transparenz um das Ergebnis verifizieren zu können

7.1.1 Einflussfaktoren der Aufwandsschätzung

In einer ersten, sehr primitiven Näherung, kann ein Schätzmodell auf die Faktoren: Software-Umfang und Mitarbeiter-Produktivität reduziert werden. Da jedoch beide Faktoren mit relativ großen Streuungen behaftet sind, empfiehlt sich eine detailliertere Betrachtung. In Abb. 7.1 [1]) werden bereits 4 Faktorengruppen zur Quantifizierung der Software-Entwicklung herangezogen.

Die grau schraffierte Fläche kennzeichnet die aufgrund der Teamgröße zur Verfügung stehende Arbeitskapazität, die zielgerichtet eingesetzt werden kann. Die an den Ecken genannten Faktoren (Ziele) konkurrieren um die verfügbaren Ressourcen. Somit impliziert die Verbesserung eines der 4 Ziele einen schlechteren Erfüllungsgrad der anderen Ziele.

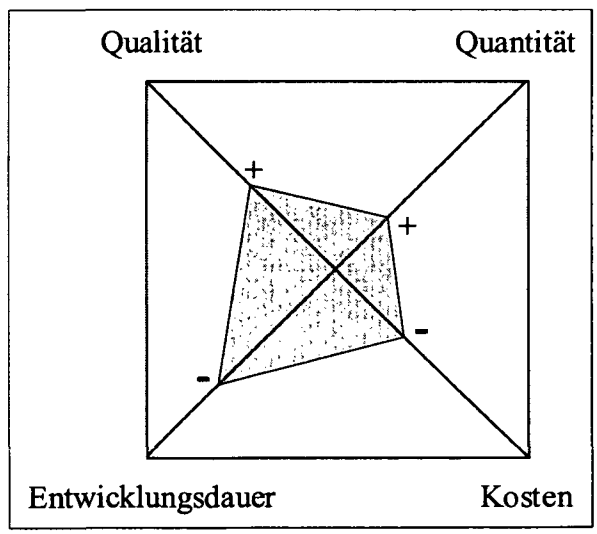


Abb. 7.1: „Teufels-Quadrat"

Es ist zu beachten, dass aufgrund der Lernkurve eine kurzfristige Erhöhung der Entwicklungskapazität unmöglich ist. Wie im Kapitel 11 gezeigt, ist dies nicht nur durch begrenzte technische Möglichkeiten, sondern vielmehr durch begrenzte organisatorische

[1] Sneed, H. M., Software-Management, Köln, 1987

Möglichkeiten bedingt. Mit der Einhaltung der Größe des „Teufels-Quadrat" sind dann auch die **Kosten** beschränkt bzw. gegeben.

Die **Quantität** umfasst Aussagen über den Umfang und die Komplexität der Software. Je nach Programmiersprache und Tools variieren diese Größen. Zudem wächst die Komplexität mit zunehmender Größe aufgrund der wachsenden Anzahl an Schnittstellen innerhalb des Codes. Um diesbezüglich eine neutrale Bewertung zu ermöglichen wird in neueren Schätzverfahren die Funktions- und Datenstruktur als Maß für den Umfang herangezogen (z.b. Function-Point-Verfahren). Damit wird zudem eine frühzeitige Abschätzung begünstigt.

Maßzahlen über die **Qualität** kommen auf sehr unterschiedliche Weise zustande. Sie reichen von der Bewertung eines kompletten Arbeitsumfeldes (z.b. Capability Maturity Model) bis zum Zählen der Fehlermeldungen (Errors) beim Testen. Zur **Entwicklungsdauer** ist anzumerken, dass sie nicht beliebig reduzierbar ist, da hierzu mehr Personalressourcen notwendig sind welche ihrerseits einen höheren Kommunikations-Aufwand zur Folge haben.

7.1.2 Klassifikation der Schätzverfahren

Aufgrund der raschen Entwicklung von Software-Entwicklungs-Methodiken existiert auch eine Vielzahl von Schätzverfahren, die aufgrund ihrer wesentlichen Merkmale in Gruppen zusammengefasst werden können. Hans-Dieter Litke empfiehlt folgende Gliederung [2]):

- Nach den Phasen, die abgedeckt werden
  - o für Einzelaktivitäten
  - o für die Programmierung
  - o für Detailentwurf und Programmierung
  - o für den gesamten Entwicklungsprozess
- Nach der theoretischen oder praktischen Absicherung, d.h.
  - o unternehmensspezifische Verfahren
  - o unternehmensunabhängige Praxisverfahren
  - o wissenschaftlich fundierte Verfahren
- Nach dem Verwendungszweck, d.h.
  - o zur Unterstützung der Investitionsentscheidungen (z.b. Kalkulation der Softwarekosten, Kosten-/Nutzenanalyse, Eigen-/Fremdentwicklung)
  - o zur Ermittlung von Plangrößen für das gesamte Projekt (z.b. Kapazitätsplanung, Terminplanung, Finanzplanung)
  - o für Abschnitte des Projekts

[2] Litke, Hans-Dieter, DV-Projektmanagement, München, 1996

In Anlehnung an die praktische Handhabung empfiehlt sich jedoch eher eine erste Unterteilung nach dem Aufwand, der in ein Schätzverfahren zu investieren ist. Demzufolge können die Schätzverfahren in

1. Expertenschätzungen
2. Faktormethoden
3. Umfassende Modelle

unterteilt werden, wobei der Aufwand beim Einsatz umfassender Modelle am größten ist.

7.1.2.1 Expertenschätzungen

Hierunter versteht man die Schätzung durch einen erfahrenen Mitarbeiter. Eine Erhöhung der Genauigkeit wird durch mehrere abhängige oder unabhängige Expertenmeinungen erreicht. Diese vielfach in der Literatur abgelehnte Methodik erweist sich in der Praxis als wertvolle und keinesfalls minderwertige Methodik der Aufwandsermittlung. Bedingung hierfür ist jedoch eine längere Erfahrung mit vergleichbaren Software-Projekten. Dies wird in den Software erstellenden Betrieben durch eine geeignete Aufbauorganisation sehr oft begünstigt.

Eine aufgrund der Art des Zustandekommens der Schätzung verwandte Methode ist die **Analogiemethode**. Hierbei vergleicht ein Experte – ohne auf Faktoren, wie sie im nächsten Abschnitt beschrieben werden, näher einzugehen – ein neues Projekt mit einem bereits abgeschlossenen. Eine Formalisierung dieses Vorgehens wird in neuerer Zeit durch die Erkenntnisse der Mustererkennungstheorie unterstützt. Im Zusammenhang mit der Zunahme der benutzen Formalismen spricht man dann auch von der **Relationenmethode**.

Bezüglich der Abgrenzung der Expertenschätzung zu anderen Verfahren ist insbesondere der Grad der Formalisierung maßgebend. In der Praxis spricht man meistens dann von einer Expertenschätzung, wenn über die Methode der Schätzung keine weiteren Informationen existieren.

7.1.2.2 Faktormethoden

Im folgenden werden verschiedene Ansätze vorgestellt auf deren Basis Verfahren zur Aufwandsschätzung entwickelt werden können. Aufgrund der inzwischen bereits existierenden und erprobten Modelle – wie sie im nächsten Punkt vorgestellt werden – finden nunmehr Eigenentwicklungen von Schätzverfahren nur noch in Ausnahmefällen statt [3]).

[3] Tavolato, Paul, Software-Projekt-Management, Wien, 1993

- **Parametrische Schätzgleichungen:**
  Dies ist ein sehr allgemeines Prinzip, das von einer gewissen Stabilität des Projektumfelds ausgeht. Des weiteren muss bereits eine größere Anzahl von abgeschlossenen Projekten mit den dafür charakterisierenden Kenngrößen vorliegen. Nun können aufgrund von Korrelationsanalysen der Kenngrößen mit den Projektaufwänden (der vergangenen abgeschlossenen Projekte) die für den Aufwand relevanten Faktoren extrahiert und in einer Gleichung formuliert werden. Obwohl dieses Prinzip mathematisch gut formulierbar ist, hat es den Nachteil der fehlenden Hilfestellung bei der Identifikation von Kenngrößen.

- **Gewichtungsmethode:**
  Diese Methode baut auf der zuvor beschriebenen auf, wobei die Faktoren der Aufwandsgleichung durch eine Gewichtung anhand der Anforderungen an ein konkretes Software-Projekt (im Vergleich zum fiktiven Standard-Projekt) abgeändert werden.

- **Arbeitspaket-basierende Methode (Aufwand-pro-Einheit- bzw. Multiplikatormethode):**
  Wie bei der Ermittlung des Leistungsstandes gezeigt, ermöglicht diese Methode zusätzlich zur Aufwandsermittlung eine Beurteilung des Fortschritts. Kennzeichnend ist die zu Beginn des Verfahrens erfolgte Gliederung des Projekts in Arbeitspakte, welche bestimmten Phasen zugerechnet werden. Anschließend können die unterschiedlichsten Verfahren zur Aufwandsermittlung der Arbeitspakete angewandt werden.

- **Bottom-up-Methode:**
  In frühen Phasen großer Projekte (mehr als 25 Programmierer) kann zum Zweck einer ersten Abschätzung eine repräsentative Teilaufgabe (bzw. die aufwändigste Teilaufgabe) analysiert und die Aufwände auf das Gesamtprojekt hochgerechnet werden. Diese Methode ist äquivalent zur herkömmlichen Zuschlagskalkulation, wobei die Personalkosten die Zuschlagsbasis bilden. Eine Variante dieses Verfahrens ist die **Prozentsatzmethode**, bei dem vom Aufwand einer Phase auf die restlichen geschlossen wird. Dieses Verfahren kann auch während des Projektverlaufs angewandt werden, indem von den bereits abgeschlossenen Phasen auf den Gesamtaufwand geschlossen wird.

7.1.2.3 Umfassende Modelle

Die Tabelle 7.1 gibt einen Überblick über die in der Literatur erwähnten Modelle [4]). Da der Einsatz der Methoden in der Praxis längere Zeit in Anspruch nimmt, sollte davon ausgegangen werden, dass das eine oder andere Verfahren auch derzeit (im Jahr 2002) in der einen oder anderen Software-Entwicklungs-Organisation noch in der Erprobungsphase ist.

[4] Noth, T., Kretzschmar, M., Aufwandsschätzung von DV-Projekten, Berlin, 1986

Schätzmethode	*Erschei-nungsjahr*	*Zugrundeliegende Basismethode*	*Einsatz-zeitpunkt*	*Berücksichtigte Faktorengruppe*
SDC	1967	Ps	P	1
IBM-Handbuch	1968	G, P	D	1, 4
SURBOCK	1978	G, P	P, D	1, 2, 4
ARON	1969	M, P	P	1, 2, 3, 4
T.O.P.	1971	A, G	P, D, E	1, 2, 3
Shell	1972	G, P, Ps	D	1, 4
Wolverton	1974	M	P	1
SLIM	1974	Ps	P, D, E	1, 4
FUTH	1975	G, P	D	1, 4
Software-Part	1976	A	P	1, 2, 4
EGW	1977	G	P, E	1, 4
Boeing	1977	G, Ps, M	P, E	1, 4
IFA-PASS	1977	A, P	P	1, 2, 4
DOTY	1977	G, Ps	P	1, 2, 4
GRIFFIN	1977	G, P	P	1, 4
Schneider	1978	Ps	P	1
INVAS	1980	R, G	I	1, 2, 3, 4
ZKP	1980	G, P	D	1, 4
COCOMO	1981	G, Ps	P	1, 2, 3, 4
Function Point	1981	A, G	I	1, 2, 3, 4

Legende

Zugrunde liegende Basismethode:	A = Analogiemethode M = Multiplikatormethode R = Relationenmethode G = Gewichtungsmethode Ps = Parametrische Schätzgleichung P = Prozentsatzmethode
Einsatzzeitpunkt:	P = Planungsphase D = Definitionsphase E = Entwurfsphase I = Iteratives Verfahren
Berücksichtigte Faktorengruppe:	1 = Quantität 2 = Qualität 3 = Entwicklungsdauer 4 = Produktivität

Tab. 7.1: Schätzmethoden

7.1.3 Die Function-Point Methode

Die Function-Point Methode ist ein mittlerweile weit verbreitetes Verfahren zur Aufwandsschätzung in der Software-Entwicklung. Sie basiert auf der vom Kunden geforderten Software-Funktionalität. Dies bedingt, dass bereits zum Zeitpunkt der Schätzung ein klares Bild

über die Nutzung der Software vorliegt. Auf Basis des Spiralmodells muss zumindest das Ergebnis eines Zykluses gegeben sein. Die Schätzung selbst wird mit Beteiligung des Kunden, bzw. jemandem der über die Funktionalität der Software Bescheid weiß, und eines Schätzexperten vorgenommen. Die Aufgabe der Schätzexperten ist die korrekte Anwendung vorgegebener Richtlinien bei der Beurteilung der Funktionalität bzw. Produktanforderungen. Von wesentlicher Bedeutung ist hierbei die über mehrere Projekte hinweg vergleichbare Bemessung der Teilaufwände in Form von Function-Points. Dies erfolgt unter Zuhilfenahme eines Schemas, das anhand der Tab. 7.2 erläutert wird.

Zunächst werden die Funktionen der zu schätzenden Software in die 5 Kategorien

1. Eingaben
2. Ausgaben
3. Abfragen
4. Referenzdaten
5. Datenbestände

eingeteilt (1. Schritt). Durch die Klassifizierung in die drei Stufen: „einfach", „mittel" und „komplex" mit ihren entsprechenden Gewichtungen werden die einzelnen Funktionen hinsichtlich des notwendigen Implementierungsaufwandes in Relation gesetzt (2. Schritt). Diese Einteilung erfolgt unter Zuhilfenahme von Richtlinien, Tabellen und Beispielen. Mit „Erg 1" ist die Summe der Function-Points ermittelt.

Im 3. Schritt werden alle sonstigen Einflussfaktoren ermittelt („Erg 2"). In Summe ergeben sie einen Wert zwischen 0 und 60 – im Durchschnitt somit 30, bzw. durch 100 dividiert: 0,3. Daraus folgt der in der Tabelle angeführte Summand: 0,7, womit bei einer durchschnittlichen Summe der Einflussfaktoren die normierte Summe der Einflussfaktoren den Wert 1 („Erg 3") liefert. Die Extreme betragen 0,7 bzw. 1,3. Mit diesem Wert wird die Summe der Function-Points („Erg 1") multipliziert, womit die bewerteten Function-Points („Erg 4") berechnet wurden. Bis hierher konzentrierte sich der Formalismus lediglich auf eine über die unterschiedlichen Projekte identisch gestaltete Abstraktion der Teilaufwände.

Im 4. Schritt wird die Produktivität des Software-Entwicklungs-Teams (bzw. der Firma) berücksichtigt. Hierzu werden aufgrund vergangener Projekte Relationen zwischen bewerteten Function-Points und tatsächlichen Aufwänden in Mann-Monaten hergestellt. Siehe hierzu die in die Literatur eingegangene Kurve der Firma IBM [5]) (vgl. Abb. 7.2). In einem letzten Schritt wird nach Fertigstellung der Software eine Ist-Kalkulation durchgeführt und in die Kurve als weitere Relation zwischen bewerteten Function-Points und tatsächlichem Aufwand aufgenommen.

[5] IBM, Die Function Point Methode, IBM Deutschland GmbH, 1985

Kategorie	Anzahl	Klassifizierung	Gewichtung	Zeilensumme
Eingaben		einfach	3	=
		mittel	4	=
		komplex	6	=
Ausgaben		einfach	4	=
		mittel	5	=
		komplex	7	=
Abfragen		einfach	3	=
		mittel	4	=
		komplex	6	=
Referenzdaten		einfach	5	=
		mittel	7	=
		komplex	10	=
Datenbestände		einfach	7	=
		mittel	10	=
		komplex	15	=
Summe				**Erg 1**
Einflussfaktoren (EF) (korrigieren den FP-Wert bis zu +/- 30%)		Verflechtung mit anderen Anwendungssystemen (0-5)		=
		Dezentrale Daten dezentrale Verarbeitung (0-5)		=
		Transaktionsrate (0-5)		=
		Verarbeitungslogik Rechenoperationen (0-10) Kontrolloperationen (0-5) Ausnahmeregelungen (0-10) Logik (0-5)		=
		Wiederverwendbarkeit (0-5)		=
		Datenbestandskonvertierungen (0-5)		=
		Anpassbarkeit (0-5)		=
Summe der 7 Einflüsse				**Erg 2**
Normierte Summe der EFs		Erg 3 = (Erg 2 / 100) + 0,7		**Erg 3**
Bewertete Function Points		Erg 4 = Erg 1 * Erg 3		**Erg 4**

Tab. 7.2: Function-Point Berechnungsformular

Obwohl die Function-Point Methode als die beste derzeit verfügbare Methode zur Schätzung kommerzieller Anwendungssysteme angesehen wird [6]), und als Standard-Methode in der Softwarebranche eingesetzt wird, birgt sie einige Nachteile:

- Die Schätzung erfolgt für ein Gesamtsystem. Ein Herunterbrechen auf Phasen ist nur mit der Prozentsatzmethode möglich.

[6] Kemerer, C.F., Reliability of Function Points Measurement – A Field Experiment, in: Communications of the ACM, Febr. 1993, p. 85-97

- Für objektorientierte Systeme ist die von Allan Albrecht (1979) ursprünglich entwickelte Schätzmethode zu sehr funktionsorientiert.
- Qualitätsanforderungen werden nur hinsichtlich einer für alle Projekte geltenden Durchschnittsqualität berücksichtigt.
- Es erfolgt keine Faktorisierung in die für die Produktivität ausschlaggebenden Faktoren.

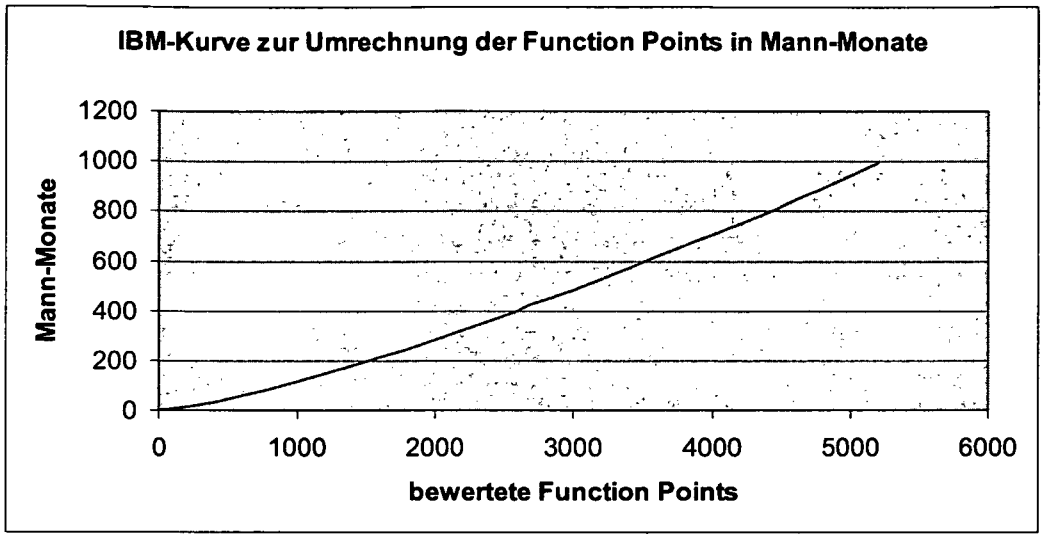


Abb. 7.2: IBM-Aufwandskurve

Ein wesentlicher Vorteil in der praktischen Anwendung ist die Automatisierbarkeit und weit verbreitete Anwendung in der Industrie. Durch die erst zum Schluss stattfindende Umrechnung der bewerteten FPs auf Mann-Monate ist zudem eine klare Trennung zwischen Firmen-unabhängigen und –abhängigen Einflüssen gewährleistet, was eine Weiterentwicklung der Methodik ohne die Preisgabe von Produktivitätskennzahlen (bewert. FPs <-> Mann-Monate) ermöglicht. Auf internationaler Ebene wird die Methode von der International Function Point Users' Group (IFPUG) vorangetrieben: „The International Function Point Users' Group (IFPUG) is a non-profit, member governed organization. The mission of IFPUG is to be a recognized leader in promoting and encouraging the effective management of application software development and maintenance activities through the use of Function Point Analysis and other software measurement techniques." [7])

7.1.3.1 Data-Point Methode

Eine Weiterentwicklung der Function-Point Methode ist die Data-Point Methode. Diese trägt dem Umstand Rechnung, dass in kommerzieller Software aufgrund der objektorientierten

[7] http://www.ifpug.org , 2004

Programmiersprachen die Datenelemente von zentraler Bedeutung sind. Genauer formuliert - die Informationsentitäten und die Nachrichten [8]). Die Informationsentitäten ergeben sich aus der Daten-, die Nachrichten aus der Kommunikations-Analyse. Die Nachrichten bezeichnen die Bildschirmmasken, Reports und Telegramme mit Hilfe derer die Informationsentitäten dargestellt werden. Ein großer Vorteil der Data-Point gegenüber der Function-Point Methode ist die Anwendbarkeit zu einem früheren Zeitpunkt, da hierfür keine zusätzliche Funktions-Analyse notwendig ist. Diese Methode eignet sich somit vorzüglich für die Entwicklung von Datenbanken, die zumindest in einer Sprache der 4. Generation programmiert sind.

Die Informationsentitäten werden nach folgenden Kriterien bewertet:

- Anzahl der Attribute
- Anzahl der Primär- und Sekundärschlüssel
- Integrationsgrad
- Nutzung als Ein- oder Ausgabe oder Ein- und Ausgabe
- zu welchem Prozentsatz das Objekt zu ändern ist

Die Nachrichten werden nach folgenden Kriterien bewertet:

- Anzahl der zu füllenden Felder
- Anzahl der Informationsobjekte, die in der Nachricht vertreten sind
- Komplexitätsgrad
- Ein- oder Ausgabe oder Ein- und Ausgabe
- zu welchem Prozentsatz das Objekt zu ändern ist

Nach Erstellung dieser Liste kann das Data-Point Verfahren automatisiert ablaufen. Anschließend an die Ermittlung der Summe der Data-Points aus den Informationsentitäten und den Nachrichten wird diese mit einem Qualitätsfaktor (= das Produkt der Qualitätsmerkmale) multipliziert. Als Qualitätsmerkmale stehen hierbei zur Verfügung:

- Benutzerfreundlichkeit
- Datenunabhängigkeit
- Effizienz
- Integrität
- Sicherheit
- Übertragbarkeit
- Wartbarkeit
- Zuverlässigkeit

Damit kann die Zahl der Data-Points um maximal +/- 50% erhöht bzw. reduziert werden. Des weiteren erfolgt eine Multiplikation des so erhaltenen Werts mit einem Projektbedingungsfaktor wodurch das Ergebnis um maximal 15% erhöht bzw. 25% reduziert wird. Der Projekt-bedingungsfaktor wird aus 10 Einzelfaktoren gebildet (Bewertung jeweils zwischen 1 und 5):

[8] Litke, Hans-Dieter, DV-Projektmanagement, München, 1996

- Projektverteilung (mehrere Länder, mehrere Orte, ein Ort, mehrere Räume, ein Raum)
- Projekterfahrung (keine bis sehr gut)
- Projektkenntnisse (keine bis sehr gut)
- Projektautomation (keine bis volle Automation)
- Rechenbedingungen (schlecht bis sehr gut)
- Projektunterstützung (keine bis sehr gut)
- Qualitätssicherung (keine bis vollautomatisch)
- Spezifikationsformalismen (informal, strukturiert, semiformal, formal, formal-grafisch)
- Programmiersprache (1. bis 5. Generation)
- Testautomation (keine bis 100%)

Nachdem nun äquivalent zu den bewerteten Function-Points auch die Data-Points aufgrund der Qualitätsanforderungen und der Projektbedingungen bewertet wurden, wird anhand einer Produktivitätstabelle der bewertete Data-Point Wert in eine Aufwandskennzahl (z.b. Mann-Monate) abgebildet.

7.1.4 Politische Einflüsse

Wie im Kapitel 11 dargestellt und von einigen Autoren, wie z.b. Tom DeMarco, eindrucksvoll beschrieben wird, haben politische Einflüsse große Auswirkungen auf den Erfolg von Software-Entwicklungsteams. Im Zusammenhang mit Schätzungen sollten folgende Patterns unbedingt berücksichtigt werden:

Die für die Schätzung verantwortlichen Personen sind organisatorisch unabhängig vom Entwicklungsteam zu positionieren. Damit soll eine mögliche Rückkopplung des Schätzergebnisses auf die mit der Schätzung betraute Person verhindert werden. Eine so zum Beispiel verhinderte Reaktion ist das Einrechnen von „Zeitpölstern". Diese werden zudem meistens willkürlich dimensioniert, wodurch der Schätzaufwand keine verwertbare Aussage wäre.

Um der Einflussnahme (von außerhalb des Schätzteams) in stärkerem Ausmaß zu begegnen sind zusätzlich zur formal-organisatorischen Unabhängigkeit Motivationsmechanismen für optimale Schätzergebnisse einzurichten. Dies ist insbesondere dann erforderlich, wenn die Schätzung mit Projektfortschritt verbessert werden soll und somit ein engerer Kontakt zum Entwicklungsteam unumgänglich ist.

Eine weitere Unterstützung für das kontinuierliche Verbessern von Schätzergebnissen wird mit einer geeigneten Teambildung erreicht. Hierzu sollte das Team aus mindesten 3 Personen bestehen, die durch untereinander geführte Diskussionen kontinuierlich an Erfahrung gewinnen.

7.2 Leistungsstand-Beurteilung

7.2.1 Einleitung

Die Leistungsstand-Beurteilung (= LSB) hat als Ziel die Formulierung einer Aussage über den Projektfortschritt. Hierdurch soll nicht nur die bereits geleistete Arbeit, sondern vielmehr die noch ausständige Arbeit, d.h. der Aufwand bis zur Fertigstellung eines Software-Projekts quantifiziert werden. Als Werkzeug zur LSB bedient man sich sogenannter Metriken, die im Grunde die Schlüsselfaktoren: Zeitaufwände und Kosten abbilden.

Die LSB von Softwareprojekten ist bis dato nicht standardisiert und zudem sehr komplex. Dies beruht einerseits auf der großen Vielfalt dessen, was einer Beurteilung zu unterziehen möglich ist, und zum anderen auf dem raschen Fortschritt der Software-Technologie. Eine zur LSB von Softwareprojekten äquivalente Situation findet man auch bei der Bilanzanalyse von Unternehmen vor. Diese dient der wirtschaftlichen Beschreibung eines Unternehmens zu einem bestimmten Stichtag im Jahr. Ähnlich den je nach Wahl der Grundsätze unterschiedlichen Vorschriften – z.b. HGB, US-GAAP, IAS [9]) - sind auch die Zielsetzungen für die LSB unterschiedlich. Als Motivation dienen u.a.:

- Projektcontrolling und Aufwandsschätzung mit Hilfe von Vergleichszahlen
  - zur Bestimmung sektorspezifischer Aufwands-Kennzahlen
  - zum allgemeinen Controlling laufender Projekte
- Produktivitätsmessung
  - zum Erkennen von Verbesserungspotentialen
  - zum Benchmarking
- Qualitätsmessung

Beginnt man nun damit, jedes dieser möglichen Ziele auf deren unabhängige Variablen zu reduzieren, erhält man jene Eigenschaften, die es im weiteren zu quantifizieren gilt. Somit ergeben sich als Durchschnittsmenge der Eigenschaften folgende sechs Schlüsselfaktoren für Softwareprojekte:

- Kosten
- Termine
- Verfügbare Ressourcen
- Qualität des Produktes
- Funktionalität des Produktes
- Performanceverhalten des Produktes

[9] Dangel, P., Hofstetter, U., Otto, P., Analyse von Jahresabschlüssen nach US-GAAP und IAS, 2001

Ähnlich dem betrieblichen Rechnungswesen, das sowohl firmeninterne als auch firmenexterne Adressaten hat, verhält es sich auch mit den Kennzahlen über ein Softwareprojekt. Aus Kundensicht ist die LSB insofern bedeutsam, da eine große Anzahl von Softwareprojekten aufwandsbezogen abgerechnet werden. Für diese verlangt dann der Kunde ein Werkzeug, um den geleisteten Aufwand selbst beurteilen zu können.

Die Problemstellung der LSB beginnt mit der fehlenden Existenz eines geeigneten Maßes. Abgesehen davon, dass der Prozess der Software-Entwicklung ohne Hilfsmittel der Wahrscheinlichkeits-Theorie nicht formulierbar ist, basieren auch die nach außen hin (z.b. gegenüber dem Kunden) getätigten Aussagen auf Größen, die den Projektfortschritt nur indirekt charakterisieren. Ähnlich wie der Fortschritt eines Künstlers von nahezu unendlich vielen Parametern bestimmt wird, die zu Projektbeginn nicht vollständig planbar sind, ist auch das Softwareprojekt von vornherein nicht vollständig formulierbar. Einen Ausweg zur Messung des Fortschritts bieten Surrogate, die auf indirektem Wege Aufschluss über den Projektfortschritt geben. Mathematisch formuliert, stellen sie eine positive Korrelation zur tatsächlich gesuchten Größe her.

Das Modell zur LSB baut auf den derzeit allgemein akzeptierten Grundsätzen zur Software-Entwicklung auf [10],[11]. Hiernach sind sowohl die Grundsätze des Projektmanagements als auch jene des Prozessmanagements von vorrangiger Bedeutung. Die Nennung des Begriffs Projekt [12] unterstreicht die Teilung der Gesamtaufgabe in einzelne Arbeitspakete. Demgegenüber unterstreicht der Begriff Prozess das Know How, um die Arbeitsschritte optimal auszuführen und zu koordinieren. Die Reihenfolge der Arbeitsschritte wird durch Phasen gekennzeichnet, wobei jeder Phase mehrere Arbeitspakete zugewiesen werden.

7.2.1.1 Modell-Strukturierung

Grundlage der Modell-Strukturierung sind Phasenablaufmodelle, wie sie bereits in einem vorangegangenen Abschnitt beschrieben wurden [13]. Unabhängig vom gewählten Phasen-Ablaufmodell werden die Phasen und die ihnen zugerechneten Arbeitspakete in einer bestimmten Reihenfolge abgearbeitet. Lässt es die Abhängigkeit zu – und dies stellt den Normalfall dar – werden manche Arbeitspakete parallel durchgeführt. Dieser Vorgang der Parallelisierung stellt überhaupt erst den Vorteil der Software-Entwicklung in Teams dar, wodurch die Zeitspanne bis zur Fertigstellung möglichst minimiert wird.

---

[10] Balzert Helmut, Lehrbuch der Software Technik: Software-Entwicklung, Heidelberg, 1996
[11] Cusumano, Michael A., Harvard Business Manager 3/98: Gipfelstürmer Microsoft, Massachusetts, 1998, p.55-66
[12] Patzak, Gerold, Rattay, Günter, Projekt Management, Wien, 1998
[13] Siemens AG Österreich – Intranet, SEM Softwareentwicklungsmethoden, Copyright © Siemens AG Österreich 1997

Um den Fortschritt zu dokumentieren müssen zu bestimmten Zeitpunkten Aussagen über den Fertigstellungsgrad der Software-Entwicklung bzw. der Phasen oder Arbeitspakete möglich sein. Als einfachste Lösung wäre hierbei die Expertenschätzung zu nennen, die den Fertigstellungsgrad als Prozentsatz zum Gesamtprojekt-Aufwand angibt. Sie kann bei Ein-Mann-Projekten durchaus Sinn machen, falls einschlägige Erfahrungen bereits existieren. Die prozentuelle Beurteilung sollte sich hierbei zumindest auf

- Inhalt,
- Aufwand (Mann-Monate (proportional zu den Kosten)), und
- Termin

beziehen. Die Angabe einer Grenze, ab welcher eine umfangreichere Messung (hinsichtlich der Anzahl an Parametern) erfolgen sollte, ist von der konkreten Projektsituation abhängig. So kann z.b. beim Programmieren von Routineaufgaben ein genaueres Controlling schon bei geringerem Aufwand durchaus wünschenswert sein. Dies zum Beispiel insbesondere dann, wenn die erwartete Gewinnspanne von vornherein niedrig kalkuliert ist. Größere Projekte (bezüglich Aufwand) sollten zwangsläufig einem detaillierten Projekt-Controlling unterliegen, dem messbare Größen zugrunde liegen (durch das Adjektiv „messbar" soll insbesondere die Nachvollziehbarkeit von ermittelten Ergebnissen betont werden).

7.2.2 Fallstudie zur Leistungsstand-Beurteilung

Die nachstehende Skizze (Tab. 7.3) gibt einen schematischen Überblick über den Zusammenhang zwischen Phasen (Phase 1, Phase 2, Phase 3), Arbeitspaketen (AP1, AP2, AP3, usw.) und Zeitpunkten (t1, t2, t3, usw.), zu denen der Fortschritt beurteilt wird. Die Start-Termine der Arbeitspakete sind dabei entweder aufgrund von Abhängigkeiten mit vorangehenden Arbeitspaketen oder aufgrund der begrenzen Ressourcen gesetzt.

Plandaten					
t1	t2	t3	t4	t5	t6
Phase 1	Phase 2				Phase 3
AP1					
	AP2				
		AP3			
			AP4		
				AP5	

Tab. 7.3: Phasen

Zum Zeitpunkt t3 sollte somit lt. Tabelle 7.3 das Arbeitspaket AP1 zu 100%, AP2 zu 66%, und AP3 zu etwa 33% fertiggestellt sein. Um die 100%-Marke von Fortschritts-Metriken festzulegen dienen projektbezogene Fertigstellungs-Planwerte, wobei die Kriterien für die 100%ige Fertigstellung eines Arbeitspakets zuvor definiert worden.

Gegebenenfalls werden die Planwerte von Phase zu Phase aktualisiert. Ein möglicher Grund hierfür wären z.b. Korrekturen im Einverständnis mit dem Auftraggeber. Anhand Tab. 7.4 wird beispielhaft der Leistungsstand des Arbeitspakets (AP2) ermittelt. Zum Stichtag (z.b. t3) werden alle laut Plan in den betreffenden Zeitraum (in unserem Fall: t2-t3) fallenden Arbeitspakete (hier: AP2, AP3) auf ihren Ist-Fortschritt hin bewertet:

AP2 (gegliedert in 5 LM)	**SOLL-Zustand** **(bezogen auf Stichtag, t3)**	**IST-Zustand** **(bezogen auf Stichtag, t3)**
Inhalt, materiell (objektiv gemessen)	AP2 zu 66% fertig für Start von AP3 muss AP2 zu 33% fertig sein für Start von AP4 muss AP2 zu 66% fertig sein (A1)	AP2 zu 50% fertig o.k. not o.k. (A2)
Inhalt, immateriell (subjektiv geschätzt)	AP2 zu 80% fertig (geschätzt vor Projektbeginn) (B1)	AP2 zu 70% fertig (geschätzt zu t3) (B2)
Termin der Fertigstellung und Abhängigkeiten	17. Juni 1998 (C1)	+ 2 Wochen bei gK; zusätzliche Verzögerung von AP5 + 5 MM bei gT der Fertigstellung (C2)
Aufwand	34 MM bis Fertigstellung (D1)	25 MM (= 74% von 34 MM); dadurch prognostizierter Gesamt-aufwand: 50 MM (D2)

Tab. 7.4: Leistungsstand

Legende: AP ... Arbeitspaket
LM ... Leistungsmerkmal
MM ... Mann-Monate
gK ... geplante Kapazität (in MM)
gT ... geplanter Termin

Erläuterungen zu Tab. 7.4:

Die Kriterien, nach denen der Leistungsstand des Arbeitspakets bewertet wird, sind:

- Inhalt, materiell (objektiv gemessen)
- Inhalt, immateriell (subjektiv geschätzt)
- Termin der Fertigstellung, sowie Abhängigkeiten
- Aufwand bzw. Kosten

zu (A1), (A2):

Die Messung des AP-Fortschritts setzt geeignete Metriken voraus, die sich am AP-Inhalt orientieren. In vielen Fällen werden die Arbeitspakete weiter in Leistungsmerkmale unterteilt und mit 0% oder 100% Erreichungsgrad bewertet (vgl. hierzu Kapitel 8).

zu (B1), (B2):

Eine subjektive Schätzung kann neben einer Expertenmeinung auch die Stimmung im Team widerspiegeln, deren Ursachen so vielfältig und gleichzeitig schwerwiegend sein können, dass sie eine objektiv ermittelte Messung sinnvoll ergänzen kann. Auch ist sie zu einer ersten Orientierung praktisch unverzichtbar.

zu (C1), (C2):

Mit der Planung der Termine ist auch auf die Mannstärke Rücksicht zu nehmen. Ausgedrückt wird dies durch die inverse Beziehung von Zeitspanne bis Fertigstellung und Personal-Aufwand (dargestellt u.a. durch die Einheit: Mann-Monat). Dabei ist der nur degressiv wachsende Sach-Fortschritt mit größer werdenden Teams zu beachten.

zu (D1), (D2):

Der erbrachte Aufwand (angefallene Kosten) kann relativ einfach über eine Stunden-Kontierung ermittelt werden. Eine Bestimmung des zukünftig noch erforderlichen Aufwands orientiert sich jedoch auch an vorher erwähnten Größen.

Der Unterschied zwischen einer objektiven Messung und einer subjektiven Schätzung besteht in der Nachvollziehbarkeit. Die objektive Messung muss jederzeit für jedermann nachvollziehbar sein, während die subjektive Schätzung auf den nicht näher zu erläuternden Erfahrungen einer Einzelperson oder Gruppe beruht. Der im Feld (A1) angeführte Prozentsatz von 66% (1. Zeile) ergibt sich aus der zugrundegelegten Metrik (in diesem Beispiel nicht näher bestimmt). Um mit Hilfe einer Prozentsatz-Angabe eine sinnvolle Aussage treffen zu können, wird eine weitgehend lineare Abhängigkeit zwischen dem zu bewertenden Inhalt und jener Hilfsgröße, anhand derer der inhaltliche Fortschritt gemessen wird, vorausgesetzt. Eine eventuell im vorhinein bekannte Nichtlinearität kann durch geeignete mathematische Transformationen bereinigt werden. Obwohl diese Methode theoretisch zum Ziel führt, ist sie jedoch praktisch kaum handhabbar.

7.2.2.1 Verfahren der kleinsten Arbeitspakete nach Exl

Eine einfachere Alternative ist eine von vornherein aufwandslineare Unterteilung in kleinste Einheiten (z.b. Leistungsmerkmale) derart, dass eine Messung auf die Ermittlung einer 0%igen oder 100%igen Fertigstellung dieser kleinsten Einheiten reduziert werden kann. Die termin- oder auch aufwands-äquivalente Unterteilung unterliegt dann jedoch einer meist subjektiven Bewertung. Diese Methode ist vergleichbar der Anwendung des Systems der „Finiten Elemente" (gegenüber einer analytischen Beschreibung).

Die Angabe von „für Start von AP3 muss AP2 zu 33% fertig sein" berücksichtigt die Abhängigkeiten von Programmteilen zwischen Arbeitspaketen. Demnach muss also AP2 nicht unbedingt zur Gänze fertig sein, um – so wie in diesem Beispiel – mit AP3 beginnen zu können. Unter Zuhilfenahme der Leistungsmerkmale könnte dies z.b. auch durch die 100%ige Fertigstellung der Leistungsmerkmale LM1 und LM2 gekennzeichnet sein, was die Angabe einer Prozentzahl (hier 33%) ersetzt.

Im Feld (A2) wird anhand der zuvor definierten Metrik für (A1) der tatsächliche Leistungsstand zum Zeitpunkt t3 angegeben. Es gelten hierbei sämtliche schon zu (A1) angeführten Bemerkungen. "o.k." und "not o.k." beziehen sich auf den Start von nachfolgenden Arbeitspaketen (in unserem Beispiel kann mit AP3 bereits begonnen werden, obwohl AP2 noch nicht fertiggestellt wurde).

Die in den Feldern (B1) und (B2) geschätzten Werte (zu Beginn und zum Zeitpunkt t3) können von den nach einem objektiven Kriterium bestimmten Werten abweichen und sollten daher gegebenenfalls in Diskussion gestellt werden.

Im Feld (C1) steht der Termin der geplanten Fertigstellung des kompletten Arbeitspakets AP2. Aufgrund des Zeitverzugs (AP2 ist zu t3 tatsächlich nur zu 50% realisiert) werden im Feld (C2) zwei Alternativen zur Fertigstellung angegeben. Die erste Möglichkeit wäre eine Verzögerung (hier 1 Woche) bei gleichbleibender Kapazität (Personal/Tag). Als Alternative dazu könnte jedoch auch die Kapazität erhöht werden. Dies hat zudem den Vorteil, dass es außer den schon verzögert begonnenen APs 3 und 4 nicht auch noch zu einer Verzögerung von z.b. AP5 kommt (AP5 würde normalerweise nach Ende von AP2 starten).

In (D1) ist der geplante Gesamtaufwand für AP2 angegeben. In (D2) ist der bis zum Zeitpunkt t3 verbrauchte Aufwand (hier 74% vom geplanten) angegeben. Geht man von einem auch weiterhin konstanten Ressourcen-Einsatz aus (Effizienz-Degression bei größeren Teams), und nimmt weiters eine Proportionalität zum Zeitaufwand vorweg, ergibt sich ein prognostizierter Gesamtaufwand von 50 MM.

7.2.2.2 Verdichtung der Fortschritts-Daten

Die bis jetzt durchgeführten Überlegungen müssen für alle Arbeitspakete des Projekts dokumentiert werden, wodurch eine Verdichtung der Daten unerlässlich wird. Während bei kleineren Projekten durch eine geeignete Zusammenfassung der Daten dies noch einfach durchführbar ist, kann eine übersichtliche Darstellung bei umfangreichen Projekten nur durch intelligentes Komprimieren der Aufwands-Daten bewerkstelligt werden. Dies wird z.B. durch eine Abbildung der unterschiedlichen Maßzahlen (Zeit, Aufwand in MM, usw.) auf eine einheitliche Größe erreicht. Auf diese Weise können die Fertigstellungsgrade der einzelnen Arbeitspakete in Aufwände (MM) konvertiert und den Planzahlen gegenübergestellt werden. Zur Illustration ein Beispiel [14]):

Arbeitspaket	**Planaufwand**	**Inhalt**		**aufgelaufener Aufwand**
AP101	3 MM	30% fertig	->	0,9 MM
AP102	4 MM	70% fertig	->	2,8 MM
AP103	7 MM	40% fertig	->	2,8 MM
Projekt	14 MM			6,5 MM -> **zu 46% fertig**

Es werden hier also die inhaltlichen Fertigstellungsgrade über den Umweg der Aufwände zu einem Gesamtfertigstellungsgrad konvertiert.

7.2.2.3 Meilenstein-Trendanalyse zur Leistungsstand-Visualisierung

Diese Form der Visualisierung informiert das Management über Verzögerungen von Projektphasen. Graphisch wird die Meilenstein-Trend-Analyse (MTA) als Dreiecksraster mit den Achsen Planungs- und Berichtszeitraum konstruiert. Da die Phasenabschnitte meist überlappt realisiert werden und somit Abhängigkeiten untereinander bestehen, kann bereits ein einziger nicht planmäßig beendeter Phasenabschnitt den Gesamtprojekt-Fertigstellungstermin gefährden – dessen Visualisierung Zweck der MTA ist. Abgesehen von den Verzögerungen einzelner Abschnitte, können aufgrund andauernder Terminüberschreitungen die Planung und Aufwandsabschätzung insgesamt in Frage gestellt werden.

In den nachfolgenden Abbildungen sind zwei Beispiele aus der Praxis dargestellt. Während das Software-Projekt aus Abb. 7.3 im Laufe eines Jahres eine Gesamtverzögerung von 3 Monaten (B600 – Ende des Systemtests, d.h. die Software ist bereit zur ersten Auslieferung) erfährt, kam es beim anderen Projekt entsprechend der Abb. 7.4 bereits beim ersten Meilenstein (B130 – Ende der Definitionsphase) zu einer Terminüberschreitung von 100%.

[14] Burghardt, Manfred, Einführung in Projektmanagement, Publics-MCD-Verl., München, 1995

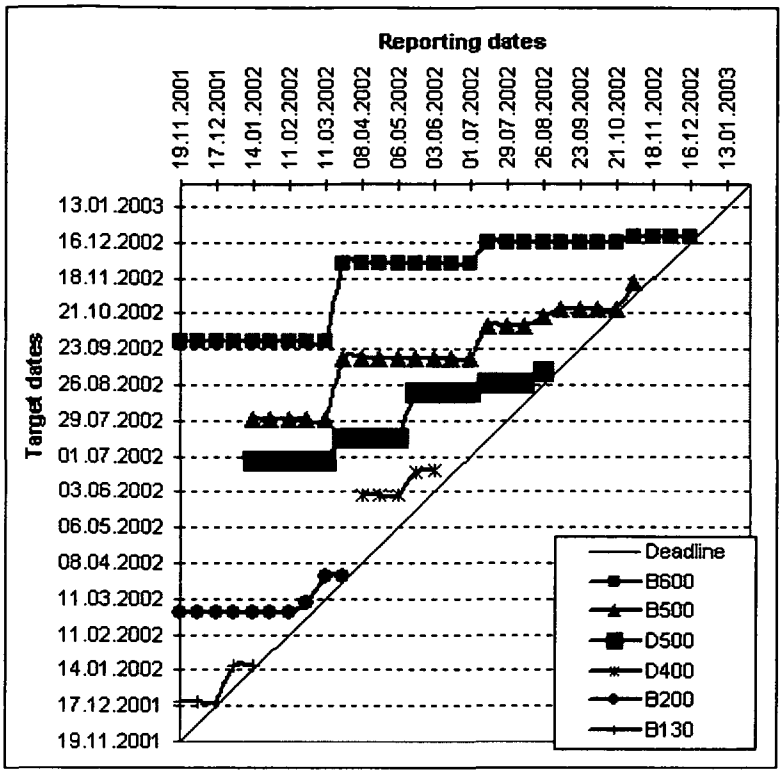


Abb. 7.3: Meilenstein-Trend-Analyse 1

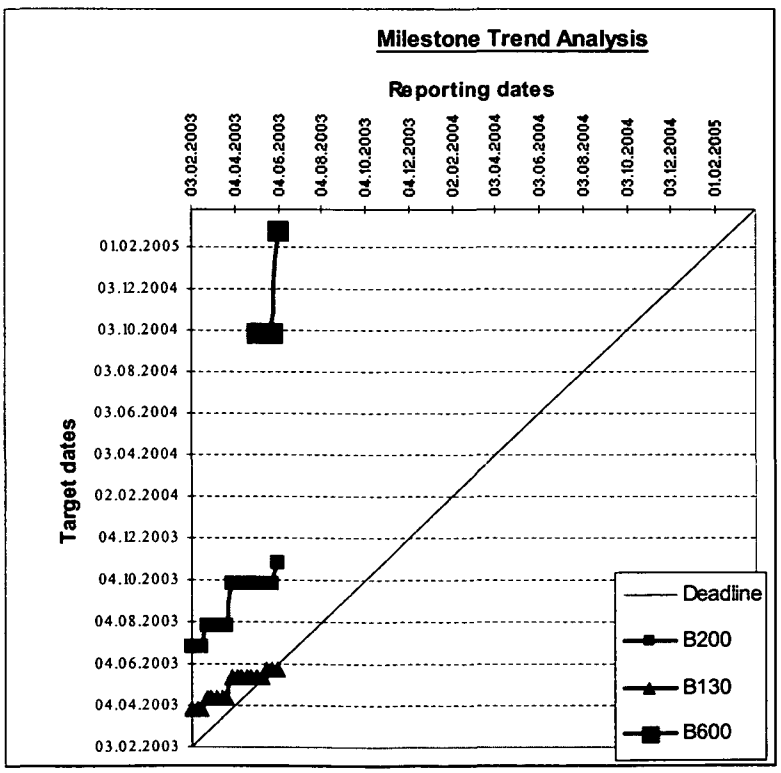


Abb. 7.4: Meilenstein-Trend-Analyse 2

7.2.2.4 Dilemma der Leistungsstand-Beurteilung

Wie an der bisherigen Diskussion ersichtlich, ist eine direkte Bewertung der Zielgröße, also eine Messung des Fortschritts selbst, bis auf tautologisch gestaltete Fälle, nicht möglich. Schlussendlich existiert auch kein Maß, den Fortschritt während der Software-Entwicklung direkt zu messen.

Unter Zuhilfenahme der Ausdrucksweise der theoretischen Modell-Bildung gelangt man zum Dilemma der Messung von Lösungsfindungs-Intelligenz. Dies ist jene Fähigkeit, derer man sich zum Lösen eines Problems bedient – hier der Formulierung von Quellcode welcher die Anforderungen an die Software erfüllt. Demzufolge sind mathematisch-logische Modelle zwar imstande, ein Problem als:

- lösbar
- unlösbar
- unentscheidbar
- usw.

zu klassifizieren, jedoch nicht schon im vorhinein den zur Bestimmung dieser Aussage notwendigen Aufwand anzugeben. - Dies würde nämlich eine Lösung bereits voraussetzen!

7.3 Produktivität

Unter der Produktivität wird allgemein das Verhältnis von Output zu Input verstanden. Der Input für die Software-Entwicklung lässt sich – auf den ersten Blick - einfach bewerten. Er setzt sich im wesentlichen aus den 3 Kostenblöcken

- Personalkosten,
- Kosten für Hard- und Software, und
- Kosten für Hilfsmittel

zusammen.

Schwieriger ist dagegen die Definition des Outputs. Eine Definition, die auf dem erzielten Preis basiert, und folglich ebenfalls einfach zu ermitteln ist, enthält zu viele Faktoren, um die Produktivität der Software-Entwicklung sinnvoll beziffern zu können. So werden in der Praxis viele Software-Projekte schon zwischendurch abgebrochen, oder deren damit erzielte Einnahmen durch die Marktverhältnisse stark beeinflusst. Ein erster Ansatz wäre die Quantifizierung der produzierten Software selbst – z.b. durch die Anzahl der „Lines of Code".

Eine äquivalente Argumentation auf der Inputseite führt zu dem Resultat, dass z.b. auch die Personalkosten keine geeignete Metrik für den Input sind. Abgesehen von der in der Praxis kaum bestehenden Korrelation von Produktivität und Einkommen, ist ein wesentlicher Faktor der individuellen Produktivität der von Tom DeMarco angeführte Umweltfaktor [15]) (vergleiche hierfür den Abschnitt über Managementeinflüsse).

Um Aussagen zur Produktivitätsverteilung bei Software-Entwicklern zu treffen, führen Tom DeMarco und Timothy Lister seit 1977 weltweit Studien in Form von Wettbewerben durch. Diese Studien tragen den Namen „Kriegsspiele für Programmierer". Die Bedingungen für die Teilnehmer gestalten sich folgendermaßen:

Bei der Auswahl der Teilnehmer wird darauf Wert gelegt, dass jeweils zwei Programmierer aus dem gleichen Arbeitsumfeld innerhalb der gleichen Firma kommen. Allen Teilnehmern wird die gleiche Aufgabe gestellt – bei vorgegebener Spezifikation ein Programm zu entwerfen, zu programmieren und zu testen. Die dafür benötigten Zeiten werden notiert und die Programme einem zentralen Abnahmetest unterworfen. Die Teilnehmer arbeiten in ihrer vertrauten Arbeitsumgebung, während der normalen Arbeitszeit, mit ihren gewohnten Programmiersprachen, ihren Werkzeugen und ihren Rechnern – wie sie es bei allen anderen Projekten gewohnt sind.

Zwischen 1984 und 1986 nahmen über 600 Programmierer aus 92 Firmen teil. In Summe war der Zeitaufwand für Entwurf, Codierung und Test wie in Abb. 7.5 verteilt:

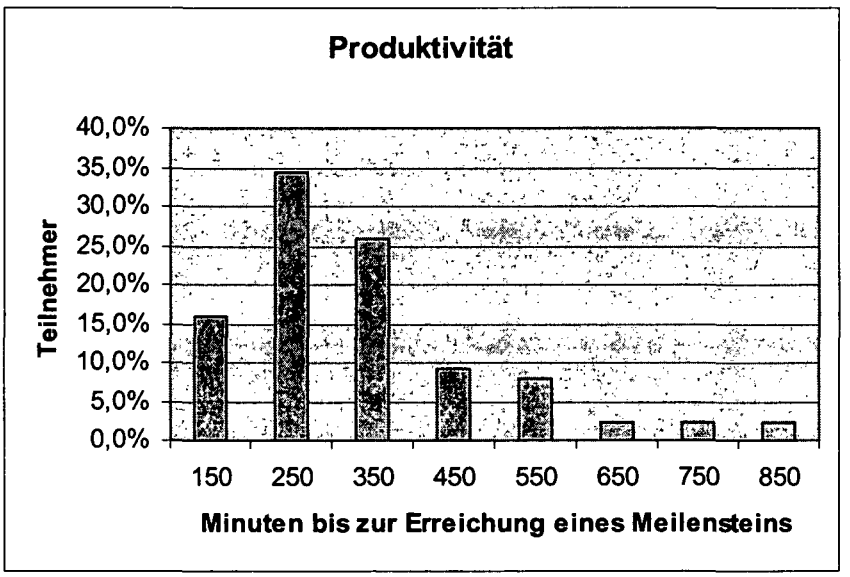


Abb. 7.5: Produktivität

Aus dieser und anderen Studien leitete Tom DeMarco folgende Grundregeln ab:

- Die besten Mitarbeiter sind um einen Faktor 10 besser als die schlechtesten.
- Die besten Mitarbeiter sind um einen Faktor 2,5 besser als der Durchschnitt.
- Überdurchschnittliche Mitarbeiter übertreffen unterdurchschnittliche im Verhältnis 2:1.

[15] DeMarco, Tom, Lister, Timothy, Wien wartet auf Dich! Der Faktor Mensch im DV-Management, München, 1999

7.3.1 Einflussfaktoren auf die Produktivität

Für die Analyse der Produktivität in der Software-Entwicklung empfiehlt sich eine Unterscheidung nach folgenden Determinanten:

7.3.1.1 Software-Produkt-Eigenschaften

Mit dem Ziel die Produktivität zu beurteilen sind folgende Kriterien relevant:

Produktkomplexität: Generell ist die Komplexität eine schwer in Zahlen zu fassende Größe. Dies ist bedingt durch den Umstand, dass ein sinnvolles Maß immer Rücksicht auf die Handhabung durch einen Menschen nehmen muss. Klassische Metriken sind jene von McCabe und Halstead. Die Metrik von McCabe dient der Beurteilung der strukturellen Komplexität von Programmen. Die Basis zur Berechnung bildet der Kontrollflussgraph [16]). Die Metriken von Halstead dienen zur Messung der textuellen Komplexität von Programmen. Hierbei spielt die Anzahl unterschiedlicher Operatoren und Operanden und deren Gesamtzahl eine tragende Rolle [17]). Die Anwendung beider Metriken ist einfach automatisierbar. Nachteilig ist die Tatsache, dass sie sich nur auf Teilaspekte der Komplexität beziehen.

Produktgröße: Diese Eigenschaft wirkt sich auf die Produktivität durch zwei gegenläufige Einflüsse unterschiedlich aus. Einerseits nimmt die Komplexität mit der Programmgröße zu, wodurch die Produktivität abnimmt – andererseits wird jedoch mit zunehmender Programmgröße auch ein Lerneffekt erzielt, der die Produktivität wachsen lässt.

Produktqualität: Auch hier sind die Auswirkungen uneinheitlich. Es existieren sowohl Argumente dafür, dass die Produktivität mit zunehmender Qualität sinkt (z.b. höhere Testaufwände), als auch solche, dass sie mit zunehmender Qualität steigt (z.b. Einsparungen durch frühe Fehlererkennung). Eine objektive Beurteilung dieses Sachverhalts führt zu einem verallgemeinerten Outputbegriff, der die Serviceleistungen miteinschließt. Geht man davon aus, dass diese Wartungs- und Pflege-Kosten oft 2/3 des Gesamtaufwands ausmachen [18]), kann eine höhere Qualität durchaus zu höheren Entwicklungs-Kosten bei geringeren Service-Einnahmen und in logischer Folge zu geringerer Produktivität führen.

---

[16] McCabe, T.J., A Testing Methodology Using the McCabe Complexity Measure, in: IEEE Computer Society Press, 1983

[17] Halstead, M.H., Elements of Software Science, New York, 1977

[18] Grady, R.B., Practical Software Metrics for Project Management and Process Improvement, 1992

7.3.1.2 Software-Entwicklungsprozess-Eigenschaften

Die Produktivität wird durch den Entwicklungsprozess in zweierlei Hinsicht beeinflusst. Einerseits werden durch Entwicklungs-Methoden die Arbeitsabläufe strukturiert, andererseits werden durch den Einsatz von CASE-Werkzeugen die Arbeitspakete automatisiert. „CASE" steht für: „Computer Aided Software Engineering" und umfasst alle computerunterstützten Hilfsmittel, die dazu beitragen, die Software-Produktivität und die Software-Qualität zu verbessern sowie das Software-Management zu erleichtern.

Der Produktivitätsgewinn liegt nach B.W. Boehm [19]) bei 1:1,92 durch einen Methoden-Einsatz bzw. bei 1:1,65 zwischen einem minimalen und maximalen CASE-Einsatz. Diese Angaben haben allerdings ohne Berücksichtigung der Lernkurve nur begrenzte Aussagekraft. Aus einer Studie der Gartner Group geht hervor, dass erst nach einer Einsatzzeit von 2-3 Jahren mit Produktivitäts-Steigerungen gerechnet werden kann. In der nachfolgenden Graphik (vgl. Abb. 7.6) ist dies für den Einsatz von Werkzeugketten, die die gesamte Entwicklung unterstützen (I-CASE), und für den Einsatz verschiedener Werkzeuge in den einzelnen Phasen (Mix) graphisch dargestellt [20]).

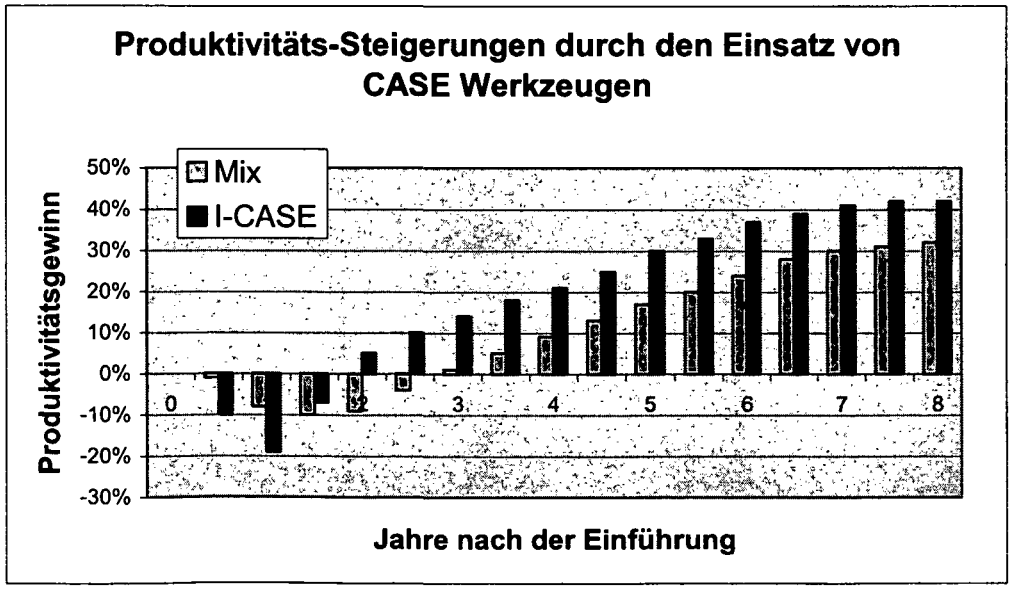


Abb. 7.6: Produktivitätssteigerung mit CASE-Werkzeugen

[19] Boehm, B.W., Improving Software Productivity, in: IEEE Computer, 1987
[20] Gartner Group, The Software Engineering Strategies Scenario, Stanford, 1990

Ohne dies mit Zahlen zu bestätigen kann ein beträchtlicher Produktivitätsgewinn durch Wiederverwendung von Software-Modulen erzielt werden. Dieser Vorteil erweist sich in der Praxis allerdings nur im erweiterten Projektumfeld als realisierbar.

Der Aufbau einer Datenbank, welche prinzipiell jedem Mitarbeiter die Möglichkeit gibt, auf beliebige bereits im Einsatz befindliche Programmsegmente zuzugreifen, hat sich aufgrund der Pflege- und Dokumentationsaufwände bis dato nicht durchgesetzt. Als Alternative zur Dokumentenstudie (z.b. mittels Datenbank) ist eine regelmäßige persönliche Kommunikation zwischen Mitgliedern unterschiedlicher Projekte unvergleichbar wertvoller. Darin ist u.a. auch der Vorteil größerer Software-Häuser im Vergleich zu kleineren (weniger als 100 Mitarbeiter) zu sehen.

7.3.1.3 Mitarbeitereinflüsse

Mit Referenz auf die Feldstudien von Tom DeMarco und Timothy Lister [21]) haben folgende Faktoren wenig oder gar keine Korrelation mit der Leistung eines Software-Entwicklers:

- Programmiersprachen
- Berufserfahrung
- Anzahl der Fehler
- Gehalt

Faktoren mit Auswirkung auf die Leistung lagen in der Feldstudie im Bereich der Managementeinflüsse. Ohne das Resultat zu kommentieren, sei darauf hingewiesen, dass das der Feldstudie zugrundeliegende Experiment (Programmierung eines einfachen Beispiels) doch beträchtlich von den tatsächlichen Arbeitsinhalten abweicht.

Meinen eigenen Erfahrungen zufolge gibt es sehr wohl Mitarbeiter-Charakteristika, die mit der Leistung korrelieren. Der wichtigste Faktor ist die persönliche Einstellung zum Entwerfen und Codieren von Software: „Welche persönliche Bereicherung erfährt der Software-Entwickler bei seiner Arbeit ?" 90% jener Kollegen, die außergewöhnliche Produktivität zeigen, beschäftigen sich auch in ihrer Privatzeit im weitesten Sinne mit der Entwicklung von Software. Der zweitwichtigste Faktor (zufolge eigener Erfahrung) ist – im Gegensatz zur Studie von DeMarco/Lister – die Berufserfahrung. Eine Bestätigung hierfür ist die oben im Zusammenhang mit dem Einsatz von CASE-Werkzeugen gezeigte Lernkurve, die den Produktivitätswachstum über mehrere Jahre! zeigt.

---

[21] DeMarco, Tom, Lister, Timothy, Wien wartet auf Dich! Der Faktor Mensch im DV-Management, München, 1999

7.3.1.4 Managementeinflüsse

Einen unbestritten wichtigen Einfluss hat das Management bei größeren Software-Projekten. Abgesehen von den Einflüssen die im Kapitel 10 genannt werden, spielt auch die Arbeitsumgebung eine große Rolle. Auffallend in den von DeMarco/Lister durchgeführten Studien war die geringe Streuung zwischen Mitarbeitern (21%) der gleichen Firma. Dies legt nahe, dass der Unterschied nicht nur beim Vorgesetzten, sondern vielmehr bei der jeweiligen Firmenkultur zu suchen ist. Wichtige, vom Management steuerbare Einflussfaktoren sind:

- Fläche des Arbeitsplatzes pro Person
- Ruhe am Arbeitsplatz
- Wahrung der Privatsphäre
- Störungen durch Telefonanrufe
- Unterbrechungen seitens der Kollegen

Im Zusammenhang mit den Störungen hat DeMarco/Lister den „Umweltfaktor" eingeführt. Dieser ist der Quotient aus „Ungestörte Stunden" und „Stunden körperlicher Anwesenheit". Wenn dieser nicht mindestens 40% erreicht, ist es für den Mitarbeiter sehr schwierig die notwendige Konzentration zu finden und aufrecht zu erhalten. DeMarco/Lister: „Während der Zeit, in der Mitarbeiter an einer einzigen Sache arbeiten, sollten sie idealerweise in einem Zustand sein, den die Psychologen ‚in Fahrt' nennen. ‚In Fahrt' ist ein Zustand tiefer, fast meditativer Versunkenheit. In diesem Zustand fühlt man eine leichte Art von Euphorie, und man hat kein Gefühl für Zeit. ... Man merkt die Arbeit nicht, sie geht einfach locker von der Hand."

Das oben gesagte trifft auf einen Teil der insgesamt von einem Software-Entwickler zu bewältigenden Tätigkeiten zu. Während einer etwa ebenso großen Zeitspanne findet eine Kommunikation mit anderen Mitarbeitern (z.b. Meetings, Reviews, usw.) statt, die, wenn sie behindert wird, ebenfalls negative Einflüsse (z.b. geringerer Lerneffekt, Doppelgleisigkeit, usw.) auf die Produktivität hat.

8. Metrik-Programme

"Not everything that counts can be counted and
not everything that is counted counts."

Albert Einstein

8.1 Gründe für Metrik-Programme

„Erfahrungen aus den verschiedensten Zweigen der Software-Entwicklung zeigen drastisch, dass ungefähr zwei von drei Metrikprogrammen in der Praxis nach ein oder zwei Jahren scheitern." – Ein Zitat von Christof Ebert aus „Software-Metriken in der Praxis" [1]). Als Gründe sind unzureichende Zielformulierungen und fehlende Konkretisierungen der Metrik-Programme bei der Umsetzung zu nennen.

Unter einer Metrik versteht man ein Maß bzw. allgemein einen Indikator für Objekteigenschaften, welche die Grundlage zur quantitativen Software-Beurteilung ist. Wie in der Physik, gibt es auch in der Informatik unterschiedlichste Maße bzw. Metriken, die sowohl das Produkt, als auch den Entwicklungs-Prozess erfassen.

Das Ziel eines Metrik-Programms kann überaus vielfältig sein. So wie jedes Modell in den unterschiedlichsten Disziplinen Anwendung findet, sind auch der Nutzung eines Metrik-Programms prinzipiell keine Grenzen gesetzt. Demzufolge kann z.b. die Datenerfassung auch auf personenbezogene Daten ausgeweitet werden, was u.a. ein Grund dafür ist, dass bei Einführung eines solchen Programms mit Widerstand seitens der Mitarbeiter zu rechnen ist. Hierzu ein Auszug aus: Wien wartet auf Dich! von Tom DeMarco und Timothy Lister [2]):

„... Um die wirklichen Vorteile aus Produktivitätsmetriken ernten zu können, muss das Management weitsichtig und selbstsicher genug sein und sich aus dem Prozess heraushalten. Das heißt, es dürfen keine personenbezogenen Daten an das Management weitergegeben werden, und jeder in der Firma muss das wissen und davon überzeugt sein. Individuelle Daten dürfen nur zum Vorteil der davon betroffenen Personen verwendet werden. ... Wenn die Vertraulichkeit jemals in Frage gestellt wird, wenn jemals irgendwelche Daten gegen eine Einzelperson verwendet werden, bricht das gesamte Schema schlagartig zusammen."

---

[1] Ebert, Christof, Dumke, Reiner, (hrsg.), Software-Metriken in der Praxis, Berlin, 1996
[2] DeMarco, Tom, Lister, Timothy, Wien wartet auf Dich! Der Faktor Mensch im DV-Management, München, 1999

Neben den in diesem Auszug angesprochenen Metriken zur Produktivitätsmessung werden Metriken zur Bestimmung und Verbesserung der Qualität – zum Qualitäts-Management – eingesetzt. Aufgrund der durch TQM und ähnlicher Qualitätsphilosophien weit ausgedehnten Anwendungsbereiche ist die Trennung zwischen personenunabhängigen und personenbezogenen Metriken kaum eindeutig vorzunehmen, wodurch der heikle Aspekt der personenbezogenen Produktivitätsmessung mit allen ihren möglichen Folgen stets mit dem Thema: Metriken verknüpft ist.

Sind die Metriken einmal definiert, und stehen sie dem Anwender in geeigneter Form zur Verfügung, beginnt die aufwendige Arbeit der richtigen Interpretation der Daten. Da die Prozesse des Software-Engineering im Vergleich zu anderen Einsatzgebieten noch relativ jung sind, existieren auch erst wenige Referenz-Prozesse. Das Thema: Software-Engineering wurde während einer NATO-Tagung vor etwa 30 Jahren zum ersten Mal erwähnt. Seither gibt es unaufhörlich Fortschritte bei den Werkzeugen zur Software-Entwicklung – und damit verbunden – auch bei den Methoden, die diese Tools in geeigneter Weise einsetzen. Stellt man den Prozess des Software-Engineering in Form eines einfachen Regelkreises dar, erhält man folgende Semantik (ein umfangreicheres Modell ist das im Kapitel 10 dargestellte 7-D-Modell nach Exl):

1. Strecke: Software-Entwicklungs-Prozess
2. Regler: Metrikerfassung, Metrikbewertung
3. Soll/Ist-Vergleich: Steuerungs-Instrumente für den Projektleiter

Die Reihenfolge entspricht dabei der Schwierigkeit zur praktischen Anwendung und dem sinkenden Angebot an Literatur. Ein anderes Faktum ist die unterschiedliche Zurechenbarkeit zu wissenschaftlichen Disziplinen. Während die Software-Entwicklung (Analyse, Entwurf, Implementierung, Test) dem inzwischen etablierten Bereich der Informatik angehört, erfordert der 3. Bereich ein hohes Maß an kognitiver und emotionaler Kompetenz. Infolgedessen sind für eine effiziente Steuerung mittels Metriken neben den technologischen Gesichtspunkten auch Managementqualitäten von Bedeutung.

Sämtliche theoretischen Modelle zur Software-Entwicklung werden von einer betrieblichen Organisation getragen, die den vielfältigen Anforderungen genügen muss. Diese Anforderungen reichen vom langfristig zu planenden Personaleinsatz bis hin zur effizienten Steuerung anhand von Leistungs- und Qualitätsmetriken. Die den vielfältigen Informationsflüssen am besten geeigneten Personalstrukturen sind Matrixorganisationen die überdies imstande sein müssen sich den Prozessänderungen rasch anzupassen. Diese Organisationsform trägt einerseits den Aufgaben der Linienorganisation und andererseits jenen der Projektorganisation Rechnung [3]).

[3] Patzak, Gerold, Rattay, Günter, Projekt Management, Wien, 1998

Determinanten der Organisationsentwicklung sind hierbei:

- Projektumfang
- Projektkomplexität
- örtliche Verteilung der Projekt-Mitarbeiter (internationale Projekte)
- Anforderungen aus dem Claim-Management
- Personal-Anforderungen (Qualifikation)
- notwendige Flexibilität um Korrekturen durchführen zu können
- standardisierte Prozesse (CMM, ISO 900x, usw.)
- firmeninterne Kultur (Spielregeln) 4)
- Technologie

Ein frühzeitig zu erfolgender Schritt bei der Einführung eines Metrik-Programms ist die Kompetenz- und Verantwortlichkeits-Verteilung. Je nach zugrundeliegendem Arbeitsteilungs-Prinzip können Verantwortliche für Phasenabschnitte oder für Teilprojekte eingerichtet werden. Während Teilprojektleiter einen guten Überblick über einen Teilfunktionsumfang der zu erstellenden Software haben, sollten Phasenabschnitts-Verantwortliche Spezialwissen hinsichtlich der Prozessabschnitte einbringen.

Betreffend dem Ursprung der Metriken kann eine erste Einteilung in personenbezogene und sachbezogene Metriken erfolgen. Eine personenbezogene Metrik-Strukturierung betont die Aufbauorganisation, während eine sachbezogene Metrik-Strukturierung auf Produkt- oder Prozessebene ansetzt (vgl. Tab. 8.1):

Personenbezogen	**Sachbezogen**
• Mitarbeiterebene	• Leistungsmerkmalebene
• Gruppenleiterebene	• Funktionsebene
• Abteilungsebene	• Phasenabschnittsebene
• Qualitätssicherungsebene	• Versionsebene
• Projektmanagementebene	• Teilprojektebene
	• Projektebene

Tab. 8.1: Metrik-Strukturierung

Die Frage nach der Quelle für eine Metrik ist eng verknüpft mit dem Fortschritt des Software-Projekts, da hiervon der Datentyp der Metrik determiniert ist. Abhängig vom Projektfortschritt (Phasenabschnitt) kann ein Teilresultat (Arbeitspaket) in Form eines Pflichtenhefts, eines Datenmodells, einer Codierung, usw. vorliegen, wodurch die Metrik z.b. aufgrund eines Dokuments oder basierend auf Quellcode ermittelt wird.

[4] Scott-Morgan, Peter, Little, Arthur D., Unwritten Rules of the Game, McGraw-Hill, 1994

8.2 Metrikprogramme des Qualitäts-Managements

Neben der Beurteilung der Produktivität bzw. des Sachfortschritts ist die Ausprägung der Qualität ein zweiter wichtiger Anwendungsbereich von Metriken. Dies ist neben vielen anderen Gesichtspunkten insofern von Bedeutung, da die Korrektur von Fehlern in späteren Phasen des Software-Entwicklungs-Prozesses besonders aufwendig und teuer ist. Ferner schränkt eine mangelnde Qualität des Software-Entwicklungs-Prozesses auch die Verlässlichkeit von Metriken zur Bewertung des Sachfortschritts ein.

Einer der wesentlichsten Punkte für die bevorzugte Anwendung von Metrik-Programmen auf den Qualitätsaspekt (im Gegensatz zum Produktivitätsaspekt) wird durch folgende Aussage von Tom DeMarco und Timothy Lister aussagekräftig beschrieben [5]): „Der Ansatz zum Messen der Produktivität bei Kopfarbeitern ist wissenschaftlich noch nicht so untermauert, wie es notwendig wäre. Einige Personen stufen die bisherigen Ergebnisse nur geringfügig besser ein als die Forschung im Bereich unbekannter Flugobjekte."

Da der Qualitätsbegriff je nach Ansatz variiert, ist eine strategische Zielformulierung die Basis von qualitätssichernden Maßnahmen. Als Ansätze sind folgende zu nennen [6]):

- TQM-Ansatz:
  Der TQM-Ansatz ist ein das gesamte Unternehmen berücksichtigender Ansatz. ISO 8402 definiert Total Quality Management (TQM) mit den Worten: „Auf der Mitwirkung aller ihrer Mitglieder basierende Führungsmethode einer Organisation, die Qualität in den Mittelpunkt stellt und durch Zufriedenheit der Kunden auf langfristigen Geschäftserfolg sowie auf Nutzen für die Mitglieder der Organisation und für die Gesellschaft zielt."

- Transzendenter Ansatz:
  Dieser Ansatz legt die Erfahrung und das Image des Software-Produkts zugrunde. Er ist einer objektiven Messung nicht zugänglich und folglich nur von geringer Bedeutung.

- Benutzerbezogener Ansatz:
  In diesem Ansatz wird die Qualität anhand der Kriterien des Software-Benutzers beurteilt.

- Produktbezogener Ansatz:
  Zur Qualitätsbeurteilung werden ausschließlich Kriterien des Endprodukts herangezogen.

---

[5] DeMarco, Tom, Lister, Timothy, Wien wartet auf Dich! Der Faktor Mensch im DV-Management, München, 1999
[6] Garvin, D.A., Managing Quality: The Strategic and Competitive Edge, New York, 1988

- Prozessbezogener Ansatz:
  Dieser Ansatz hat die Optimierung des Software-Entwicklungs-Prozesses zum Ziel. Er ist der in der Praxis am häufigsten implementierte Ansatz. Der Grund hierfür ist die Annahme, dass die Produkt-Qualität über die Prozess-Qualität zu erreichen ist.

- Kosten-/Nutzenbezogener Ansatz:
  Dieser Ansatz sollte allen Software-Produkten zugrunde gelegt werden, da er die Basis für betriebswirtschaftliches Handeln darstellt.

In der Praxis der Software-Entwicklung spielen alle Ansätze eine mehr oder minder wichtige Rolle. Während z.b. der Benutzerbezogene Ansatz vor allem in der Definitionsphase von Bedeutung ist, kommen produktbezogene Kriterien in der Mehrheit erst beim Endprodukt zum Tragen. Aufgrund des sehr umfassenden Anspruchs hat der **TQM-Ansatz** als philosophischer Ausgangspunkt für konkrete Standards oder Modelle großen Einfluss. Eine konkrete Ausgestaltung ist dem einzelnen Unternehmen vorbehalten. Charakteristisch für ihn sind:

- Berücksichtigung der Interessen aller Stakeholder (Mitarbeiter, Kunden, u.a.)
- Interpretation des Unternehmens als sozio-technisches System
- Primat der Qualität und der kontinuierlichen Qualitätsverbesserung
- Orientierung an den Unternehmens-Prozessen

Genormte Begriffe zur Software-Qualitäts-Thematik sind u.a. in ISO 9126 zu finden. Hierin wird unter Software-Qualität die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts verstanden, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen. Unter Zuhilfenahme von Qualitätsmodellen werden die Ansätze und Definitionen operationalisiert.

In den folgenden Abschnitten werden einige der derzeit am häufigsten angewandten Ansätze bzw. Modelle erläutert. Als Repräsentant des **Produktbezogenen Ansatzes** wird der

- Factor Criteria Metrics – Ansatz

vorgestellt. In neuerer Zeit basieren die in der Praxis zumeist angewandten Modelle auf einer Kombination von prozess- und produktbasiertem Ansatz, wobei der erstere einen vorrangigen Gestaltungseinfluss auf die Software-Projekt-Management-Organisation einnimmt. Die bedeutendsten Referenz-Modelle für den **Prozessbezogenen Ansatz** sind:

- Quality Improvement Paradigm (QIP) – Ansatz
- ISO 9000 – Ansatz
- Capability Maturity Model (CMM) – Ansatz
- SPICE – bzw. ISO/IEC 15504 – Ansatz

Bei den 3 letztgenannten Ansätzen handelt es sich eigentlich um Familien von Standards, Richtlinien und begleitenden Dokumenten die die Terminologie, die Durchführung von Assessments und die Umsetzung von Maßnahmen zum Inhalt haben [7]). Eine Sammlung von Links zu diesen Themen ist im Internet [8]) zu finden.

8.2.1 Factor Criteria Metrics - Ansatz (FCM)

Dieses Modell basiert auf einer zumindest 3-stufigen Hierarchie. In einem ersten Schritt werden die Qualitätsmerkmale (Factor) festgelegt. Sie basieren auf den Kunden-Anforderungen. Danach werden sie weiter untergliedert und mit Charakteristika (Criteria) der Software in Verbindung gebracht. Ein Kriterium kann hierbei für mehrere Qualitätsmerkmale relevant sein. Im 3. Schritt werden den Kriterien schließlich Metriken (Metrics) zugeordnet. - Mit dieser Relation wird sowohl ein Maß als auch eine Meßmethode definiert. Im Rahmen der DIN ISO 9126 werden folgende Qualitätsmerkmale für Software-Produkte definiert:

- Funktionalität (Richtigkeit, Angemessenheit, Interoperabilität, usw.)
- Zuverlässigkeit (Reife, Fehlertoleranz, usw.)
- Benutzbarkeit (Bedienbarkeit, Erlernbarkeit, usw.)
- Effizienz (Zeitverhalten, usw.)
- Änderbarkeit (Modifizierbarkeit, Stabilität, usw.)
- Übertragbarkeit (Installierbarkeit, Austauschbarkeit, usw.)

Im strengen Sinne handelt es sich hierbei um ein Meta-Modell, da erst durch die Konkretisierung der Hierarchie das Modell entsteht. Ein anderes Beispiel für die Realisierung eines FCM-Meta-Modells ist das FURPS-Modell von Hewlett Packard [9]). Seine Qualitäts-Merkmale sind: Functionality, Usability, Reliability, Performance, und Supportability.

8.2.2 Quality Improvement Paradigm – Ansatz (QIP)

QIP ist ein Ansatz zur kontinuierlichen Verbesserung und zur systematischen Durchführung von Software-Projekten. Beschrieben wird das QIP anhand von mehreren Schritten, die zyklisch angewandt werden [10]):

---

[7] http://www.questhouse.com/excerpt1.htm , Engineering a Better Software Organization, 2003

[8] http://www.tantara.ab.ca/info.htm , Informational Hotlist (for software process improvement, software quality assurance,...), 2004

[9] Grady, R.B., Caswell D.L., Software Metrics: Establishing a Company-Wide Program, 1987

[10] Basili, V. R., Applying the Goal / Question / Metric Paradigm in the Experience Factory, in: Software Quality Assurance & Metrics, Chapman & Hall, London, 1994

1. Charakterisiere Projekt und identifiziere existierendes Know-how
   (d.h. erfasse den Ist-Zustand und künftige Trends)
2. Definiere Ziele des Projekts in messbarer Form
   (d.h. stelle Verbesserungsziele und Hypothesen auf)
3. Wähle geeignete Prozesse und stelle Projektplan zusammen
   (d.h. identifiziere geeignete Projekte oder Experimente zum Testen der Hypothesen)
4. Führe das Projekt gemäß Plan durch, erfasse und verwende Daten zur Projektkontrolle
   (d.h. führe Pilotprojekte oder Experimente durch)
5. Analysiere den Projektverlauf und schlage Verbesserungen vor
   (d.h. werte Ergebnisse aus)
6. Sichere Erfahrungen
   (d.h. bringe Ergebnisse als Know-how in künftige Projekte ein)

Anhand des QIP sollen nun die Wesensbestandteile eines Programms zur Qualitätsverbesserung von Software näher erläutert werden. Damit einhergehend wird eine gezielte Steuerung des Entwicklungs-Prozesses ermöglicht.

Die Grundlagen zum QIP-Ansatz lassen sich am besten mit Hilfe folgender drei Infrastruktur-Technologien charakterisieren:

1. Zielorientiertes Messen

Es berücksichtigt neben der Zielorientierung auch die Zweckgebundenheit und die Kontextabhängigkeit. Hierzu brauchbare Ansätze werden von folgenden Modellen geliefert (eine detaillierte Beschreibung erfolgt später):

- Goal Question Metrics (GQM)
- Software Quality Metrics (SQM)
- Quality Function Deployment (QFD)

2. Explizites Modellieren

Mittels QIP modellierte Softwareprodukte und –prozesse erfordern eine Prozess-Modellierungs-Sprache mit folgenden Eigenschaften:

- alle relevanten Objekte (Produkte, Prozesse, Ressourcen, Eigenschaften, usw.) sind modellierbar
- alle wichtigen Beziehungen zwischen diesen Objekten (Produktfluss, Kontrollfluss, Ressourcenzuordnung, usw.) sind modellierbar

- Konzepte zur Verfeinerung bzw. Aggregation, Sichtenextraktion und -kombination sind modellierbar
- Projektpläne können analysiert werden und sind für Messungen instrumentierbar

Als ein in der Industrie angewandtes Beispiel gilt MVP-L (Multi View Process Modeling Language)

3. Umfassende Wiederverwendung

Dadurch sollen die Gesamtkosten der Software-Entwicklung minimiert und die Gesamtqualität maximiert werden. Es wird hierbei unterstellt, dass ein wesentlicher Anteil der Kosten durch die Testphase verursacht wird, und die Erzielung beinahe fehlerfreier Software eine hohe Priorität genießt.

8.2.2.1 Goal Question Metric Ansatz (GQM)

Bei Anwendung des GQM-Ansatzes [11]) können beliebige Ziele zu beliebigen Zwecken und aus beliebigen Blickwinkeln definiert werden. Die wichtigsten Prinzipien des GQM-Modells sind:

- Festlegung expliziter Analyseziele
  (was wird wozu unter welchem Blickwinkel gemessen)
- Top-down-Definition und Bottom-up-Interpretation der definierten Maße
- Miteinbeziehung der betroffenen Personen in alle Phasen

Umgesetzt werden diese Prinzipien durch die GQM-Methode, die die Planung und Durchführung von Messungen umfasst. Hierzu beschreibt ein Plan die Umsetzung der Ziele in Maße und die Art der Interpretation. Die Praxisnähe dieser Methode zeigt sich auch an den Formularen (GQM-Masken), die von den Mitarbeitern im Zuge der Anwendung auszufüllen sind. Geordnet wird das Datensammeln durch einen GQM-Messplan, der festlegt, welche Informationen wann, durch wen und wie erhoben werden. Nach Projektende werden die gewonnenen Daten analysiert und interpretiert. Um die Daten auch für zukünftige Projekte verfügbar zu halten, ist eine geeignete Aufbereitung in einer Erfahrungsdatenbank zweckmäßig.

Die Beschreibung eines GQM-Modells erfolgt durch die Definition eines **Ziels** sowie einer Menge dazugehöriger **Fragen und Maßen**. Während das Ziel auf der konzeptionellen Ebene angeordnet ist, sind die Fragen auf der operativen und die Maße auf der quantitativen Ebene definiert.

[11] IEEE, A methodology for collecting valid software engineering data, IEEE Vol. SE-10, 6, 1984, p. 728-738

Ein **Ziel** ist definiert durch die Angabe eines Objekts, eines Zwecks, eines Qualitätsmerkmals, eines Blickwinkels und eines Kontexts; z.b: *Analysiere* den Komponententest [Objekt] *zum Zwecke* des Verstehens [Zweck] *bezüglich* der Effektivität [Qualitätsaspekt] *aus dem Blickwinkel* des Testers [Blickwinkel] *bei* Projekt A [Kontext]. Die Objekte der Messung können Prozesse, Produkte und Ressourcen sein. Als Zweck sind z.b. Verbesserung, Kontrolle und Rationalisierung zu verstehen. Die **Fragen** werden zu den Themen Prozessdefinition und Produktdefinition, Qualität und Feedback gestellt.

8.2.2.2 Software Quality Metrics (SQM) und Quality Function Deployment (QFD)

- **QFD**: Dieser Ansatz soll während der Planung jene Weichen stellen, die das Produkt aus Kundensicht optimal zu gestalten helfen. Es werden nur Produktmerkmale betrachtet. Zur Ableitung von Charakteristika wird keine Hilfestellung gegeben.

- **SQM**: Vergleichbar mit dem QFD-Ansatz optimiert man hierarchisch definierte Kriterien.

Aus der folgenden Übersicht (Tab. 8.2) lassen sich die wesentlichsten Merkmal-Unterschiede schnell erkennen [12]). Die Rubrik „Ableitung von Maßen" unterstreicht den anspruchsvollen Arbeits-Schritt der Bildung von Maßen. Er wird durch die Modelle beschrieben und unterstützt:

Kriterien	QFD	SQM	GQM
Anforderungen des QIP:			
Zielorientierung	ja	ja	ja
Zweckgebundenheit	eingeschränkt	eingeschränkt	ja
Kontextabhängigkeit	nein	nein	ja
Prozessmaße	nein	nein	ja
Produktmaße	ja	ja	ja
Ableitung von Maßen: Prinzip	Ableitung des Produktcharakters aus dem Benutzercharakter des Produkts	Verfeinere Faktoren in Kriterien und Maße	Verfeinere Ziele in Fragen und Maße
Mechanismus	auswählen, anpassen	auswählen	auswählen, anpassen

Tab. 8.2: QIP-Ansätze

[12] Ebert, Christof, Dumke, Reiner, (hrsg.), Software-Metriken in der Praxis, Berlin, 1996

8.2.3 ISO 9000 – Ansatz

Um Qualitäts-Management-Systeme international in den unterschiedlichsten Branchen zu vereinheitlichen, erschien 1985 von der ISO (International Organisation for Standardization) die Normenserie ISO 9000 bis ISO 9004 (im weiteren kurz als ISO 900x bezeichnet). Sie erfüllt zwei grundlegende Anwendungssituationen:

- Nachweis von qualitätssichernden Tätigkeiten gegenüber Dritten (ISO 9001)
- Unterstützung des Aufbaus eines Qualitäts-Management-Systems (ISO 9004)

Die für ein Software-Haus maßgebliche Norm ist ISO 9000-3, die die Interpretation von ISO 9001 für die Entwicklung, Lieferung, Installierung und Wartung von Software beschreibt [13]). Ist das Qualitäts-Management-System entsprechend ISO 9000-3 realisiert, erfüllt es automatisch die für die Zertifizierung nach ISO 9001 notwendigen Kriterien. Mit der Zertifizierung eines Unternehmens nach ISO 900x wird ausschließlich bestätigt, dass das Unternehmen dem Normenwerk entsprechende Strukturen aufweist. Eine damit einhergehende hohe Produktqualität wird angenommen, jedoch nicht notwendigerweise vorausgesetzt. Hinsichtlich inhaltlichem Aufbau orientiert sich ISO 9000-3 grundsätzlich am Auftraggeber-Lieferanten-Verhältnis, wobei Prüfbarkeit, Nachvollziehbarkeit und Personenunabhängigkeit im Zentrum der Forderungen stehen. Dies gilt für die Aufbau- und Ablauforganisation, die Verantwortungsbereiche, die Zuständigkeiten und die Dokumente.

Als Vorteile einer ISO 900x Zertifizierung sind vor allem die Reduzierung des Produkthaftungsrisikos und die Erleichterung bei Akquisitionen zu nennen. Als Nachteile sind der unsystematische Aufbau – keine Trennung zwischen Fach-, Management- und QS-Aufgaben – und der hohe bürokratische Aufwand – durch die Vielzahl an geforderten Dokumenten – zu nennen. Inhaltlich bezieht sich die Norm auf kein Vorgehens-Modell. Als Minimalforderung wird ein Phasenmodell vorausgesetzt, das die Vorgaben, Tätigkeiten und Ergebnisse jeder Phase festlegt. Im Detail werden folgende Dokumente eingefordert [14]):

- Vertrag: Auftraggeber-Lieferant
  - Annahmekriterien
  - Behandlung von Änderungen der Auftraggeberforderungen während der Entwicklung
  - Behandlung von Problemen, die nach der Annahme entdeckt werden, einschließlich qualitätsbezogener Ansprüche und Auftraggeberbeschwerden
  - Tätigkeiten, die vom Auftraggeber erbracht werden, insbesondere die Rolle des Auftraggebers bei der Festlegung der Forderungen, bei der Installation und bei der Abnahme
  - Vom Auftraggeber beizustellende Einrichtungen, Werkzeuge und Software-Elemente
  - Anzuwendende Normen und Verfahren

---

[13] Europäische Norm EN ISO 9000-3: 1997

[14] Balzert, Helmut, Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Heidelberg, 1998

- Forderungen an die Vervielfältigung

- Spezifikation
  - Vollständiger und eindeutiger Satz von funktionalen Forderungen
  - Leistung
  - Ausfallsicherheit
  - Zuverlässigkeit
  - Datensicherheit
  - Persönlichkeitsschutz
  - Schnittstellen zu anderen Software- und Hardwareprodukten

- Entwicklungsplan
  - Festlegung des Projekts einschließlich seiner Ziele und Verweise auf mit diesem Projekt in Beziehung stehende Projekte des Auftraggebers oder des Lieferanten
  - Planung der Projektmittel einschließlich Teamstruktur, Verantwortlichkeiten, Heranziehung von Unterlieferanten und zu verwendender materieller Hilfsmittel
  - Entwicklungsphasen
    - Welche Phasen?
    - Welche Vorgaben sind für jede Phase gefordert?
    - Welche Ergebnisse sind von jeder Phase gefordert?
    - Welche Verifizierungsverfahren sind in jeder Phase durchzuführen?
    - Festlegung, dass potentielle Probleme zu analysieren sind
  - Management
    - Termine für Entwicklung, Implementierung und dazugehörige Lieferungen
    - Fortschrittsüberwachung
    - Organisatorische Verantwortungen, Mittel und Arbeitszuteilungen
    - Organisatorische und technische Schnittstellen zwischen verschiedenen Gruppen
  - Entwicklungsmethoden und Werkzeuge
    - Regeln, Praktiken und Übereinkommen für die Entwicklung
    - Werkzeuge und Techniken für die Entwicklung
    - Konfigurationsmanagement
  - Projektplan
    - Festlegung aller durchzuführenden Aufgaben
    - Festlegung der für jede der Aufgaben nötigen Mittel und die benötigte Zeit
    - Festlegung der Wechselbeziehungen zwischen den Aufgaben
  - Identifikation verwendeter Pläne, wie z.b. Qualitätssicherungsplan, Konfigurationsmanagementplan, Integrationsplan, Testplan

- Qualitätsplanung

  Hierzu ein Ausschnitt aus der Norm EN ISO 9000-3: 1997, deutsche Fassung:

  „Die Qualitätsplanung sollte sich, soweit zutreffend, auf folgende Punkte richten:

  a) Qualitätsanforderungen, die möglichst in messbaren Größen anzugeben sind;
  b) das für die Softwareentwicklung anzuwendende Phasenmodell;
  c) festgelegte Kriterien für Beginn und Abschluss jeder Projektphase;
  d) Festlegung der Arten von Prüfungen, Tests und weiterer auszuführender Verifizierungs- und Validierungstätigkeiten;
  e) Festlegung von anzuwendenden Verfahren des Konfigurationsmanagements;
  f) detaillierte Planung (eingeschlossen Ablaufpläne, Verfahren, Ressourcen und deren Genehmigung) sowie spezifische Verantwortlichkeiten und Befugnisse für:

- Konfigurationsmanagement,
- Verifizierung und Validierung von entwickelten Produkten,
- Verifizierung und Validierung von beschafften Produkten,
- Verifizierung von Produkten, die vom Kunden beigestellt werden,
- Lenkung fehlerhafter Produkte und Korrekturmaßnahmen,
- Sicherstellung der Durchführung der im Qualitätsmanagementplan beschriebenen Maßnahmen"

- Testplan
  - Pläne für Software-Elemente, Integration, Systemtest und Annahmeprüfung
  - Testfälle, Testdaten und erwartete Ergebnisse
  - Arten der durchzuführenden Tests, z.b. Funktionstest, Test unter Grenzbedingungen, Leistungstests, Brauchbarkeitstests
  - Testumgebung, Werkzeuge und Test-Software
  - Kriterien für die Vollständigkeit des Tests
  - Anwenderdokumentation
  - Erforderliches Personal und damit verbundene Schulungserfordernisse

- Wartungsplan
  - Umfang der Wartung
  - Identifikation des Ausgangszustands des Produktes
  - Unterstützende Organisationen
  - Wartungstätigkeiten
  - Wartungsaufzeichnungen und –berichte

- Konfigurationsmanagementplan
  - Organisationen, die am Konfigurationsmanagement beteiligt sind, sowie die ihnen zugewiesenen Verantwortlichkeiten
  - Auszuführende Konfigurationsmanagement-Tätigkeiten
  - Zu verwendende Konfigurationsmanagement-Werkzeuge, -Technologien und -Methoden
  - Stadium, in dem Elemente der Konfigurationslenkung unterworfen werden sollen

Des weiteren werden in der Norm noch phasenunabhängige Tätigkeiten gefordert, wie z.b. hinsichtlich des Unterauftragsmanagements (Beurteilung von Unterlieferanten, Validierung von beschafften Produkten). Für die Erteilung eines Zertifikats ist eine Zertifizierungsstelle verantwortlich – wie z.b. in DIN EN 45012 angeführt. Die Zertifizierung läuft in 4 Phasen ab, wobei zunächst Informationsgespräche, dann die Prüfung von QS-Unterlagen, und abschließend ein Audit – basierend auf einem Interview-Fragebogen - durchgeführt werden. Die Gültigkeit ist auf 3 Jahre beschränkt, sofern jährlich Überwachungs-Audits erfolgreich durchgeführt werden. Anschließend ist der Zertifizierungsprozess erneut zu durchlaufen. Hierdurch ist sowohl die Firmenleitung, als auch jeder einzelne Mitarbeiter, zur kontinuierlichen Einhaltung der ISO 900x-Bestimmungen angehalten.

8.2.4 Capability Maturity Model – Ansatz (CMM)

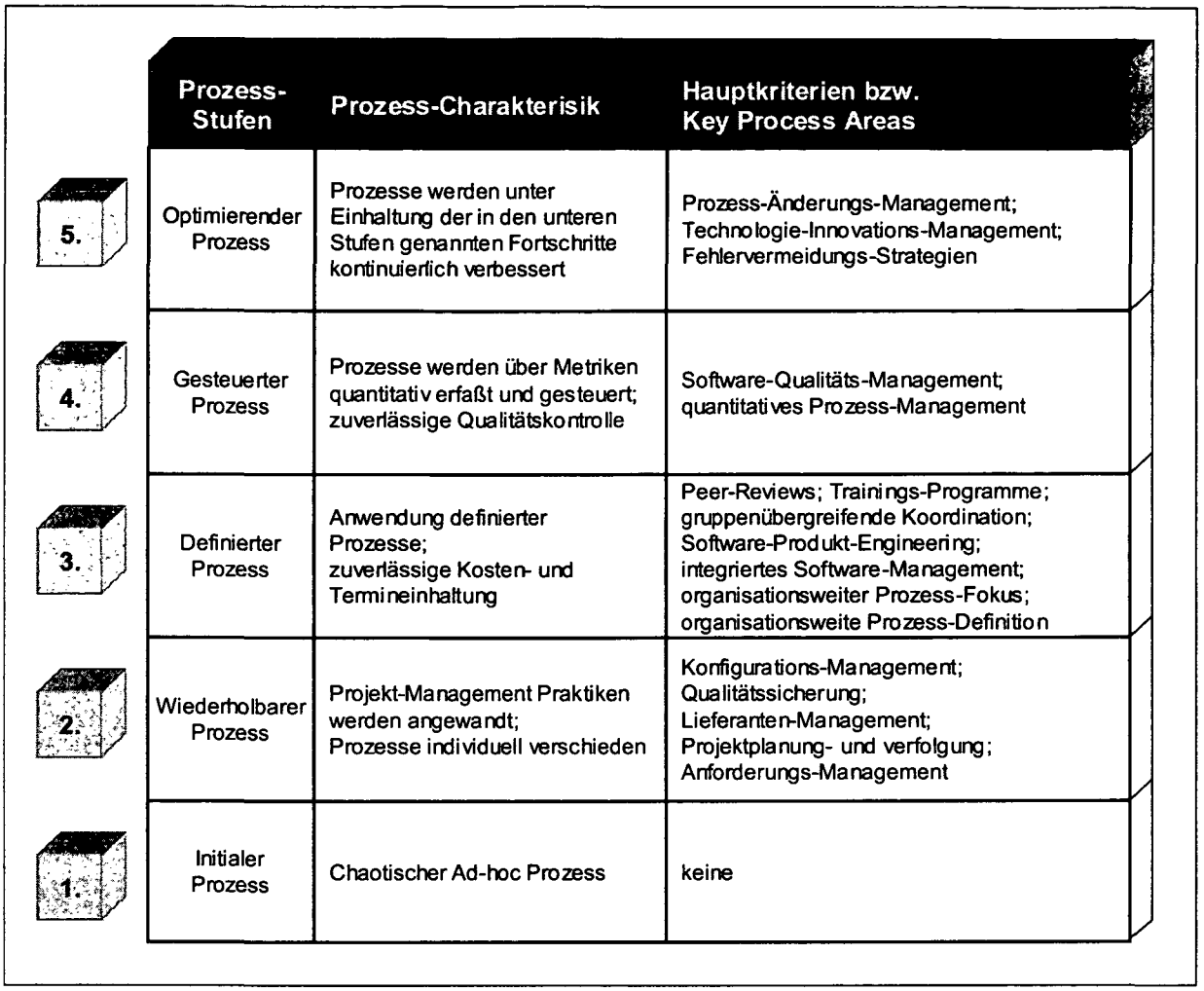
Das Capability Maturity Model (CMM) wurde am Software Engineering Institute (SEI) der Carnegie Mellon University (Pittsburgh, USA) im Jahr 1991 in der Version 1.0 erstmals veröffentlicht. Hierbei ist die Gründung des SEI sogar in unmittelbarem Zusammenhang mit dem Entstehen des CMM zu sehen. – Die Motivation für die Errichtung des SEI war der Wunsch des US-Department of Defense (kurz DoD) ein Verfahren zu entwickeln, das die Fähigkeit besitzt, einen Software-Lieferanten des DoD zu beurteilen. Mittlerweile ist das hierfür entwickelte CMM das wahrscheinlich weltweit bekannteste Referenzmodell zur Bestimmung des Reifegrades bzw. Qualitätsstandards eines Softwarehauses.

Deklariertes Ziel des CMM – wie auch vergleichbarer Modelle – ist die Steigerung der mit dem Reifegrad der Organisation korrelierte Wettbewerbsfähigkeit, die sich durch kürzere Entwicklungszeit, niedrigere Entwicklungskosten, höhere Produktivität und Qualität auszeichnet. Dazu ist im Anschluss an ein Assessment eine Diagnose (in Form eines Stärken- und Schwächenprofils), und darauf aufbauend eine Therapie (in Form eines zu implementierenden Maßnahmenkatalogs) durchzuführen. Die im Zusammenhang mit diesen Verbesserungsmaßnahmen entstehenden zusätzlichen Kosten werden durch sinkende Aufwände für Software-Tests und –Korrekturen kompensiert.

In der Literatur (u.a. auch im Internet) wird sehr oft über einen ROI (= Return on Investment) von bis zu fast 1000% berichtet (siehe z.b. [15])). Diese Zahlen werden oft dadurch begründet, dass die Kosten für Software-Korrekturen mit jeder Phase des Entwicklungs-Prozesses exponentiell steigen. In anderen Fällen ist die Anwendung von Reifegrad-Modellen durch die Ausschreibung oder Nachfrage nach Produkten mit hoher Qualität schon a priori vorgeschrieben.

Eine interessante Studie von Capers Jones [16]) verdeutlicht, dass ein hoher Qualitäts-Standard nicht notwendigerweise mit einer hohen Produktivität korreliert. Ganz im Gegenteil zeigt die Studie, dass Industrien mit hohen Qualitäts-Standards bezüglich Produkten und Entwicklungs-Prozessen eine sehr geringe Produktivität aufweisen (vor allem Software für den Militär- und Telekom-Sektor). Im Falle militärischer Software entfallen 52% der Kosten auf die Erstellung von Dokumenten. Daraus wird ersichtlich, dass durch die Anwendung des CMM-Modells die Prozessqualität steigt, die Gesamtproduktionskosten jedoch nicht notwendigerweise sinken müssen.

[15] Dion, R., Process Improvement and the Corporate Balance Sheet, in: IEEE Software, July 1993
[16] Jones, Capers, Assessment and Control of Software Risks, USA, 1994



	Prozess-Stufen	Prozess-Charakterisik	Hauptkriterien bzw. Key Process Areas
5.	Optimierender Prozess	Prozesse werden unter Einhaltung der in den unteren Stufen genannten Fortschritte kontinuierlich verbessert	Prozess-Änderungs-Management; Technologie-Innovations-Management; Fehlervermeidungs-Strategien
4.	Gesteuerter Prozess	Prozesse werden über Metriken quantitativ erfaßt und gesteuert; zuverlässige Qualitätskontrolle	Software-Qualitäts-Management; quantitatives Prozess-Management
3.	Definierter Prozess	Anwendung definierter Prozesse; zuverlässige Kosten- und Termineinhaltung	Peer-Reviews; Trainings-Programme; gruppenübergreifende Koordination; Software-Produkt-Engineering; integriertes Software-Management; organisationsweiter Prozess-Fokus; organisationsweite Prozess-Definition
2.	Wiederholbarer Prozess	Projekt-Management Praktiken werden angewandt; Prozesse individuell verschieden	Konfigurations-Management; Qualitätssicherung; Lieferanten-Management; Projektplanung- und verfolgung; Anforderungs-Management
1.	Initialer Prozess	Chaotischer Ad-hoc Prozess	keine

Abb. 8.1: CMM-Ansatz

Aufgrund der Erfüllung bzw. Nichterfüllung bestimmter Kriterien weist das CMM [17]) einer Software-Entwicklungs-Organisation einen bestimmten Reifegrad zu. Für das Erreichen einer bestimmten Prozessstufe wird die Erfüllung aller in den darunter liegenden Stufen definierten Kriterien (= Key Process Areas) vorausgesetzt – zumindest für die Berechnung der erreichten Stufe aufgrund eines Assessments (vgl. Abb. 8.1). In der Praxis verteilen sich die erfüllten Kriterien auf zumeist zwei bis drei Stufen. Eine Übersicht über die Techniken bzw. Methoden (= Key Practices) der einzelnen Key Process Areas ist im Anhang zu finden.

Interessante Artikel über den Praxiseinsatz des CMM im Einflussbereich des DoD werden im Magazin: „CROSS TALK – The Journal of Defense Software Engineering" [18]) publiziert. Ein Vergleich der Mitte 1998 und Mitte 2002 nach CMM weltweit zertifizierten Firmen zeigt einen Zuwachs an Level-4 Firmen von 11 auf 73 (davon 39 in den USA und 28 in Indien), bzw. an Level-5 Firmen von 4 auf 69 (davon 19 in den USA und 46 in Indien) [19]).

[17] Paulk, M., Chrissis, M., Curtis, B., Capability Maturity Model for Software, USA, 1993.
[18] http://www.stsc.hill.af.mil/CrossTalk/ , CrossTalk: The Journal of Defense Software Engineering, 2004
[19] http://www.sei.cmu.edu/sema/welcome.html , Software Engineering Measurement & Analysis (SEMA), 2004

Inhaltlich lassen sich von den Level-5 Firmen in den USA über 60% dem Militär-Sektor zurechnen. Der Rest entfällt fast ausschließlich auf Luftfahrt-, Raumfahrt-, und Telekommunikations-Software. Die im Jahr 2003 in Indien zertifizierten Level-5 Firmen produzieren zivile Software für die unterschiedlichsten Branchen.

Zur Relativierung dieser Zahlen muss angemerkt werden, dass die Angaben über Level-4 und Level-5 Reifegrade von den Firmen selbst publiziert wurden und dass nur wenige Firmen ihren (meist niedrigeren) CMM-Level überhaupt öffentlich publizieren. Des weiteren muss hinzugefügt werden, dass die bis 1993 vom SEI zertifizierten Firmen einen durchschnittlichen Reifegrad von 1,5 aufwiesen [20]). Selbst im Jahr 2003 muss davon ausgegangen werden, dass der durchschnittliche Reifegrad unter 2 liegt, zumal die Mehrzahl der Software-Entwicklungs-Organisationen in keinem anwendungs-kritischen Segment (z.b. Web-Publishing) arbeiten.

Der CMM-Ansatz wird unter der Bezeichnung „SW-CMM" (= Software-CMM) seit 1991 in der Version 1.0 und seit 1993 in der Version 1.1 vermarktet. Davon abgeleitet wurden vom SEI an der Carnegie Mellon University auch Rahmenwerke für die Disziplinen Systems-Engineering: „SE-CMM", Personal-Management: „P-CMM" und Software-Einkauf: „SA-CMM" (= Software Acquisition) entwickelt und vermarktet. Die derzeit (im Jahr 2004) aktuellste Veröffentlichung des SEI ist die „CMMI" (= Capability Maturity Model Integration) Product Suite in der Version 1.1 aus dem Jahr 2002: „CMMI^{SM} for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS, V1.1)".

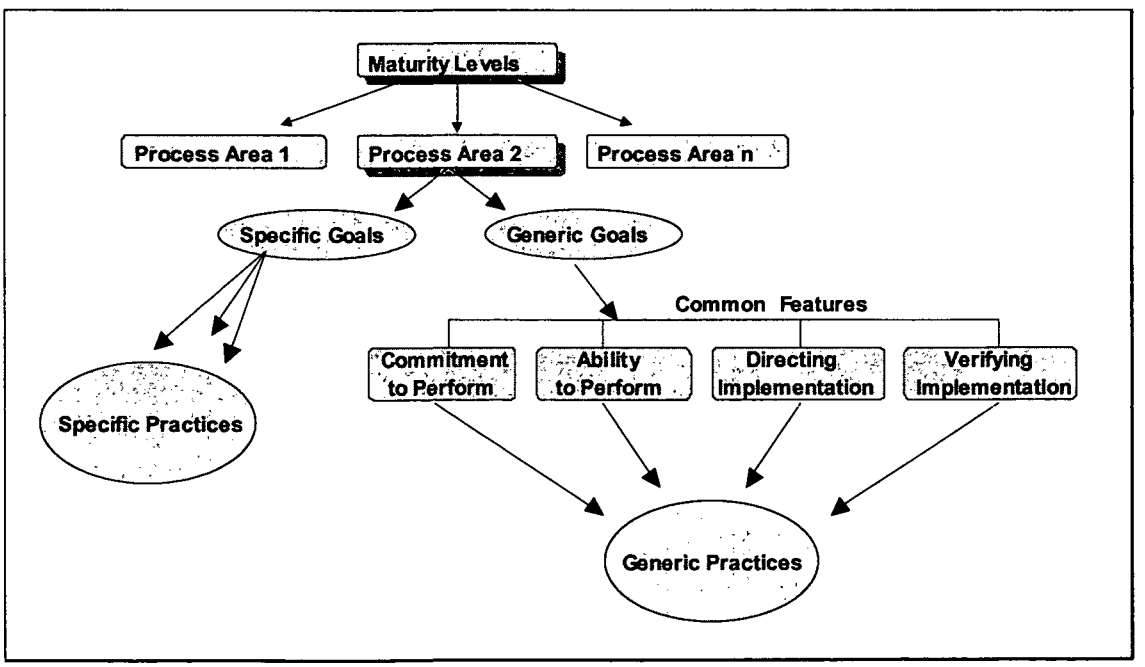


Abb. 8.2: CMMI Model Components

[20] Jones, Capers, Assessment and Control of Software Risks, USA, 1994

Die Zielsetzung des CMMI entsprechend der Staged Representation CMU/SEI-2002-TR-012 vom März 2002 [21]): "The CMMI Product Team's mission was to combine three source models – (1) Capability Maturity Model for Software (SW-CMM) v2.0 draft C, (2) Electronic Industries Alliance Interim Standard (EIA/IS) 731, and (3) Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98 – into a single improvement framework for use by organizations pursuing enterprise-wide process improvement." Die CMMI Modell-Komponenten zeigt Abb. 8.2 [22]). Ausgehend vom anvisierten Reifegrad werden allgemeine und spezifische Maßnahmen abgeleitet.

Auf Basis von vier Disziplinen, die auch als „Bodies of Knowledge" bezeichnet werden, bietet das CMMI-Rahmenwerk für jede Disziplin ein geeignetes Modell an. Hinsichtlich der Modell-Wahl ist im oben genannten Dokument folgendes zu finden:

"Systems Engineering:
Systems engineering covers the development of total systems, which may or may not include software. Systems engineers focus on transforming customer needs, expectations, and constraints into product solutions and supporting these product solutions throughout the life of the product.

Software Engineering:
Software engineering covers the development of software systems. Software engineers focus on applying systematic, disciplined, and quantifiable approaches to the development, operation, and maintenance of software.

Integrated Product and Process Development:
Integrated Product and Process Development (IPPD) is a systematic approach that achieves a timely collaboration of relevant stakeholders throughout the life of the product to better satisfy customer needs, expectations, and requirements. The processes to support an IPPD approach are integrated with the other processes in the organization. The IPPD process areas, specific goals, and specific practices alone cannot achieve IPPD. If a project or organization chooses IPPD, it performs the IPPD-specific practices concurrently with other specific practices used to produce products (e.g., the Engineering process areas). That is, if an organization or project wishes to use IPPD, it chooses a model with one or more disciplines in addition to selecting IPPD.

Supplier Sourcing:
As work efforts become more complex, projects may use suppliers to perform functions or add modifications to products that are specifically needed by the project. When those activities are critical, the project benefits from enhanced source analysis and from monitoring supplier activities before product delivery. The supplier sourcing discipline covers acquiring products from suppliers under these circumstances."

Der Vollständigkeit wegen sei noch auf die Standard CMMI-Bewertungsmethode für Prozess-Verbesserungen hingewiesen, die vom SEI unter dem Begriff „SCAMPI^{SM}" vermarktet wird. Sie wird zur Messung des CMMI-Levels eingesetzt.

---

[21] http://www.sei.cmu.edu/cmmi/general/ , CMMI General Information, 2004
[22] http://www.sei.cmu.edu/cmmi/models/ , CMMI-Models, CMMI-SE/SW/IPPD/SS, V1.1, 2004

8.2.5 ISO/IEC 15504 – Ansatz (SPICE)

Aufgrund der Historie sind in (zumindest europäischen) Software-Entwicklungs-Organisationen die Vorgaben von ISO 900x bzw. die Verbesserungs-Hinweise des CMM gleichzeitig relevant. Dies war Anlass zur Integration beider Ansätze, welche zunächst unter dem Namen „SPICE" (= Software Process Improvement and Capability Determination) und später dann als ISO 15504 bei der International Standardization Organization (ISO) und der International Electrotechnical Commission (IEC) entwickelt wurde bzw. wird (Stand: September 2003). Inwieweit sich die beiden Quellen von SPICE decken zeigt Tab. 8.3.

Grundsätzlich unterscheiden sich die beiden Ansätze darin, dass die ISO 900x Zertifizierung der Nachweis für ein Qualitäts-Management-System entsprechend der Norm ist, während der CMM-Ansatz ein Werkzeug zur Qualitäts- und Produktivitäts-Steigerung des gesamten Software-Entwicklungs-Prozesses ist. Im Falle der USA ist die CMM-Zertifizierung darüber hinaus die Voraussetzung für Aufträge von Bundesbehörden (z.b. vom DoD). Verglichen mit der TQM-Philosophie sind beide Ansätze (CMM und ISO 9000) konkreter formuliert, beziehen dafür aber die soziale Komponente (Mitarbeiter, Kunden) weniger in die Überlegungen mit ein.

	CMM	ISO 900x
Gegenstand	Software-Entwicklungs-Prozesse	Vielzahl industrieller Organisationen, Produkte und Prozesse
Ziel	Assessment und Maßnahmen zur Prozess-Verbesserung	Qualifikationsnachweis zur Erzeugung qualitätsgerechter Produkte
Status	Hilfsmittel zur Prozess-Analyse und -Verbesserung	Fixer Industriestandard
Forderungen	Hierarchie an Forderungen in Abhängigkeit der Reifestufe	ausnahmslos zu erfüllende Minimalanforderungen
Basis	flexibles Capability Maturity Modell	starrer Normentext
Ergebnis	Ist-Stand, Stärken- und Schwächen-Profil	Anerkanntes Zertifikat
Kosten vs. Nutzen	Ziel ist ein positiver ROI	Nutzen nur durch erteiltes Zertifikat

Tab. 8.3: CMM versus ISO 900x

Um die unterschiedlichen Ansätze – vor allem CMM und ISO 9000 – in einem einzigen Modell zu integrieren, wird seit 1993 unter dem Dach der ISO/IEC der SPICE-Ansatz entwickelt. Ziel ist es, einen umfassenden Rahmen zur Bewertung, Verbesserung und Reifegrad-Feststellung von Software-Prozessen zur Verfügung zu stellen – im Originaltext: „Improve product quality

through proven, consistent and reliable assessment of the state of an organization' software process and the employment of the results of these assessments as part of coherent improvement programs."

Die Software-Prozess-Domänen die vom ISO 15504 adressiert werden betreffen u.a. den Einkauf, die Lieferung, die Entwicklung, den Betrieb, die Wartung und den Service von Software. Die endgültige Version – der Standard – liegt zum Zeitpunkt September 2003 noch nicht vor. Als aktuellste Grundlage dienen stattdessen die im Vorfeld verabschiedeten Technical Reports der ISO/IEC aus den Jahren 1998 und 1999. Diese sind im Detail [23]):

- TR 15504-1:1998 - Part 1: Concepts and introductory guide
- TR 15504-2:1998 - Part 2: A reference model for processes and process capability
- TR 15504-3:1998 - Part 3: Performing an assessment
- TR 15504-4:1998 - Part 4: Guide to performing assessments
- TR 15504-5:1999 - Part 5: An assessment model and indicator guidance
- TR 15504-6:1998 - Part 6: Guide to competency of assessors
- TR 15504-7:1998 - Part 7: Guide for use in process improvement
- TR 15504-8:1998 - Part 8: Guide for use in determining supplier process capability
- TR 15504-9:1998 - Part 9: Vocabulary

Im Part 2: „A reference model for processes and process capability" wird das Referenz-Modell definiert. Es liefert die Basis für jedes konkrete Modell das zum Zweck des Software-Prozess-Assessments eingesetzt wird. Hinsichtlich der Architektur ist das Referenz- bzw. Assessment-Modell in zwei Dimensionen gegliedert (vgl. Abb. 8.3):

- die Prozess-Dimension dient zur Bewertung der Vollständigkeit von Prozessen
- die Reifegrad-Dimension dient zur Bewertung der Leistungsfähigkeit von Prozessen

In der Prozess-Dimension werden 29 Prozesse zu 5 Prozess-Kategorien verdichtet:

- Kunden-Lieferanten-Prozess-Kategorie (z.b. Kundendienst)
- Entwicklungs-Prozess-Kategorie (z.b. Definitionsphase)
- Unterstützende Prozesse-Kategorie (z.b. Qualitätssicherung)
- Management-Prozess-Kategorie (z.b. Projektmanagement)
- Organisations-Prozess-Kategorie (z.b. Personalmanagement)

Das jeweilige Prozess-Ziel wird durch „Grundlegende Aktivitäten" beschrieben, von denen es in Summe 200 gibt. Diese werden wiederum anhand von „Arbeitsprodukten" (Ein- und Ausgabe-Produkte) charakterisiert. Anhand dieser Aktivitäten bzw. Arbeitsprodukte wird die Prozess-Dimension im Assessment bewertet.

---

[23] http://www.iso.ch/iso/en/stdsdevelopment/tc/tclist/TechnicalCommitteeStandardsListPage.TechnicalCommitteeStandardsList?COMMID=40 , Standards and/or guides of JTC 1/SC 7, 2004

In der Reifegrad-Dimension werden die Prozesse einem von 6 Reifegrad-Stufen zugeordnet. Diese sind Indikatoren für die Vollständigkeit und Leistungsfähigkeit des jeweiligen Prozesses im Unternehmen. Dazu wird die Leistungsfähigkeit eines jeden Prozesses durch 9 Prozessattribute beurteilt (Level 1 ist ein Prozessattribut zugeordnet, Level 2 bis 5 sind jeweils zwei Prozess-Attribute zugeordnet), die anhand von 4 Ausprägungen

„none"	(= nicht erfüllt (N))	00% - 15%
„partially"	(= teilweise erfüllt (P))	16% - 50%
„largely"	(= größtenteils erfüllt (L))	51% - 85%
„fully"	(= voll erfüllt (F))	85% - 100%

je eine Charakteristik des Prozesses messen. Die Bestimmung der Ausprägung selbst hängt von der Erfüllung bzw. Nicht-Erfüllung von Management-Aktivitäten ab, die ihrerseits wieder auf Ressourcen- und Infrastruktur-Charakteristiken Bezug nehmen. Hierbei gilt ein Capability-Level als erreicht, wenn alle Prozess-Attribute darunterliegender Level voll erfüllt sind (F) und jene, dessen Level es zu erreichen gilt, größtenteils (L) oder voll (F) erfüllt sind. In der Abb. 8.4 wird dies anhand eines Beispiels verdeutlicht [24]).

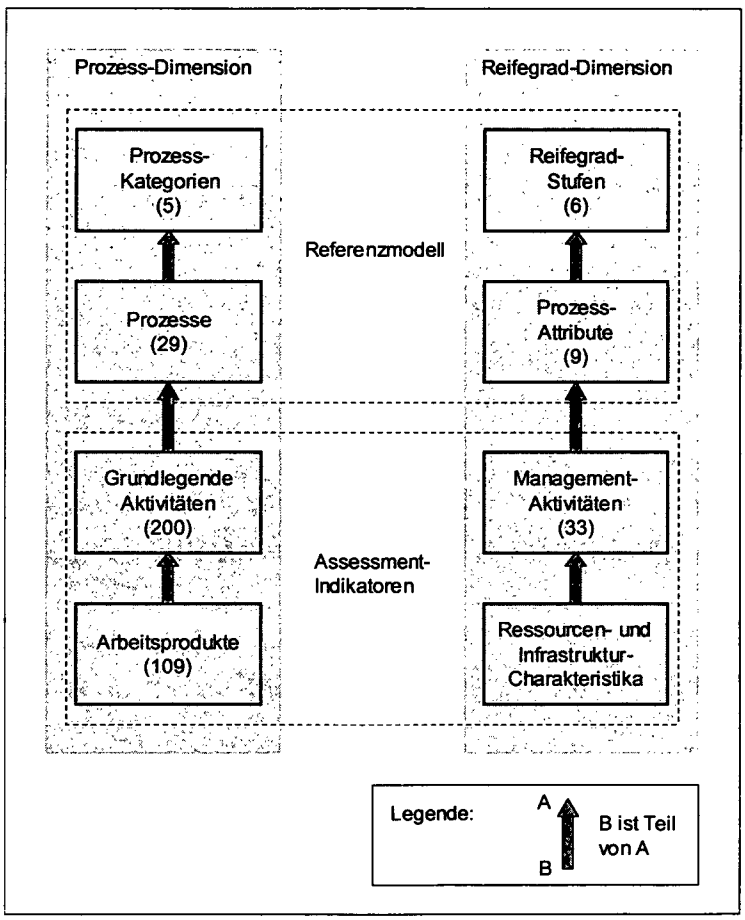


Abb. 8.3: SPICE - Referenz- und Assessment-Modell

Da der SPICE-Ansatz auf dem CMM aufbaut, entsprechen die 6 Stufen des SPICE (Level 0 bis 5) in etwa den 5 Stufen des CMM. Da sich aber auch das CMM weiterentwickelt und durch die SPICE-Aktivitäten beeinflusst wird, muss die Zukunft erst zeigen, welche Ansätze in der Praxis schlussendlich dominieren werden.

[24] http://www.itq.ch/ , Qualität und Informatik, Capability Dimension von SPICE/ISO 15504, Zürich, 2004

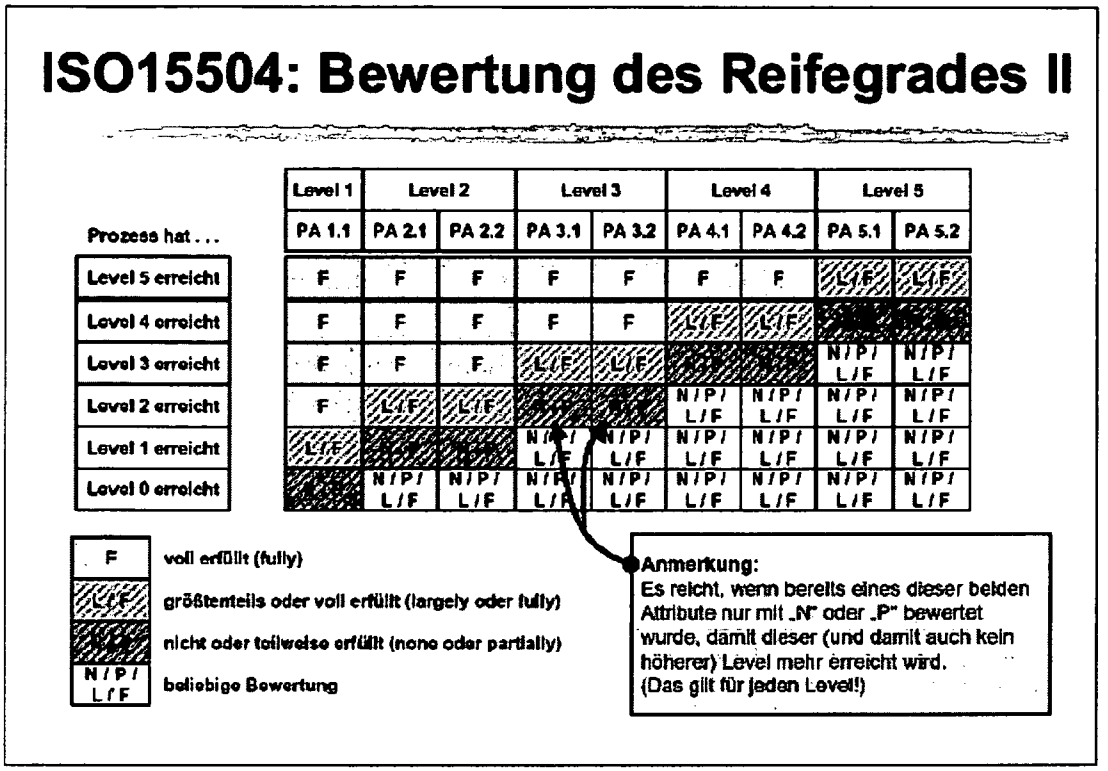


Abb. 8.4: Bewertung des 2. Reifegrades (Level 2)

Neben dem CMM-Ansatz und dem ISO 9001 (ISO 9000-3) Standard haben noch folgende Ansätze, Modelle, technische Reports oder Standards Einfluss auf den demnächst verabschiedeten Standard: ISO/IEC 15504:

- ISO/IEC 12207 (widmet sich den Inhalten der Software-Prozesse zur Entwicklung, dem Einkauf, der Bereitstellung und dem Management – im Gegensatz zur Bewertung und Verbesserung wie ISO/IEC 15504))
- ISO 9004-4 (adressiert die Qualitäts-Verbesserung)
- Trillium (inhaltlich ähnlich dem CMM)
- TickIT (inhaltlich ähnlich ISO 9001 und ISO 9000-3)

8.2.6 Application of Metrics in Industry (AMI)

Dieser Ansatz [25]) wurde in einem ESPRIT-Projekt entwickelt und schlägt als Bewertungsmethode das CMM und als Messansatz das GQM-Modell vor. Der AMI-Ansatz konzentriert sich auf die Bewertung des Standards der Software-Entwicklung und die

[25] A Quantitative Approach to Software Management, Addison-Wesley, 1995.

Einführung von Maßstäben, um einen Verbesserungsansatz zu unterstützen. Die dafür notwendigen Schritte sind:

1. **Assess**
   - Bewertung der Umgebung
   - Definition der Ziele
   - Validierung der Ziele
2. **Analyze**
   - Verfeinerungen der Ziele in Subziele
   - Überprüfung der Konsistenz des resultierenden Zielbaums
   - Diskussion zur Identifizierung von Maßen
3. **Metricate**
   - Entwurf und Validierung des Messplans
   - Sammlung relevanter Daten
   - Verifizierung der Rohdaten
4. **Improve**
   - Verteilung, Analyse und Diskussion der Messdaten
   - Validierung der Maße
   - Die Daten werden in Bezug zu den Zielen gesetzt und Maßnahmen verwirklicht

Ein Vergleich der Modelle führt zu folgender Übersicht [26]) (vgl. Tab. 8.4):

Modellvergleich	**Ad-hoc**	**CMM**	**AMI**	**QIP**
Ingenieurwissenschaftliche Vorgehensweise	-	+	0	+
Empirische Vorgehensweise	-	-	-	+
Messen	0	+	+	+
Modellieren	-	+	-	+
Wiederverwenden	-	0	-	+
Einmalige Verbesserungen	0	+	+	+
Wiederholbare Verbesserungen	-	+	0	+
Nachweisbare Verbesserungen	-	+	-	+

Tab. 8.4: Qualitäts-Modelle im Vergleich

Legende: "-" keine Unterstützung
"0" teilweise Unterstützung
"+" volle Unterstützung

[26] Ebert, Christof, Dumke, Reiner, (hrsg.), Software-Metriken in der Praxis, Berlin, 1996

8.3 Metriken

8.3.1 Grundlagen

Erste Veröffentlichungen zu Software-Maßen bzw. Software-Metriken (obwohl synonym verwendet, ist dies bei Software der häufiger anzutreffende Begriff) gibt es zumindest seit 1968 [27]). Der Einsatz von Metriken lag zu Beginn in der quantitativen Erfassung der Beziehung zwischen Qualität, Entwicklungs- und Wartungsaufwand. Zu diesem Zweck wurden eine Reihe von Kennzahlen definiert, die zunächst noch kaum Praxisbezug hatten.

Eine der ersten Metriken war eine Kennzahl zur Messung des Programm-Umfangs. Sie zählt die Zeilen des Programmcodes und wird als LOC (= **Lines of Code**) bezeichnet. Mit der Entwicklung unterschiedlicher Programmiersprachen wurde es jedoch immer schwieriger die Quelltexte zu vergleichen. Eine Tabelle in [28]) gibt Aufschluss über die Mächtigkeit der höheren Programmiersprachen gegenüber Assembler – der ersten Sprache, die die Prozessorbefehle in einer groben Näherung 1:1 abbildet. Dieser Tabelle zufolge entsprechen 100.000 Zeilen Assembler-Code etwa 25.000 Zeilen PL/1-Code oder 5.000 Zeilen Code in einer Query-Language. Einen weiteren Einfluss auf das Ergebnis hat die Zählweise der Code-Zeilen. Hierfür erwies sich als sicherste Methode die Zählung der Sprachkonstrukt-Begrenzer (= Delimiter).

Bezieht man diese Mengenangabe hinsichtlich der Code-Zeilen auf die dafür benötigte Zeit, erhält man Produktivitäts-Maße – z.b. in LOC / MM (= Lines of Code per ManMonth). In diesem Zusammenhang müssen allerdings jene Zeilen separat berücksichtigt werden, die bereits existieren und in dem betreffenden Programmstück wiederverwendet wurden, und jene, die zwei- oder mehrmals abgeändert (korrigiert) wurden. Unter Einbeziehung der oben erwähnten Tabelle kann man die Parameter folgendermaßen zu einer Formel zusammenbauen:

$$AE_{LOC} = (ALL_{LOC} + CH_{LOC} - 0{,}8 * RE_{LOC}) * HOL_{EQ}$$

AE_{LOC} ...	Lines of Code in Assembler als Maß für den Programm-Umfang, den Aufwand, und die Produktivität (auf die Ressourcen Zeit- und Personaleinsatz bezogen)
ALL_{LOC} ...	Anzahl aller Programm-Zeilen
CH_{LOC} ...	Anzahl an geänderten Programm-Zeilen
RE_{LOC} ...	Anzahl wiederverwendeter Programm-Zeilen aus einem anderen Programm (in dieser Formel wird der eingesparte Aufwand mit 80% berücksichtigt)
HOL_{EQ} ...	Faktor, der die Mächtigkeit einer höheren Programmiersprache

[27] Rubey, R.J., Hartwick, R.D., Quantitative Measurement of Program Quality, 23. National Conference ACM; 1968

[28] Thaller, Georg E., Software-Metriken einsetzen, bewerten und messen, Berlin, 2000

im Vergleich zu Assembler angibt

Aufgrund der Konstrukte (z.b. Vererbung, Polymorphismus, usw.) objektorientierter Sprachen (z.b. C++ oder Ada) sind für die Messung des Umfangs zusätzlich folgende Metriken maßgebend:

- Breite und Höhe der Vererbungshierarchien
- Anteil wiederverwendeter Klassen, Operationen, usw.
- Anzahl der Klassenattribute und –operationen
- Ausmaß der Anwendung des Polymorphismus

Die Anwendung herkömmlicher Metriken (z.b. Dateiumfang des Quellcodes) führen aufgrund der Konstrukte objektorientierter Sprachen zu einer geringeren Zeilenanzahl.

Die Nachteile dieser Umfangsmetrik(en) ergeben sich aus den nur unzureichend bestimmbaren Parametern, und dem Umstand, dass die Aufstellung von Prognosen über den Programm-Umfang a priori nicht möglich ist. Eine dieser Kritik Rechnung tragende Metrik ist jene der **Function Points**. Hierbei ist nicht das Endprodukt (die für den Menschen lesbare Form der Software im Gegensatz zum daraus erzeugten binären Maschinen-Code), sondern der Anforderungskatalog des Kunden das zu (ver)messende Objekt. Details zum Prozess der Function-Point-Methode werden im Kapitel 7 beschrieben.

Der Function-Point-Methode liegt die Hypothese zugrunde, dass der Aufwand zur Produktion einer bestimmten Software durch die Beurteilung der Komplexität der Ein- und Ausgaben, der Dateizugriffe und Abfragen bestimmbar ist. Dieser Ansatz eignet sich daher insbesondere zur Aufwands-Abschätzung kommerzieller Software, in der Datenbank-Anwendungen im Gegensatz zur Algorithmen-Entwicklung eine wesentliche Rolle spielen. Die in einem ersten Schritt ermittelten Function-Points (FPs) – als Metrik für den Aufwand (der Programmumfang ist hierbei weniger der Fokus) – werden in einem zweiten Schritt über die Relation „FP <-> MM“ zu Aufwandszahlen mit der Einheit „MM“ konvertiert. Die Relation „FP <-> MM“ wird durch eine nichtlineare Kurve, basierend auf historischen Daten, dargestellt.

Diese nichtlineare Relation wird durch sämtliche Einflussgrößen, die aus der Zusammenarbeit von Teams und unterschiedlichen Software-Anwendungen resultieren, bestimmt. Siehe hierzu Tab. 8.5 [29]), wo die Verteilung der Aufwände auf die unterschiedlichen Phasen – abhängig vom Programmumfang in FPs – gezeigt wird.

[29] Jones, Capers, Applied Software Measurements, Assuring Productivity and Quality, New York, 1991

Größe in Function Points	Management und Support	Fehlerkorrektur	Dokument-Erstellung	Codierung
40.960	18	37	33	12
20.480	17	36	32	15
10.240	16	35	31	18
5.120	15	34	30	21
2.560	14	33	29	24
1.280	14	30	26	30
640	13	28	24	35
320	12	26	22	40
160	12	23	20	45
80	11	20	15	54
40	11	19	12	58
20	11	18	10	61
10	11	17	7	65
5	11	16	5	68

Tab. 8.5: Verschiebung der Aufwandsverteilung mit Zunahme des Programm-Umfangs

8.3.1.1 Gliederung von Metriken

Metriken als Instrument zur Beschreibung des Softwarezustandes sind in der Theorie stets Teil von umfassenden Modellen. Meistens lassen sich derartige Modelle jedoch nur mit erheblichem Aufwand verwirklichen, oder sind aus anderen Gründen zur vollständigen Anwendung nicht geeignet. Aufgrund dieser Einschränkungen wird eine geeignete Auswahl an Metriken getroffen und in den Software-Prozess mitaufgenommen. Gliedert man die in der Praxis zur Anwendung kommenden Metriken, ergibt sich folgende Übersicht:

- Umfangsmetriken
  z.b. Lines of Code, Function Points
- Zeit- und Aufwandsmetriken
  z.b. Meilensteintrendanalysen, Kostentrendanalysen
- Strukturmetriken
  z.b. Änderungshäufigkeit von Leistungsmerkmalen, codeabhängige Metriken
- Fehlermetriken
  z.b. Fehlerentdeckungsrate, Fehler-Verteilung auf Teilsysteme und Phasen
- Testabdeckungsmetriken
  z.b. Testumfang, Testschrittabarbeitung, Aussagekraft von Tests, Prognosen

In Tab. 8.6 [30]) werden Objekte eines Software-Entwicklungs-Prozesses aufgezählt, für welche Metriken grundsätzlich definiert werden könnten.

Measurable Entities in a Software Process				
Things Received or Used	Activities and their Elements	Things Consumed	Things Held or Retained	Things Produced
Products and by-products from other processes Ideas, concepts Resources: • People • Facilities • Tools • Raw materials • Energy • Money • Time Guidlines and directions: • Policies • Procedures • Goals • Constraints • Rules • Laws • Regulations • Training • Instructions	Processes and controllers: • Requirements analysis • Designing • Coding • Testing • Configuration control • Change control • Problem management • Reviewing • Inspecting • Integrating Flow paths • Product paths • Resource paths • Data paths • Control paths Buffers and dampers: • Queues • Stacks • Bins	Resources: • Effort • Raw materials • Energy • Money • Time	People Facilities Tools Materials Work in process Data Knowledge Experience	Products: • Requirements • Specifications • Designs • Units • Modules • Test cases • Test results • Tested components • Documentation • Defects • Defect reports • Change requests • Data • Acquired materials • Other artefacts By-products: • Knowledge • Experience • Skills • Process improvements • Data • Good will • Satisfied customers

Tab. 8.6: Examples of Measurable Entities in a Software Process

[30] Florac, William A., Carleton, Anita D., Measuring the Software Process, SEI Series, Massachusetts, 1999

8.3.2 Eine Auswahl praxisrelevanter Metriken

Bezogen auf die Stufen des SW-CMM wird der Einsatz von Metriken erst bei einer sehr hohen Reifegrad-Stufe einer Software-Organisation gefordert (Stufe 4 von insgesamt 5 Stufen). Dem steht gegenüber, dass bei Festpreisangeboten zur Beherrschung von Risken und einer besseren allgemeinen Planung Metriken dennoch eingesetzt werden.

Der erste Schritt zur Einführung eines effizienten Metrik-Programms ist eine Zusammenstellung einfach zu bestimmender Metriken, die zudem eindeutige Aussagen über den Prozess-Status zulassen. Ein praktisches Hilfsinstrument ist die Einführung eines Qualitäts-Management-Systems mit dem Ziel der stufenweisen Verbesserung der Prozesse (z.b. CMM). Um die Zusammenstellung der wesentlichen Metriken zu erleichtern, sind Vorlagen besonders hilfreich. In diesem Zusammenhang muss betont werden, dass die Komplexität von Metrik-Systemen in der Praxis stets unterschätzt wird. Vielfach besteht der Wunsch nach wissenschaftlich fundierten Systemen, die jedoch in der Software-Entwicklung kaum über den gesamten Prozess konsistent angewandt werden können.

Im Anschluss werden die am häufigsten angewandten Metriken im Detail diskutiert. Der Vollständigkeit wegen sei auch auf die wachsende Anzahl an Vorschlägen in den für die Software-Entwicklung gültigen Normen (z.b. ISO 900x) und Software-Qualitäts-Modellen (z.b. SW-CMM) hingewiesen.

8.3.2.1 Stabilitäts-Index

Der Stabilitäts-Index ist eine Metrik, die den Grad der „Neuheit" eines Software-Produkts bzw. einer neuen Version kennzeichnet. Man geht dabei von der Annahme aus, dass proportional zum Anteil der wiederverwendeten Software die Stabilität bzw. Fehlerfreiheit zunimmt. Enthält ein neues Release eine große Anzahl neuer oder geänderter Funktionen muss tendenziell mit einer geringeren MTTF (= Mean Time To Failure) gerechnet werden.

$$I_{St} = (F_{all} - (F_{new} + F_{ch} + F_{can})) / F_{all}$$

I_{St} ...	Stabilitäts-Index
F_{all} ...	Gesamtzahl der in der aktuellen Version enthaltenen Funktionen
F_{new} ...	Anzahl der in der aktuellen Version neuen Funktionen gegenüber der Vorgänger-Version
F_{ch}	Anzahl der in der aktuellen Version geänderten Funktionen gegenüber der Vorgänger-Version
F_{can} ...	Anzahl der in der aktuellen Version gestrichenen Funktionen gegenüber der Vorgänger-Version

Der Wertebereich dieser Metrik liegt zwischen 0 (komplette Neuentwicklung) und 1 (die Software wurde nicht verändert). Je näher dieser Index bei 0 liegt umso größer ist der Testaufwand.

Abgesehen von der Funktionalität als Basis für das Ausmaß der Änderungen kann auch die Dokumentation, der Entwurf oder der Quelltext (auch als Quellcode bezeichnet) herangezogen werden. Dabei vergleicht man die Änderungen jeweils mit jenen der Vorgänger-Version oder den Status der vergangenen Woche, Monat, usw. So kann z.b. auch eine Abnahme der Change-Requests als Grad der Reifung eines Software-Produkts interpretiert werden.

8.3.2.2 Textuelle Komplexität des Quellcodes

Einer der grundlegendsten Faktoren für hochqualitative Programme ist ein einfach zu lesender Quelltext oder –code. Die Einhaltung einer festgelegten Struktur, aussagekräftige Bezeichner für Variablen und Funktionen, eingefügte Kommentare (und vieles mehr) tragen zur Übersichtlichkeit und damit zum leichteren Auffinden von Fehlern (in Reviews) in erheblichem Maße bei. Zur Messung dieses Sachverhalts hat Halstead [31]) – der zu jenen Pionieren zählt, die Metriken schon sehr früh in den Dienst der Software-Entwicklung stellten – mehrere Metriken vorgeschlagen, u.a.:

ein Maß für die Schwierigkeit ein Programm zu schreiben und zu verstehen:
$D = (n_1 * N_2) / (2 * n_2)$

ein Maß für die Größe des Vokabulars:
$n = n_1 + n_2$

ein Maß für die Länge der Implementierung (Programmlänge):
$N = N_1 + N_2$

wobei:

n_1 ...	Anzahl der unterschiedlichen Operatoren (z.b. „+“, „()“, „IF“, usw.)
n_2 ...	Anzahl der unterschiedlichen Operanden (z.b. Variable, Konstante, usw.)
N_1 ...	Gesamtzahl der Operatoren im Quellcode
N_2 ...	Gesamtzahl der Operanden im Quellcode

Während diese Metrik einfach und für alle Programmiersprachen anwendbar ist, hat sie den Nachteil, dass sie nur den Implementierungs-Aspekt berücksichtigt.

[31] Halstead, Maurice H., Elements of Software Science, New York, 1977

8.3.2.3 Strukturelle Komplexität des Quellcodes

Die hierfür bekannteste Metrik ist die McCabe-Metrik [32]), deren Basis der Kontrollflussgraph ist. Sie geht von der Tatsache aus, dass ein Programm mit zunehmender Anzahl an Verzweigungen und Schleifen schwieriger zu verstehen und nachzuvollziehen ist.

$V(G) = e - n + (2 * p)$

V(G) ... Zyklomatische Zahl des Graphen G (Komplexitätsmaß nach McCabe)
e ... Anzahl der Kanten des Kontrollfluss-Graphen
n ... Anzahl der Knoten
p ... Anzahl der verbundenen Komponenten

Diese Metrik lässt sich einfach berechnen und kann als Referenz für die Testabdeckung verwendet werden. Ihr Nachteil liegt in der Anwendung bei objektorientierten Sprachen, da die einzelnen Module (Objekte) für sich genommen eine sehr geringe Komplexität aufweisen, wodurch die Metrik an Aussagekraft verliert. Im Falle objektorientierter Sprachen sind folgende Kriterien für die strukturelle Komplexität und damit Qualität (durch geringe Fehlerwahrscheinlichkeiten) maßgebend [33]):

- Tiefe des Vererbungsbaumes (schlechter mit wachsender Tiefe)
- Anzahl der Kinder einer Klasse (schlechter mit geringerer Anzahl an Kindern bzw. geringerer Wiederverwendung der Oberklasse)
- Anzahl der in jeder Klasse definierten Funktionen und Operatoren (schlechter mit wachsender Anzahl)
- Kopplung zwischen den Klassen (schlechter mit wachsender Kopplung bzw. dem zunehmenden gegenseitigen Aufrufen von Methoden und Variablen)

8.3.2.4 Grad der Wiederverwendung

Grundsätzlich lässt sich der Grad der Wiederverwendung als Formel folgendermaßen darstellen:

$W = (S_w / (S_w + S_n)) * 100\%$

W ... wiederverwendeter Software-Anteil in %
S_w ... wiederverwendete Software in z.b. Lines of Code oder Function Points

---

[32] McCabe, T.J., A Complexity Measure, Structured Testing, in: IEEE Computer Society Press, 1983, p. 3-15
[33] Ebert, Christof, Dumke, Reiner, (hrsg.), Software-Metriken in der Praxis, Berlin, 1996

S_n ... neu erstellte Software in der gleichen Einheit wie S_w

Daraus lassen sich in der Folge auch Metriken zur Kosteneinsparung ableiten. Erleichtert wird die Messung der Wiederverwendung durch objektorientierte Programmiersprachen. Dort bezieht sich der Grad der Wiederverwendung auf Klassen, die einfach zu zählen sind (S_w würde für die Anzahl wiederverwendeter Klassen stehen; S_n für neu geschriebene Klassen). Hierbei ist erwähnenswert, dass es für viele Anwendungen kommerziell erhältliche Klassenbibliotheken gibt, die wesentlich für die Verringerung der Fehlerwahrscheinlichkeit beitragen.

8.3.2.5 Fehler-Metriken

Neben den Metriken zur Kennzeichnung des Programmumfangs gehören die Metriken der Testphase zu den ersten die in die Praxis der Software-Entwicklung Einzug hielten. Im folgenden werden hierzu einige Fehler-Metriken vorgestellt.

Zu Beginn eines jeden Software-Projekts wäre zur Planung der Testphase eine einigermaßen genaue Vorhersage des Umfangs der auftretenden Fehler – eine **Fehlervorhersage-Metrik** – wünschenswert. Diese Metrik basiert auf statistischen Erfahrungswerten und kann nur auf empirischem Wege für annähernd identische Entwicklungs-Umgebungen sinnvoll angewandt werden. Da sie durch eine Vielzahl von Einflüssen bestimmt wird, existieren in der Literatur nur Fallstudien aber keine generellen Ansätze. Ein Beispiel hierfür ist die Formel von Muneo Takahashi und Yuji Kamayachi von der japanischen Telefongesellschaft NTT [34]):

F = C1 + C2 * (CHL / kLOC) – C3 * EXP – C4 * (DOC / kLOC)
mit den Konstanten: C1 = 67,98; C2 = 0,46; C3 = 9,69; C4 = 0,08

F ... Fehlerrate pro 1000 Lines of Code
CHL ... Anzahl der Änderungen am Lastenheft während des Entwicklungs-Prozesses
kLOC ... Code-Umfang in 1000 Lines of Code
EXP ... durchschnittliche Erfahrung eines Entwicklungsteams in Jahren
DOC ... Anzahl der Änderungen im Entwurf während des gesamten Entwicklungs-Prozesses

In verfeinerten Analysen wird die Fehlerentdeckungsrate auf die Phasen des Entwicklungs-Prozesses verteilt. Dabei gilt grundsätzlich, dass die Kosten für die Fehlerbehebung mit jeder Phase in der sie nicht entdeckt wurden, zunehmen. In umgekehrter Richtung kann man

[34] Takahashi, Muneo, Kamayachi, Yuji, An Empirical Study of a Model for Program Error Prediction, in: IEEE Transactions on Software Engineering, Vol. 15/1, January 1989

aufgrund der Fehlerentdeckungsrate pro Phase Rückschlüsse auf den Reifegrad des Software-Entwicklungsprozesses ziehen.

In der Folge können die Fehler hinsichtlich der Ursache und des zur Behebung erforderlichen Aufwands näher charakterisiert werden. Hierzu werden in der Praxis in periodischen Abständen Fehler-Reports erstellt, die die neu gefundenen, noch offenen und bereits korrigierten Fehler enthalten. Daraus lässt sich dann der **gewichtete Fehlerindex** bilden:

$$I = w_1 * (E_1 / G) + w_2 * (E_2 / G) + w_3 * (E_3 / G)$$

I ...	gewichteter Fehlerindex
w_i ...	Wichtungsfaktoren der Fehlerklasse i
	(z.b. $w_1 = 0,7$; $w_2 = 0,25$; $w_3 = 0,05$)
E_i ...	Anzahl der Fehler der Fehlerklasse i
G ...	Gesamtzahl der gefundenen bzw. bekannten Fehler

Aus dem Verlauf der während der verschiedenen Phasen gefundenen Fehler können Abschätzungen über die Restfehlerrate für die geplante Nutzungsdauer der Software abgeleitet werden. Diese Metriken werden insbesondere bei sicherheitskritischer Software (z.b. in der Medizin, Luft- und Raumfahrt, Atomreaktor-Technologie, usw.) eingesetzt. Andererseits können sie unter Umständen aus vertraglicher Sicht von Relevanz sein. Und zwar dann, wenn der Vertrag zur Software-Erstellung Formulierungen über die Mean-Time-to-Failure (MTTF) oder Mean-Time-between-Failure (MTBF) enthält. Diese Unwägbarkeiten werden durch ein dem Vertrag zugrunde gelegtes mathematisches Modell (zur Prognose der Restfehler aus einfach zu ermittelnden Parametern) zumeist aufgehoben.

9. Organisationsgestaltung

Aufgrund der personalintensiven Eigenschaft der Software-Entwicklung spielt die Organisationsgestaltung eine wichtige Rolle. Während die Ablauforganisation an anderer Stelle im Rahmen der Entwicklungs-Prozesse beschrieben wird, sollen hier die Gestaltungsfelder erläutert werden. Darüber hinaus werden im Anschluss daran zwei wichtige dynamische Konzepte präsentiert (Change-Management-Konzept und Stakeholder-Konzept). Auf deren Prinzipien basierend kann die Organisation eines Softwarehauses bei Bedarf umgestaltet werden, wobei die Interessen der Beteiligten weitestgehend Berücksichtigung finden.

9.1 Gestaltungsmodelle für die Organisation eines Softwarehauses

Im Rahmen eines ganzheitlichen Gestaltungsansatzes sollten folgende Grundsätze Berücksichtigung finden [1]):

- Systemorientierung (vgl. hierzu das einleitende Kapitel 2)
  Das zentrale Augenmerk liegt hierbei bei einem abgeschlossenen System, das über 3 Kanäle mit der Außenwelt in Verbindung steht (Input, Output, Umwelt).
- Situationsorientierung (vgl. hierzu Kapitel 11)
  Situationsabhängig lassen sich erfolgserprobte Basisstrukturen weiter optimieren.
- Zielorientierung (vgl. hierzu Kapitel 3)
  Ein wichtiges Gestaltungskriterium ist die Zielsetzung der Organisation.
- Inhalts- und Vorgehensorientierung
  Während die ersten 3 Punkte die Anfangs- und Randbedingungen sowie den Zweck der Organisation zur Diskussion stellen, rückt dieser Punkt die Subjekte (sozio-organisatorischer Gestaltungsbereich) und Objekte (techno-organisatorischer Gestaltungsbereich) in den Mittelpunkt. Im **Modell der 5 Gestaltungsfelder** [2]) wird dieser Grundsatz in 5 Gestaltungsfeldern abgebildet. Für ein Softwarehaus sind alle Felder gleichermaßen von Bedeutung.

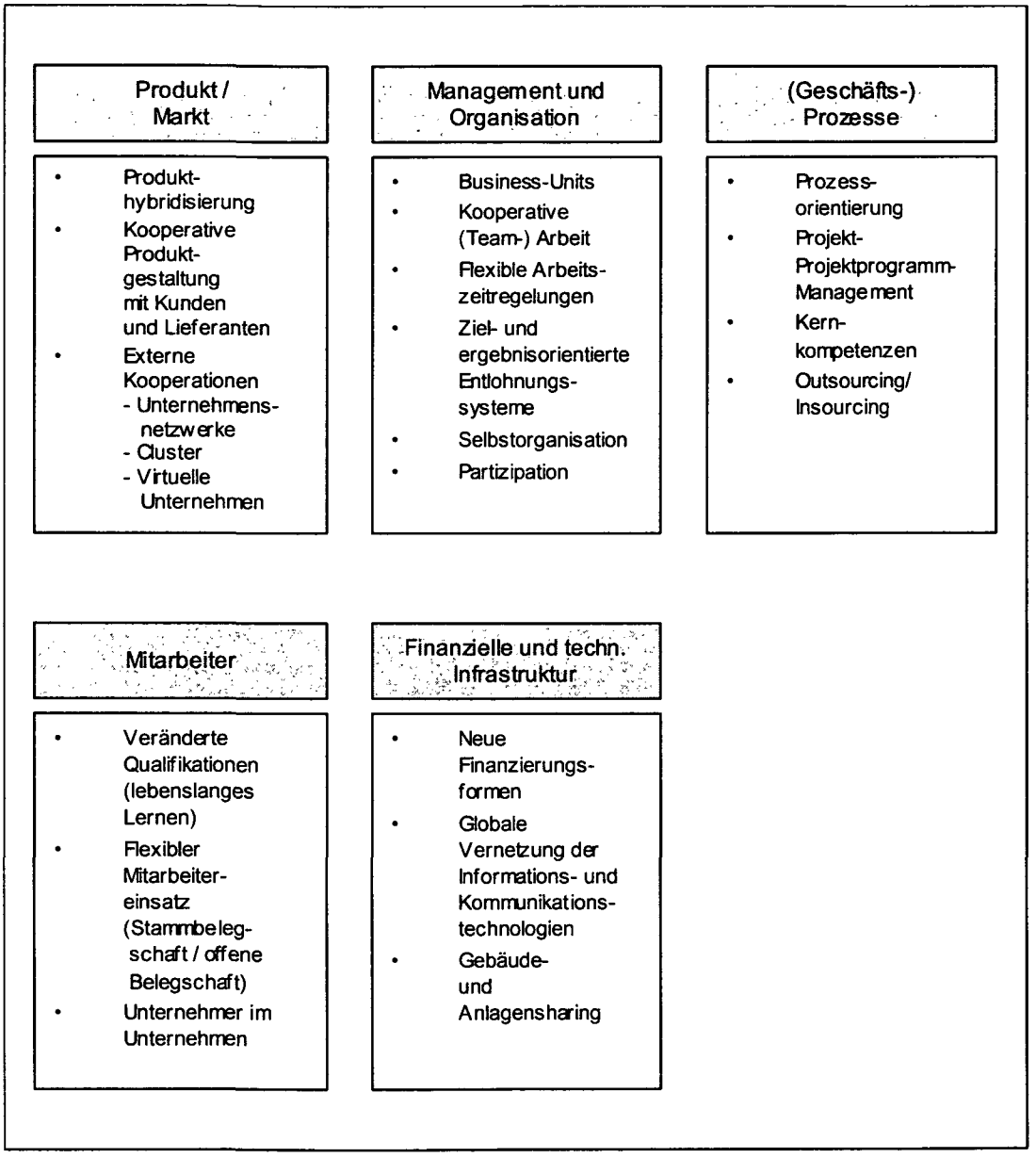
In den folgenden Unterabschnitten werden drei für die Gestaltung eines Softwarehauses geeignete Modelle präsentiert, wobei jenes von Horx lediglich als Modell-Ergänzung zu betrachten ist.

---

[1] Wojda, Franz, Organisation und Führung, TU-Wien, 1998
[2] Wojda, Franz, Buresch, M., Seghezzi (hrsg.), Ganzheitliche Unternehmensführung, Stuttgart, 1997

9.1.1 Modell der 5 Gestaltungsfelder nach Wojda

Tabelle 9.1 [3]) gibt einen Überblick über wichtige Gestaltungsmerkmale und wie sie den Gestaltungsfeldern zuzuordnen sind:



Tab. 9.1: Gestaltungsfelder und –faktoren nach Wojda

Die Gestaltungsfaktoren eines Softwarehauses werden in weiterer Folge anhand dieses Modells präsentiert.

[3] Wojda, Franz (hrsg.), Waldner, B., Innovative Organisationsformen, Stuttgart, 2000

9.1.2 Organisationsgestaltung nach Mintzberg

Ein ebenso inhaltlicher Zugang auf die Organisation ist jener von Mintzberg [4]). Seiner Theorie zufolge wird eine Organisation durch

9 Gestaltungsparameter:

- Aufgaben-Spezialisierung (Positions-Gestaltung)
- Verhaltens-Formalisierung (Positions-Gestaltung)
- Ausbildung und Indoktrinierung (Positions-Gestaltung)
- Gruppierung von Einheiten (Aufbauorganisations-Gestaltung)
- Größe der Einheiten (Aufbauorganisations-Gestaltung)
- Planungs- und Kontrollsysteme (Gestaltung der lateralen Verbindungs-Struktur)
- Kontaktinstrumente (Gestaltung der lateralen Verbindungs-Struktur)
- vertikale Dezentralisierung (Gestaltung der Entscheidungs-Findung)
- horizontale Dezentralisierung (Gestaltung der Entscheidungs-Findung)

5 Koordinationsmechanismen:

- gegenseitige Abstimmung
- persönliche Weisung
- Standardisierung der Arbeitsprozesse
- Standardisierung der Arbeitsprodukte
- Standardisierung der Mitarbeiter-Qualifikationen

und **4 situative Faktoren:**

- Alter und Größe der Organisation
- Technisches System
- Umwelt
- Machtbefugnisse

determiniert. Im weiteren Verlauf dieser Arbeit werden einige dieser Gestaltungsmerkmale wieder aufgegriffen.

[4] Mintzberg, H., Die Mintzberg-Struktur: Organisationen effektiver gestalten, 1992

9.1.3 Organisationsgestaltung nach Horx

Einen in der Software-Branche zu erkennenden Trend hinsichtlich der Gestaltungsmerkmale der Organisation hat Matthias Horx bereits in einer Publikation aus dem Jahre 2000 über das „Unternehmen im Jahre 2010" vorweggenommen [5]). Ihm zufolge sind die zukünftigen Gestaltungsfaktoren:

- „War for Talents":
  Der zukünftige Mitarbeiter wird als Kunde betrachtet und muss stetig umworben werden
- „Life-Balance-Politik":
  Die Ausgeglichenheit und ganzheitliche Gesundheit aller Mitarbeiter wird zum betrieblichen Megatrend
- „Vision Design":
  Visionen sollen durch „Mind Mapping" innerhalb des Unternehmens visualisiert und realisiert werden
- „Informationsökologie":
  Das Wissen wird in Kreisläufen (von der Wissens-Aggregation bis zur „Wissens-Entsorgung") organisiert, um es in optimaler Weise für den Praxiseinsatz vorzubereiten
- „Innovationsorientierung":
  Gestaltung des Wissens-Zustroms von außerhalb der Firma bzw. Organisation
- „Prosuming-Kultur":
  Durch den Einfluss des Konsumenten auf die Produktgestaltung wird er zum Produzenten seiner Nachfrage
- „Workholder und Riskholder":
  Die neue Teilhabepolitik gegenüber den Mitarbeitern verändert die individuelle Verantwortung des einzelnen gegenüber dem Unternehmen
- „Benefiz-Kapitalismus":
  Das neue Gesellschaftsengagement der Unternehmen
- „Die Business-Revolution":
  Übergang vom „gespaltenen" Kapitalismus zum „Win-Win-Modell" für alle Bürger (im volkswirtschaftlichen Sinn)

Wie bereits erwähnt, kann dieses Modell lediglich als Ergänzung zu den beiden vorhergehenden, alles umfassenden, Modellen betrachtet werden. Im nächsten Abschnitt wird kontextbezogen auf den einen oder anderen Faktor aus dieser Liste Bezug genommen.

[5] Horx, Matthias, Die acht Sphären der Zukunft, Wien, 2000

9.2 Angewandtes Modell der 5 Gestaltungsfelder nach Wojda

In den folgenden Abschnitten werden wesentliche Gestaltungsfaktoren eines Softwarehauses – gegliedert nach den 5 Gestaltungsfeldern – dargestellt.

9.2.1 Produkt / Markt

Zu diesem Gestaltungsfeld zählen u.a. Produktstrategien, Preisgestaltung und Absatzpolitik. Wesentliche Faktoren sind:

- Abgrenzung zwischen Eigenentwicklung und Zukäufen (OEM-Produkte). Dadurch wird einerseits eine Fokussierung auf die eigenen Stärken forciert, andererseits kann auch manches Entwicklungs-Risiko auf Zulieferfirmen abgewälzt werden.

- Aufgrund der Eigenschaft, dass die Herstellungskosten nahezu unabhängig von der Stückzahl sind, bedeutet (vereinfacht) eine Verdopplung der Absatzmenge eine Verdopplung des Gewinns. – Andererseits wird durch die Individualisierung der Software im industriellen Sektor eine Kundenbindung erzeugt, die durch Massenware unmöglich zu erreichen ist.

- Kooperationen sind insbesondere aus zwei Gründen wichtig. Sie senken sowohl die Entwicklungskosten als auch die Entwicklungszeiten für Software-Projekte.

Hinsichtlich der Märkte sind auch bei Softwareprodukten – abgesehen von der Sprache – Unterschiede zu berücksichtigen. Diese sind z.b. auf kulturelle Unterschiede, aber auch auf stark variierende Lohnkosten zurückzuführen.

9.2.2 Prozesse

Prozesse, die die Arbeitsabläufe eines Softwarehauses regeln, wurden bereits an anderer Stelle dieser Dissertation ausführlich dargestellt. Sie dienen in der Software-Technologie insbesondere der Koordination unterschiedlicher Tätigkeiten und weniger der Verfahrensbeschreibung (wie z.b. in der Chemie). Während der Software-Entwicklungsprozess wesentlich durch die eingesetzte Software-Technologie bestimmt wird, koordinieren andere Prozesse – wie z.b. der Marketing- oder OEM-Prozess – die Verantwortlichkeiten und Schnittstellen für einen reibungslosen Arbeitsablauf. Dies ist umso wichtiger, je dezentraler die Entwicklung, der Vertrieb und der Service weltweit verteilt sind.

Die von Mintzberg angeführten Koordinationsmechanismen werden alle in der Software-Entwicklung eingesetzt. Ein in großen Softwarehäusern manchmal nachteilig wirkender Mechanismus ist jener der gegenseitigen Abstimmung. Obwohl auf technischer Seite keinesfalls substituierbar, führt er auf Management-Ebene oft zu sogenannten „Dead Locks". Diese sind dann anzutreffen, wenn durch lokale Optimierung und fehlender persönlicher Weisung eine Einigung nur sehr langsam erzielbar ist. Um diese Situation zu vermeiden empfiehlt sich eine hierarchisch hoch genug angesiedelte Stabsstelle, die eine für das Gesamtunternehmen optimierte Entscheidung vorbereitet.

9.2.3 Management und Organisation

Durch die hohe Personalintensität wird der Arbeitsteilung und Kommunikationsgestaltung ein hoher Stellenwert beigemessen. Darüber hinaus sind bei international verteilten Entwicklungsniederlassungen auch steuerliche Optimierungen für die Organisationsgestaltung maßgebend.

Abb. 9.1: Vier-Ebenen-Modell internationaler F+E Organisationen

Basierend auf dem Vier-Ebenen-Modell von Gassmann [6]) werden die eigenen Erfahrungen des Dissertanten in Form von Gestaltungsmerkmalen in Abb. 9.1 der jeweiligen Organisationsebene nach Gassmann zugeordnet. Hierbei wird die Gesamtorganisation als Überlagerung von Organisations-Ausschnitten (jeweils an unterschiedlichen Kriterien bzw. Dimensionen ausgerichtet – z.b. regional und legal) dargestellt.

In den folgenden Abschnitten wird auf die funktionale und die Projektorganisation speziell Bezug genommen. Die regionale Organisation ist im wesentlichen durch die Umwelt der Unternehmung bestimmt, während die informelle Organisation stark durch die Firmenkultur mitbestimmt wird – auf beide wird hier nicht gesondert eingegangen.

9.2.3.1 Funktionale Organisation

Die Aufgaben-Spezialisierung kann generell in vertikaler, horizontaler oder vertikaler & horizontaler Richtung geschehen. Für die Software-Entwicklung eignet sich insbesondere eine horizontale Spezialisierung, d.h. dass unterschiedliche Teile der Software (GUI, Datenbank, usw.) von unterschiedlich spezialisierten Mitarbeitern erstellt werden. Dies vor allem vor dem Hintergrund, dass der Umfang an bereits erfolgreich entwickelten Subsystemen unüberschaubar zunimmt und eine einzelne Person kaum mehr einen Gesamtüberblick hat. In vertikaler Richtung erfolgt eine vergleichsweise nur gering ausgebildete Spezialisierung entsprechend dem Entwicklungsprozess (z.b. Anforderungsdefinition, Entwurf & Codierung, Test).

Eine Verhaltens-Formalisierung ist weniger ausgeprägt, da die Software-Entwicklung in einem hohen Grad von den Fähigkeiten: Phantasie und Kreativität bestimmt wird. Bürokratische Strukturen können sich der notwendigen Flexibilität nicht anpassen. Ausbildung und Indoktrinierung sind ebenfalls wichtige Gestaltungsparameter. Durch betriebliche Schulungen wird ein charakteristisches Vokabular aufgebaut. Da die fachliche Komponente je nach Projektbedürfnissen sehr unterschiedlich sein kann, erfolgt eine Verhaltens-Formalisierung eher durch Schulungen über interne Prozesse (vgl. hierzu die 9 Gestaltungsparameter von Mintzberg).

Die Aufbauorganisation eines Softwarehauses entspricht im Grunde der Struktur von Produktions-Unternehmen; in Softwarehäusern gibt es ebenso Vertriebs-, Personal-, Buchhaltungs-, Gebäudeinstandhaltungs-Abteilungen, usw. - um nur einige Beispiele zu nennen. Jener Teil der Gesamtorganisation, der die Software erstellt, ist primär hierarchisch strukturiert. Innerhalb dieser Struktur wird nach Kundensegmenten bzw. in geringerem Ausmaß auch nach Technologien untergliedert. Begleitende Aktivitäten (z.b. Qualitätssicherung, Aufwandsschätzung, usw.) werden von Stabsstellen wahrgenommen. Da jedes Projekt

[6] Gassmann, Oliver, Internationales F+E Management, München, 1997

unterschiedliche Anforderungen an das Personal stellt, muss diesem durch eine entsprechende Projektorganisation Rechnung getragen werden. In diesem Zusammenhang spricht man auch von einer Matrixorganisation (Projekt- versus Linienorganisation).

Die Linienorganisation übernimmt Aufgaben der persönlichen Mitarbeiterbetreuung – unabhängig von der Projektsituation, wie z.b. Karriereplanung, allgemeine Weiterbildung, usw. Da der Projektleiter kaum mit disziplinarischen Vollmachten ausgestattet ist, führt er mit Autoritätsbewusstsein, Überzeugungskraft, Verhandlungsgeschick und Einfühlungsvermögen.

Die Gruppierung von Einheiten kann grundsätzlich nach

- Objekten (Ort, Kunde, Produkt, u.a.) oder
- Verrichtungen (Funktion, Qualifikation, Arbeitsprozess, u.a.)

erfolgen.

In großen Softwarehäusern wird in der Praxis in produktschaffende und kundenorientierte (vertriebliche) Organisationseinheiten gruppiert. Da die von beiden Organisations-Einheiten geforderten Qualifikationen und Arbeitsprozesse sehr unterschiedlich sind, kommt es an der Schnittstelle kaum zu Synergien. Die Schwierigkeit in der Lösung dieses Bruchs im Arbeitsprozess ist vor allem in der stark unterschiedlichen Mentalität der Personen in diesen Organisationseinheiten begründet (Entwicklung versus Vertrieb). Im Gegensatz dazu sind hierin jedoch die Stärken von kleinen Softwarehäusern begründet.

9.2.3.2 Projektorganisation

Für die Realisation eines Software-Projekts ist die Gestaltung der Projektorganisation von größter Bedeutung. „Unter Projektorganisation sind jene Regeln, Werte und Normen zu verstehen, die dazu nötig sind, die Zusammenarbeit aller am Projekt Beteiligten möglichst effizient zu gestalten. Darüber hinaus wird mit einer Projektorganisation durch die Festlegung von Verantwortung und Kompetenzen auch die Eingliederung des Projekts in die bestehende Unternehmensorganisation vereinbart.“ [7])

Zu den wichtigsten Rollen in einer Projektorganisation zählen:

- Projektsponsor
  Im Laufe eines Software-Projekts ergeben sich oft Situationen bzw. Konflikte, die ohne das Eingreifen einer hierarchisch höher (als der Projektleiter) gestellten Person nicht lösbar sind. Dies kann z.b. dann der Fall sein, wenn mehrere Projekte auf die gleichen Ressourcen zugreifen.

[7] Patzak, Gerold, Rattay, Günter, Projekt Management, Wien, 1998

- Projektleiter
  Zu den zentralen Aufgaben des Projektleiters zählen das Umfeldmanagement, die Führung des Projektteams und die inhaltliche Realisierung des Projekts, deren Grundlage in erster Linie die Projektdefinition und das Risiko-Management sein sollten.

- Projektcontroller
  Im Gegensatz zum Projektleiter nimmt der Controller eine eher passive bzw. reflektive Rolle ein. Er unterstützt den Leiter in methodischen Angelegenheiten und pflegt das Berichts- und Dokumentationswesen.

- Projektteam
  Aufgrund der häufig stattfindenden Abstimmungsrunden in Software-Projekten ist ein Mindestmaß an Zusammengehörigkeitsgefühl innerhalb des Teams für den Erfolg unerlässlich [8]). Insbesondere in Matrixstrukturen (Projektstruktur innerhalb einer Linienstruktur) können Konflikte zwischen Teammitgliedern rasch zur (Teil-)Auflösung von Teams führen.

- Projektteam-Coach
  Diese Rolle hat als einzige Aufgabe die Führung eines Teams. Dabei kann diese Rolle je nach Projektgröße vom Projektleiter selbst oder von Teamleitern wahrgenommen werden. Ihr Zweck wird im Buch von Ethan Rasiel [9]) in sehr eindrucksvoller Form beschrieben:

> *„Take your team's temperature. Talk to your teammates. Make sure they are happy with what they are doing. Find out if they have questions about what they are doing or why they are doing it, and answer them. If they are unhappy, take remedial action quickly. ...*
>
> *Let your teammates know why they are doing what they're doing. People want to feel that what they are doing is adding value to the client. There are few things more demoralizing than doing something that you and your team leader know is valueless. No one on your team should ever feel, I've just spent two weeks of my life for nothing. ...*
>
> *Get to know your teammates as people. Are they married? Do they have kids? What are their hobbies? It will help you to understand them. Share a bit about yourself as well; that makes it more likely that your teammates will think of you as part of „us", rather than „them". This incidentally, is a much better way of team bonding than taking your team out to the ball game."*

[8] Kellner, Hedwig, Projekt-Mitarbeiter finden und führen, München, 2003
[9] Rasiel, Ethan M., The McKinsey Way – Using the Techniques of the World's Top Strategic Consultants to Help You and Your Business, USA, 1999

Eingeschränkt werden generelle Aussagen zur Projektorganisationsgestaltung speziell im Bereich der Software-Entwicklung durch den Umstand, dass sogenannte „Software-Projekte" nicht immer alle typischen Merkmale eines klassischen Projekts aufweisen. Hinzu zählen:

- Software-Projekte sind nicht immer einzigartig. Hinsichtlich der Technologie und der Zielgruppe bestehen oft Übereinstimmungen über mehrere Projekte hinweg, vor allem dann, wenn das Produkt an mehrere Kunden vertrieben wird.

- Der rudimentäre Software-Entwicklungsprozess ist weitgehend standardisiert.

9.2.3.2.1 Teamgröße

Die Teamgröße wird durch zwei gegensätzlich wirkende Mechanismen bestimmt. Grundsätzlich kann durch eine stärkere Arbeitsteilung – d.h. mehr Mitarbeiter – der gesamte Arbeitsaufwand in kürzerer Zeit erledigt werden (unter der Annahme, dass die Arbeit grundsätzlich teilbar ist). Als Formel ausgedrückt ist demnach der Zeitbedarf „t" indirekt proportional zur Anzahl der Mitarbeiter „n" ($t \sim 1/n$). Dem steht entgegen, dass bei starken Interdependenzen zwischen den Arbeitspaketen, und damit den Personen, der Kommunikationsaufwand mit zunehmender Größe des Teams wächst. Als Formel ausgedrückt ist der Kommunikationsaufwand, und damit der Zeitbedarf „t", proportional der Anzahl der Kommunikationspfade; bei „n" Mitarbeitern gibt es $n*(n-1)/2$ Kommunikationspfade. Kombiniert kommt man somit zum Resultat (vgl. Abb. 9.2):
-> $t \sim 1/n + k*n*(n-1)/2$

In der Software-Entwicklung ist demnach eines der wichtigsten Kriterien für die Arbeitsteilung die Minimierung der notwendigen Kommunikation zwischen den Personen. In der Praxis können 3 bis 7 Personen optimal an einem Thema – wie auch immer abgegrenzt – arbeiten. Damit benötigen größere Projekte eine zusätzliche Koordination der einzelnen Teams. In Kombination mit der Lernkurve ist somit auch das „Brooks'sche Gesetz" offensichtlich [10]): „Adding manpower to a late software project makes it later".

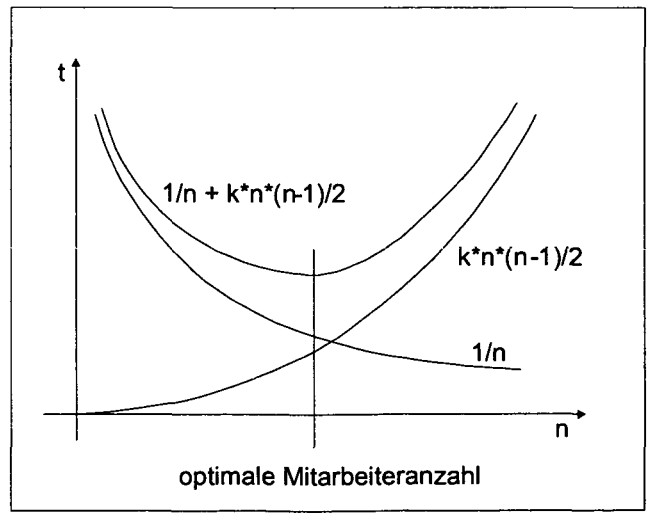


Abb. 9.2: Teamgröße

[10] Brooks, F.P., The mythical Man-Month, 1975

9.2.3.3 Personalallokation

Spätestens seit dem Einbruch der „New Economy" im Winter des Jahres 2000 hat sich ein neuer Trend der organisatorischen Rahmenbedingungen entwickelt. Während die Software-Entwicklungsarbeit zuvor im wesentlichen durch Angestellte des eigenen Hauses („Stammbelegschaft") und Angestellte anderer Firmen (outgesourcte Themen) geleistet wurde, kristallisiert sich eine immer stärkere Schicht von „Flexworkern", „Ich-AGs" und „Freien Agenten" - auch als „offene Belegschaft" bezeichnet - heraus. Der größte Teil von ihnen wird derzeit noch von Leihfirmen verwaltet, was aber in Zukunft z.b. auch von Universitäten oder übers Internet koordiniert werden könnte.

Der Grund dafür ist, dass die Personalverantwortung seitens des Arbeitgebers – vor allem hinsichtlich der Einhaltung arbeitsrechtlicher und gewerkschaftlicher Randbedingungen (Kollektivverträge, Firmenvereinbarungen, u.a.) – aufgrund der Firmenzugehörigkeit unterschiedlich gestaltet werden kann:

1. Angestellte der Stammfirma genießen zumeist den größten arbeitsrechtlichen Schutz. Sie bilden die Kernmannschaft des Softwarehauses.
2. Angestellte von Tochterfirmen unterstützen die Kernmannschaft z.b. wegen der lokalen Präsenz zum Zielmarkt oder wegen geringerer Lohnkosten in Billiglohnländern.
3. Angestellte von Personalleasing-Firmen gleichen längerfristigen Personalmangel aus.
4. Für zahlenmäßig geringe Kapazitätsanpassungen sind freie Mitarbeiter eine kostengünstigere Variante als Mitarbeiter von Personalleasing-Firmen.
5. Die größte Flexibilität wird allerdings durch den Zukauf von Teilprodukten erreicht, da in diesem Fall keinerlei Verpflichtungen hinsichtlich des beim OEM-Lieferanten eingesetzten Personals übernommen wird (die Personalorganisation ist somit im Produktpreis „inkludiert"). Zudem erlaubt diese Variante eine flexible Produktgestaltung (vgl. hierzu Kapitel 5).

9.2.4 Mitarbeiter

Die Mitarbeiter stellen den wichtigsten Faktor der Wertschöpfung in einem Softwarehaus dar. Einige bis jetzt noch nicht dargestellte Eigenschaften dieses Gestaltungsfeldes werden in den nächsten Abschnitten beschrieben. Darüber hinaus wird kontextabhängig auch in anderen Kapiteln auf Personalthemen Bezug genommen (vgl. z.b. Kapitel 11).

9.2.4.1 Personaleinsatz-Qualifikation

Aufgrund der Tatsache, dass die Produktivität der Software-Entwickler um eine Größenordnung variieren kann, hat die ihr zugrundeliegende Personal-Qualifikation einen großen Einfluss auf die Produktivität der gesamten Organisation. Die Kehrseite dieser Medaille sind die hohen Personalkosten hochqualifizierter Fachkräfte, die bei stark fluktuierender Auslastung im Projektgeschäft ein hohes finanzielles Risiko für das Softwarehaus darstellen.

In vielen Publikationen zum Thema: „Ursachen für fehlgeschlagene Software-Projekte" werden in erster Linie personenbezogene Gründe angegeben. Zudem wird dies der fast ausschließlich technikfokussierten Ausbildung zugeschrieben [11]). Notwendige ergänzende Fähigkeiten sind daher [12]):

- gute kommunikative Fähigkeiten, um komplexe Sachverhalte darstellen zu können – hierzu zählen auch gute Englisch-Kenntnisse, da die meisten Publikationen, aber auch interne Dokumente, in Englisch verfasst sind
- der Wille zum lebenslangen Lernen
- die Bereitschaft zum häufigen Reisen, Präsentieren und Pflegen von Kundenkontakten
- Teamfähigkeit
- Einfühlungsvermögen (z.b. Verständnis für die Bedürfnisse der Mitarbeiter haben)
- Durchsetzungsfähigkeit
- Networking Skills (z.b. Partnerschaften und Allianzen zum gegenseitigen Wissens- und Erfahrungsaustausch bzw. Nutzen organisieren)
- Initiative (z.b. durch proaktives Vorgehen)
- Veränderungsfähigkeit
- Analysefähigkeit
- Planungs- und Organisationsgeschick
- Entscheidungsfähigkeit
- Kreativität
- Strategisches Denken
- Kundenorientierung
- Motivationsfähigkeit
- Coaching und Mentoring (z.b. durch konstruktives Feedback)
- Ergebnisorientierung

[11] DeMarco, Tom, Lister, Timothy, Wien wartet auf Dich! Der Faktor Mensch im DV-Management, München, 1999

[12] teilweise basierend auf persönlichen Gesprächen mit Vertretern der Personalabteilung der Siemens AG Österreich

9.2.4.2 Spezialisierung

Grundsätzlich lassen sich die Aufgaben in einem Softwarehaus anhand von Rollen beschreiben. Im Entwicklungsbereich existiert folgende klassische Rollenverteilung (mit Angabe der wichtigsten Tätigkeiten):

- Projekt-Akquisiteur
  - Identifikation potentieller Projekte
  - Präsentation des Know-hows der jeweiligen Entwicklungsabteilung
  - Gestaltung der Personalauslastung
- Projektleiter bzw. Projektmanager
  - Kommunikation mit dem Auftraggeber
  - Kooperation mit der Qualitätssicherung und dem Konfigurations-Management
  - Aufstellen der Kosten-Nutzen-Analyse
  - Planung, Steuerung und Kontrolle des Projekts
  - Erstellung des Projekthandbuchs (Aufwände, Termine, Personal und Betriebsmittel)
  - Ausschreibungsdurchführung und Vertragsüberwachung
  - Beauftragung der Projektmitarbeiter
  - Berichtsmanagement
  - Konfliktmanagement
  - Risikomanagement
- Teilprojektleiter
  - Übernahme von Aufgaben des Projektleiters
- Qualitätssicherer
  - Auswahl und Implementierung von Qualitätssicherungs-Prozessen
  - Definition von Metriken
  - Durchführung der Qualitätskontrolle
- Systemanalytiker
  - Erstellung des Pflichtenhefts
  - Durchführung der Software-Systemanalyse
- Software-Architekt (oft mit der Rolle des Systemanalytikers gekoppelt)
  - Entwurf des Software-Systems
- Programmierer
  - Codierung des Entwurfs
- Konfigurations-Manager
  - Verwaltung der Software-Versionen
- Anwendungsspezialist
  - Einbringung des Know-hows über das Fachgebiet für die die Software erstellt wird

Dieses Rollenverständnis findet sich in einem erweiterten Sinne in der Abhandlung über Patterns wider. Eine andere Variante der Aufgabengliederung ist die horizontale bzw. vertikale Spezialisierung.

Bei der horizontalen Spezialisierung werden die einzelnen Stufen des Entwicklungs-Prozesses von unterschiedlichen Personen durchgeführt. Dies führt zur Erhöhung der Qualität und Effizienz innerhalb der jeweiligen Stufe, birgt andererseits aber die Gefahr der Inkompatibilität mit den Aktivitäten der voran- oder nachgelagerten Arbeiten. Im Falle der vertikalen Spezialisierung wird der Entwicklungs-Prozess eines Moduls von derselben Person wahrgenommen. Dies fordert ein sehr breit gefächertes Wissen über die auf den einzelnen Stufen angewandten Techniken und erschwert die Wiederverwendung, da ähnliche Tätigkeiten nur in großem Zeitabstand wiederkehren.

9.2.4.3 Personalentwicklung

Es sind Persönlichkeiten, nicht Prinzipien, die etwas bewegen.
Oscar Wilde

Aufgrund der hohen Ausbildungsanforderungen werden für die Software-Entwicklung vorzugsweise Absolventen von Höheren Technischen Lehranstalten, Fachhochschulen und Universitäten eingesetzt. Das Risiko der Fehleinstellungen wird durch einen möglichst frühzeitigen Kontakt zu potentiellen zukünftigen Mitarbeitern reduziert. Dieser Weg lässt sich insbesondere in der Software-Entwicklung relativ einfach verwirklichen, da der praktische Ausbildungsanteil schon gut auf die Anforderungen in der Branche abgestimmt ist. Ein besonderer Vorteil erwächst für beide Seiten aus längerfristigen Kontakten – z.b. durch kontinuierliche geringfügige Beschäftigung während des Studiums. Defizite gegenüber den durchschnittlichen Anforderungen sind eher in den Softskills erkennbar. Hierzu zählt die Teamintegration oder die Durchsetzungsfähigkeit in Diskussionen (vgl. hierzu die Zusatzanforderungen aus der Darstellung im vorangegangenen Abschnitt).

Aufgrund des zu anderen Branchen vergleichsweise raschen Fortschritts ist eine kontinuierliche Weiterbildung Voraussetzung für einen Software-Entwickler. Diese sollte in einem Ausmaß von zumindest 3 Wochen pro Jahr stattfinden. Zudem muss durch den Einsatz neuer, effizienterer Methoden auch das hohe Kostenniveau in Westeuropa gerechtfertigt werden. So liegen z.b. die Personalkosten in Indien oder China um etwa einen Faktor 10 unter jenen in Deutschland. Eine teilweise Kompensation der geringen Personalkosten ist auf folgende Nachteile im Falle der Softwareproduktion in Billiglohnländern zurückzuführen:

- (durchschnittlich betrachtet) geringeres einschlägiges Know-how
- geographische Entfernungen führen zu hohen Reiseaufwänden
- größere Unterschiede bezüglich der Zeitzonen schränken die Kommunikationszeiten ein
- Kommunikationsverbindungen zu Billiglohnländern weisen meist eine niedrigere Qualität auf
- eine einheitliche Kommunikation in Englisch erschwert manchen Entwicklern die Darstellung komplizierter Sachverhalte
- höhere Personalfluktuation (z.b. in Indien, wo der Job eines Software-Entwicklers als „Karrieresprungbrett" oder „Weg in die USA" gesehen wird)
- Verfolgung lokaler Eigeninteressen
- u.a.

Ein weiteres kritisches Thema ist die Karriereplanung in Entwicklungs-Abteilungen. Einerseits zeigen die stark variierenden Leistungen hinsichtlich der Software-Entwicklung potentielle Kandidaten für Beförderungen auf, andererseits sind diese Kandidaten oft ungeeignet Teams zu führen. Hieraus ergibt sich die Notwendigkeit sowohl einer Führungslaufbahn wie auch einer Fachlaufbahn. Innerhalb der Fachlaufbahn werden Mitarbeiter ermutigt sich zu Experten zu entwickeln (mit entsprechenden Gehaltssteigerungen). Dies erfordert vor allem eine Abkehr von der weit verbreiteten Kopplung zwischen Gehalt und Anzahl der Untergebenen.

9.2.5 Unternehmensinfrastruktur

Diese wird im Modell der 5 Gestaltungsfelder grundsätzlich in die technische und die finanzielle Infrastruktur untergliedert.

9.2.5.1 Technische Infrastruktur

Hinsichtlich der technischen Infrastruktur wird der Software-Entwicklungsprozess, abgesehen vom Entwicklungswerkzeug (Computer), von der geographischen Allokation der Entwicklungs-Stätten stark beeinflusst. Damit sind neben den technischen Leistungsdaten der Computer auch die Leistungsdaten der sie verbindenden Netze Teil der technischen Infrastruktur. Dies umso mehr, je verstreuter die Entwicklungsstandorte in der Welt sind. So ist z.b. insbesondere bei „Billig-Entwicklungsstandorten" wie Indien oder China die Qualität der Datenübertragung zu den Auftraggeberstandorten ein entscheidendes Leistungs- und damit auch Effizienz-Kriterium.

Ein anderer in der Softwarebranche an Bedeutung gewinnender Gestaltungsfaktor ist das Tele- oder Homeworking. Dadurch erspart man sich einen Teil der Büroflächen. Allerdings sind diese

insbesondere in der Software-Branche nicht beliebig verkleinerbar, da damit nachweislich die Produktivität der Softwareentwicklung negativ beeinflusst wird [13]).

9.2.5.2 Finanzielle Infrastruktur

Grundsätzlich ist die Softwarebranche durch 2 Charakteristika geprägt:

1. der Großteil der Kosten (zwischen 80 und 90%) wird durch Personalkosten verursacht
2. analog zu anderen Forschungs- und Entwicklungsorganisationen ist die Zeitspanne zwischen Investition und Kapitalrückfluss relativ lang (bis zu etwa 5 Jahren!)

Während bei etablierten Unternehmungen mit Produkten in unterschiedlichen Reifestufen neue Entwicklungen durch „Cash Cows" (siehe BCG-Portfolio [14])) finanziert werden, fehlt neugegründeten Softwarehäusern oft ausreichendes Eigenkapital, welches zudem in kürzester Zeit verfügbar sein muss. Diesem Umstand Rechnung tragend existieren insbesondere im angelsächsischen Raum (USA, GB) unterschiedliche Finanzierungsvarianten – u.a. [15]):

- Institutionelle **Venture-Capital Gesellschaften** sind Risikokapitalgeber mit dem Ziel eines Investments in rasch wachsende Unternehmungen – meist ohne persönlichen Bezug zu den Unternehmensgründern. Zwischen Investment und Deinvestment (durchschnittlich zwischen 3 und 7 Jahren) wird das Investmentrisiko durch überdurchschnittlich gute Renditen (Ziel: > 100% / Jahr) abgegolten. Die Kapitalausstattung der Venture-Capital Gesellschaft erfolgt über einen Fonds, an dem z.b. Banken, Versicherungen, Unternehmen u.a. beteiligt sind (vgl. Abb. 9.3). Der Anteilserwerb an dem Beteiligungsunternehmen wirkt sich meist durch eine Kapitalerhöhung aus.

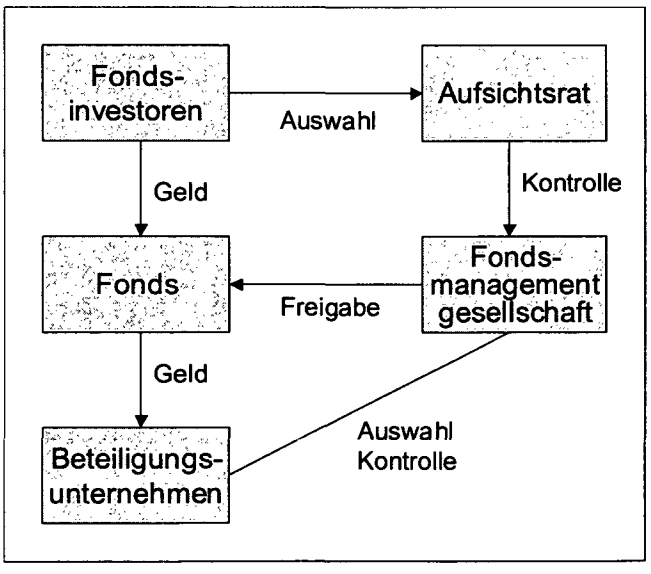


Abb. 9.3: Fonds-Struktur

[13] DeMarco, Tom, Lister, Timothy, Wien wartet auf Dich! Der Faktor Mensch im DV-Management, München, 1999

[14] Henderson, Bruce D., Oetinger, Bolko von, (hrsg.), Das Boston Consulting Group Strategie-Buch, Düsseldorf, 2000

[15] Stadler, Wilfried (hrsg.), Beteiligungsfinanzierung, Wien, 1997

- **Business Angels** sind informelle Eigenkapitalgeber – zumeist private Personen: Verwandte, Freunde, (Ex)Unternehmer, (Ex)Manager, u.a. Neben dem Kapital stellen sie auch Know-how und Erfahrung bereit. Gegenüber Venture-Capital Gesellschaften investieren sie in einer sehr frühen Lebensphase des Unternehmens, investieren eher kleinere Beträge, sind risikofreudiger mit langfristigen Kapitalengagements und investieren eher in deren geographisch nähere Umgebung. Zudem genügen privaten Risikokapitalgebern wesentlich einfachere Strukturen als z.b. institutionellen Venture-Capital Gesellschaften, wo die Funktion des Business Angels auf Investoren, einen Aufsichtsrat und Investmentmanager verteilt wird.

- **Seed-Financing** *ist eine Sonderform der finanziellen Unterstützung in der Anfangsphase* einer Unternehmensgründung. Es handelt sich dabei grundsätzlich um einen staatlichen Zuschuss, der im Falle des Unternehmenserfolges mit Zinsen zurückbezahlt werden muss.

- **IPO (Initial Public Offer)** – der Börsegang an eine (Technologie-)Börse, wie z.b. die NASDAQ ist nach erfolgreicher Firmengründung ein Weg zur Finanzierung der Unternehmensexpansion über die Ausgabe von Aktien.

Abgesehen vom Hochtechnologie-Charakter, der durch große Chancen und Risken gekennzeichnet ist, unterliegt die Finanzstruktur eines Softwarehauses den allgemeinen Gegebenheiten und wird in dieser Arbeit nicht gesondert dargestellt.

Im Falle von bereits etablierten Software-Unternehmen steht mit steigender Tendenz das Thema der Mitarbeiterbeteiligung bzw. allgemein die erfolgsabhängige Entlohnung zur Diskussion. Dieser Trend kann auch unter dem Begriff: „Unternehmer im Unternehmen" oder „Angestellten-Entrepreneur" subsummiert werden. Auf diese Weise wird der Einfluss bzw. die Abhängigkeit der Kapitalgeber auf das Unternehmen verringert, wobei gleichzeitig Erfolg bzw. Misserfolg auf die Mitarbeiter übertragen wird (vgl. „Workholder und Riskholder" bei Horx [16])).

Abschließend sei in Hinblick auf die finanzielle Infrastruktur auch auf die steuerlichen und exportkontrollrechtlichen Gestaltungsmöglichkeiten bei international operierenden Firmen hingewiesen (siehe hierzu die unterste Ebene in Abb. 9.1). So kann z.b. durch die gezielte Produkt-Zusammenstellung das potentielle Absatzgebiet aufgrund der Vermeidung exportkontrollrechtlicher Einschränkungen erweitert werden.

[16] Horx, Matthias, Die acht Sphären der Zukunft, Wien, 2000

9.3 Change-Management

Aufgrund der kurzen Technologiezyklen und des projekthaften Charakters der Softwarebranche sind Software-Entwicklungs-Organisationen in regelmäßigen Abständen von Umstrukturierungen betroffen. Die dazu notwendigen Veränderungen in der Hierarchie, den Funktionen und Prozessen stehen in engem Zusammenhang mit den Schicksalen der vom Veränderungsprozess betroffenen Mitarbeiter. Sehr oft sind damit begründete, aber auch unbegründete, negative Reaktionen verbunden.

Eine Methode diesen Reaktionen (auf Veränderungen) in geeigneter Weise zu begegnen beschreibt Lewis Gray in einem Artikel für Crosstalk [17]). Seine Arbeit basiert auf Veröffentlichungen von Watts S. Humphrey [18]) und [19]). Im Mittelpunkt der Abhandlung steht eine Arbeitsteilung, die die für den Änderungsprozess notwendige Aktionen auf 4 Rollenbilder verteilt. Sie werden als "Sponsor", "Champion", "Change Agent" und "Coach" bezeichnet. Bevor nähere Betrachtungen der 4 Rollenbilder angestellt werden, soll eine analoge Konstellation aus Karl Popper's berühmtem Werk: „Die offene Gesellschaft und ihre Feinde" zitiert werden [20]):

"Marx hatte also vollkommen recht, als er betonte, dass sich die Geschichte nicht auf dem Papier planen lässt. Aber Institutionen lassen sich planen, und sie werden fortwährend geplant. Nur indem wir, Schritt für Schritt, Institutionen zur Sicherung der Freiheit und insbesondere der Freiheit vor Ausbeutung planen, nur so können wir hoffen, eine bessere Welt zustande zu bringen."

Für den Change-Management-Prozess bedeutet dies analog, dass zuerst die Infrastruktur (d.h. Organisation) und anschließend die Abläufe zu planen sind. In Abb. 9.4 wird der Organisationsaufbau (mit Angabe von Bezeichnungen, wie sie jenen in Mannschaftssportarten entsprechen) dargestellt. Basierend auf einer Übereinkunft zwischen Sponsor und Champion wird von intern oder extern ein Coach engagiert, der zusammen mit dem Champion mehrere Change Agents von innerhalb der von der Umstrukturierung betroffenen Organisation auswählt.

Näher eingegangen wird auf dieses Thema auch im P-CMM (People Capability Maturity Model) des SEI (vgl. hierzu Kapitel 8).

---

[17] Gray, Lewis, in: Crosstalk The Journal of Defense Software Engineering, Sept. 1998
[18] Humphrey, Watts S., Managing the Software Process, Addison-Wesley, Reading, Massachusetts, 1989
[19] Humphrey, Watts S., A Discipline for Software Engineering, Addison-Wesley, Reading, Massachusetts, 1995
[20] Popper, Sir Karl R., Die offene Gesellschaft und ihre Feinde, Neuseeland, 1944

9.3.1 Die 4 Rollenbilder

Sponsor: Der Sponsor ist jene Person, welche uneingeschränkt über die notwendigen Ressourcen verfügt. Damit trägt er die finanzielle Verantwortung hinsichtlich der Realisierung der Umstrukturierung. Seine einzige Aufgabe besteht in der rechtzeitigen Zurverfügungstellung der vom *Champion* benötigten Ressourcen während der gesamten Dauer der Umstrukturierung.

Champion: Der Champion ist mit einem Manager vergleichbar, der die Gesamtverantwortung für die Durchführung des Umstrukturierungs-Programms trägt. Er initiiert zusammen mit dem Sponsor das Programm, führt es während der gesamten Dauer der Umstrukturierung, und verteidigt die daran beteiligten Personen gegen Störungen von außen (z.b. Abwerbung von am Änderungsprozess betroffenen Mitarbeitern). Er engagiert den *Coach* und ernennt gemeinsam mit ihm die *Change Agents* von innerhalb der von der Umstrukturierung betroffenen Organisation. Zusammen mit dem *Coach* stellt er auch den Aktionsplan für die im Detail notwendigen Änderungen auf. Zudem vertritt er den Umstrukturierungs-Prozess nach außen (z.b. im Berichtswesen oder bei Anfragen und Einwänden von außen). Ebenso wie die Rolle des *Sponsors* verlangt auch die Rolle des Champions keine exklusive Widmung gegenüber dem Umstrukturierungs-Prozess.

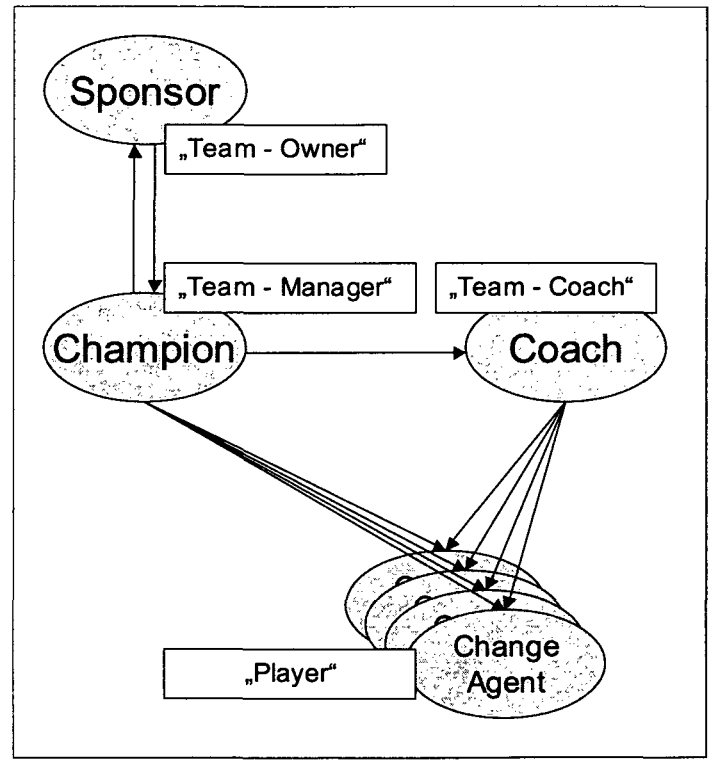


Abb. 9.4: Infrastruktur-Initiierung

Change Agents: Einerseits sind sie nach wie vor in ihren alten Strukturen (Teams) verankert, wodurch sie die zu verändernden Prozesse kennen, andererseits sind sie als Change Agents Teil des Umstrukturierungs-Teams und damit für die Einschulung ihrer Kollegen in die neuen Prozesse verantwortlich. Inhaltlich basiert ihre Tätigkeit auf dem Aktionsplan (vom *Champion* und *Coach* gemeinsam ausgearbeitet), welchen sie im Detail umsetzen. Ihr erster Ansprechpartner im Falle von Problemen ist der *Coach*. Auseinandersetzungen sind deshalb nie auszuschließen, da sie nun gegenüber ihren Kollegen Weisungen geben.

Nach Umsetzung des Aktionsplans überprüfen sie die Auswirkungen auf die umstrukturierten Prozesse, um die zu Beginn formulierten Ziele des Umstrukturierungs-Prozesses auf ihren Erreichungsgrad hin zu bewerten.

Coach: Als Projektleiter des Umstrukturierungs-Prozesses ist er auch mit den Details der Umstrukturierung bestens vertraut. Es liegt in seiner Verantwortung die Strategie und die notwendigen Aktionen zu planen und die Arbeit der *Change Agents* nötigenfalls zu korrigieren. Des weiteren ist er Ratgeber und Informant für den *Sponsor* und den *Champion*. Für das Gelingen eines Restrukturierungsvorhabens ist die Wahl des richtigen Coaches am einflussreichsten. Für seine Besetzung eignet sich auch eine außenstehende Person, die mit ähnlichen Vorhaben bereits Erfahrung gesammelt hat. Lewis Gray's Erfahrung zufolge [21]) zeichnen folgende Eigenschaften erfolgreiche Coaches aus:

- "Knowledge of the game and its strategies. ... Most good sports coaches also were once good players." Damit sind erfahrene (Ex-)Software-Entwickler angesprochen, die in die Prozesse eingebunden waren.
- "A sense of humor. ... Excellence is not based on drudgery; it is based on fun."
- "Honesty and straightforwardness"
- "Inspire trust. ... Change agents cannot feel they must constantly second-guess the coach. Trust is built on honesty and success."
- "Good communication"
- "Respect for the team. A good SPI (= Software Process Improvement) coach listens to team members when they raise an objection."
- "No grudges. ... The coach must find a way to avoid or defuse antagonistic situations."
- "Negative reactions seen in the proper perspective."
- "Focus on what can be done. ... Some practices, such as requirements management, software project planning, and software configuration management, can be much easier to perform with decent software tools than with pencil and paper. Nevertheless, pencil and paper are much underrated as tools. ..."

In Abb. 9.5 werden in übersichtlicher Form die 4 Rollenbilder mit einer kurzen Zusammenfassung ihrer Aufgaben dargestellt.

[21] Gray, Lewis, in: Crosstalk The Journal of Defense Software Engineering, Sept. 1998

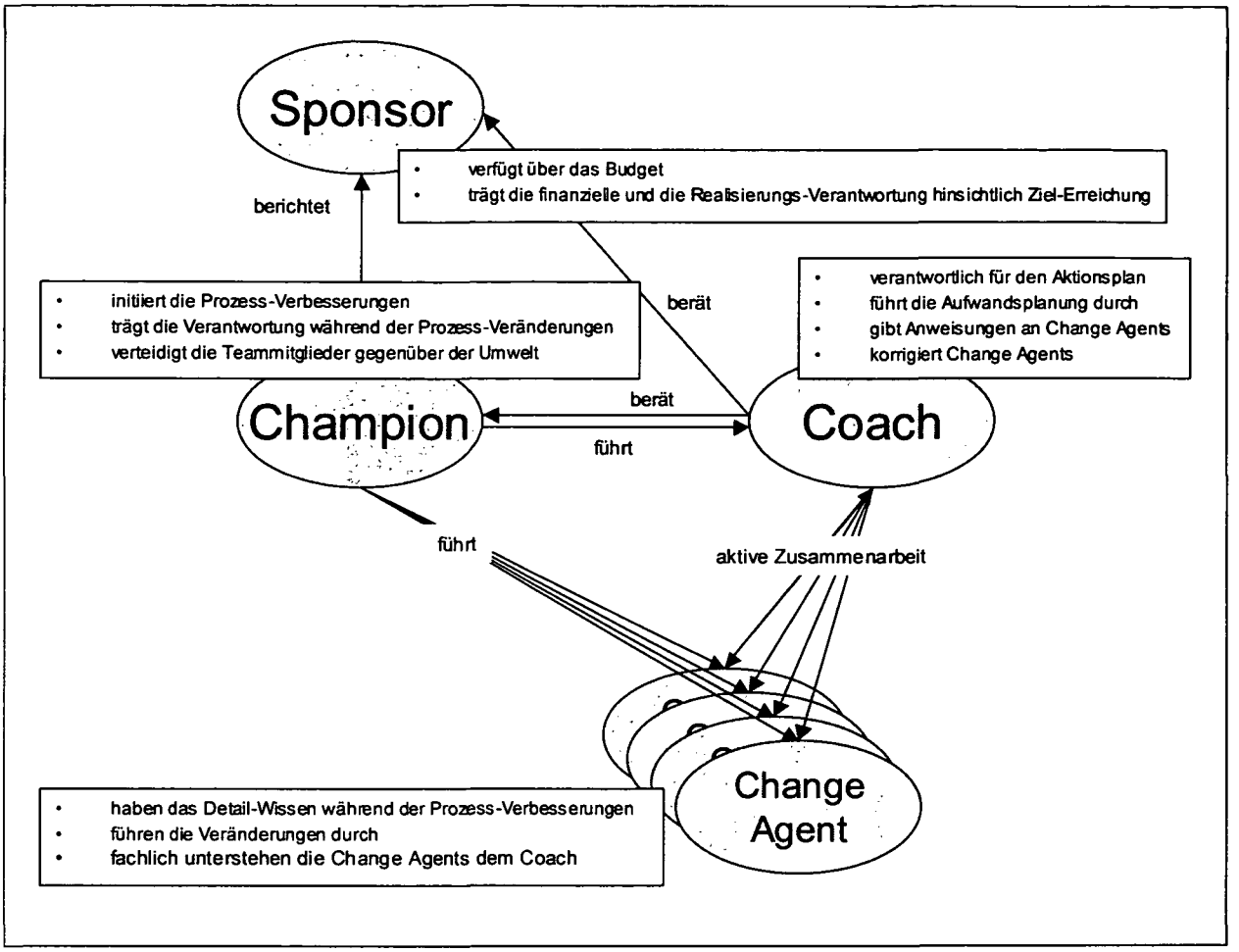


Abb. 9.5: Zusammenspiel der 4 Rollen

9.4 Stakeholder-Konzept

Stakeholder – ein Begriff aus dem Englischen – bezeichnet [22]): „Individuals and organizations who are actively involved in the project, or whose interests may be positively or negatively affected as a result of project execution or successful project completion."

In diesem Konzept werden alle Personen und Organisationen hinsichtlich der Interessen am Software-Projekt berücksichtigt und nach Möglichkeit eingebunden. Abgesehen von technischen Anforderungen, die sich im Projektverlauf ändern, gibt es auch nicht-technische Einflüsse, die gewöhnlich statischer Natur sind (z.b. das Interesse des Betriebsrats, politische Einflüsse, usw.). Aufgabe des Projektleiters ist es, die Interessen aller Stakeholder während der gesamten Laufzeit des Projekts kontinuierlich zu verfolgen.

[22] Cleland, David I. (hrsg.), Field Guide to Project Management, USA, 1998

Dies umfasst:

- die Ermittlung aller Projekt-Stakeholder
- die Abschätzung der Erwartung und des Einflusses eines jeden Stakeholders auf das Projekt
- das Entwerfen von Szenarien, die zu einer Änderung der Erwartung und/oder des Einflusses führen
- die Identifikation der Beziehungen zwischen den „Projekt-Freunden" und „Projekt-Feinden"
- die Planung des Projekt-Informationsflusses
- die Integration der aus den obigen Punkten gewonnenen Informationen in den Risiko-Management-Prozess

Larry Smith formuliert die Anforderungen wie folgt [23]): „On significant projects this endeavor requires a certain level of political astuteness or street smarts. ... Using the metaphor of a stage production, consider the benefit of visualizing not only the actors on the stage, but also the producers, financiers, stagehands, marketers, benefactors, and the ultimate customer – the audience that we wish to return night after night."

9.4.1 Prozess zum Stakeholder-Konzept

Fasst man die am Stakeholder-Konzept ausgerichteten Aktivitäten als Prozess auf, lassen sich folgende Schritte unterscheiden:

1. Identifikation aller Stakeholder:
   Hierzu werden z.b. in Form eines Brainstormings alle Personen und Organisationen identifiziert, die ein Interesse (zustimmend oder ablehnend), oder zumindest Einfluss auf das Projekt haben könnten.

2. Identifikation der Interessen aller unter Punkt 1 genannten Stakeholder, deren Einfluss und relative Priorität untereinander:
   In diesem Schritt werden für jeden Stakeholder die Interessen und der potentielle Einfluss auf das Projekt identifiziert. Dabei sind möglichst alle Interessen der Stakeholder aufzulisten. Im Anschluss daran werden sie entsprechend dem Einfluss gereiht. Hierbei sind die Interessen jeweils aus der Perspektive des Stakeholders zu betrachten. Dies ist insofern schwierig, da nicht alle Interessen in Hinblick auf das Projekt offen und über die Projektdauer konstant sind. Des weiteren sind unter Rücksichtnahme auf die individuellen Interessen der Stakeholder potentielle Konflikte zwischen ihnen selbst zu identifizieren.

---

[23] Smith, Larry W., Project Clarity Through Stakeholder Analysis, in: CrossTalk, http://www.stsc.hill.af.mil , Dec. 2000

Zusammengefasst sind also folgende Punkte im Rahmen der Stakeholder-Analyse zu erörtern (zum Erarbeiten dieser Punkte eignet sich am besten eine Gruppendiskussion mit Flip-Charts, usw.):

- offene und versteckte Schlüsselinteressen
- Projekterwartungen und Benefitverteilung bei erfolgreichem Projektabschluss
- Konflikte zwischen Projekt- und Stakeholderinteressen
- Konflikte zwischen Projekt-Erfolgskriterien und Stakeholder-Erfolgskriterien
- Interessenskonflikte zwischen den Stakeholdern
- Verlauf der Interessen über die Projektdauer
- potentieller Einfluss auf das Projekt entsprechend der Merkmale: „positiv"-„negativ"-„neutral"-„unklar" und „hoch"-„niedrig"-„durchschnittlich"-„unklar"
- Stakeholder-Priorisierung

3. Bewertung der Stakeholder hinsichtlich **Gestaltungs-Einfluss (influence)** und **Entscheidungs-Gewicht (importance)** auf das Projekt:
   Diese Bewertung der Stakeholder ermöglicht eine tiefergehende Diskussion über zusätzliche Annahmen und Risken. In Abb. 9.6 sind die Stakeholder (S_X) entsprechend deren Positionierung in ein Diagramm eingezeichnet. Durch eine Quadrantenbildung werden sie in vier Gruppen zusammengefasst, wobei im oberen rechten Quadranten die Key-Stakeholder zu finden sind, welche aktiv in das Projektgeschehen einzuplanen sind. Die Stakeholder des unteren rechten Quadranten sind potentielle Risikoquellen. Sie stellen nicht unmittelbar Ressourcen für das Projekt bereit, haben aber großen Einfluss auf jene, die fürs Projekt Schlüsselleistungen erbringen.

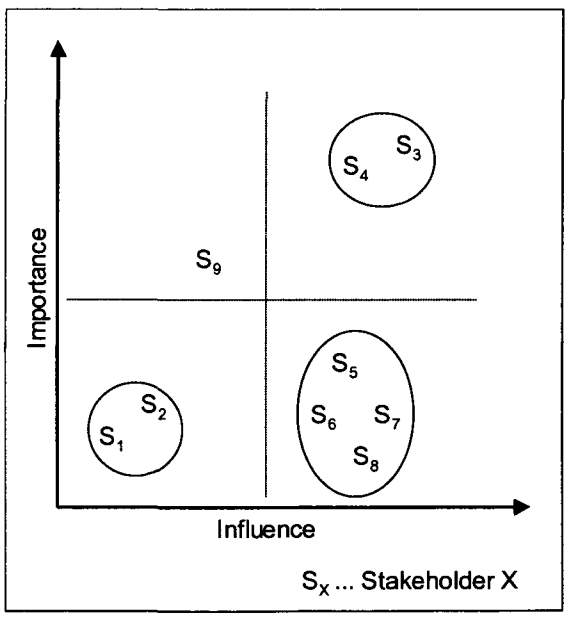


Abb. 9.6: „Importance" vs. „Influence"

Larry Smith definiert die Eigenschaften "Influence" und "Importance" in seinem Artikel folgendermaßen:

„**Influence** indicates a stakeholder's relative power over and within the project. A stakeholder with high influence would control key decisions within the project and have strong ability to facilitate implementation of project tasks and cause others to take action. Usually such influence is derived from the individual's hierarchical, economic, social, or

political position, through often someone with personal connections to other persons of influence also qualifies. Other indicators identified ... include: expert knowledge, negotiation and consensus building skills, charisma, holder of strategic resources, etc."

„**Importance** indicates the degree to which the project cannot be considered successful if needs, expectations, and issues are not addressed. This measure is often derived based on the relation of the stakeholder need to the project's goals and purposes. For instance, the human resources department may be key to getting the project new resources at a critical time, and the accounting department key to keeping the finances in order and the project manager out of jail. The users of the project's product or service typically are considered of high importance. ... A project may have an important financial sponsor that can shut down the project at any time for any reason, but does not participate at all in the day-to-day operations of the project."

Um die Importance-Influence-Relation über die Projektdauer korrekt wiederzugeben, sollte die Abb. 9.6 zu jedem Meilenstein neu erstellt werden, wodurch ein dynamisches Bild entsteht, aus dem weitere Schlussfolgerungen zur Risikoentwicklung gezogen werden können.

4. Auflistung von Annahmen (= Voraussetzungen) und Risken für jeden Stakeholder:
   Larry Smith: „Project success also depends on the validity of key assumptions and risks." In diesem Schritt werden die Annahmen und Risken betreffend der Key-Stakeholder und evtl. anderer einer Überprüfung unterzogen. Dazu werden unterschiedliche Annahmen hinsichtlich des Erfüllungsgrades der Bedürfnisse und Erwartungen (der (Key-)Stakeholder) getroffen und in Form von Szenarien deren (der (Key-)Stakeholder) Reaktionen eingeschätzt. Darauf basierend werden die Annahmen überprüft und die Risken evtl. neu formuliert. Tab. 9.2 zeigt beispielhaft eine solche Tabelle. Diese Ergebnisse stellen dann die Basis eines Risiko-Management-Plans dar.

Stakeholder	Estimated Project Influence	Estimated Project Importance	Assumptions and Risks
Owner	low	high	Providing all ressources; no specific requirements
Project Manager	high	medium	Highly motivated; lack of coordination skills

Tab. 9.2: Annahmen-Risken-Übersicht

5. Festlegung der Stakeholder-Partizipation und -Kommunikation:
   Abhängig vom Analyseergebnis des vorhergehenden Schritts wird die Einbindung der Stakeholder in die Projektorganisation festgelegt. Je nach zugedachtem Rollen-Konzept werden Stakeholder z.b. aktiv ins Projekt miteinbezogen oder lediglich über Ergebnisse in Kenntnis gesetzt. Dadurch ergibt sich eine unterschiedlich starke Integration in das Projekt, welche zudem auch je nach Phasenabschnitt variieren kann (vgl. Abb. 9.7 nach Larry Smith).

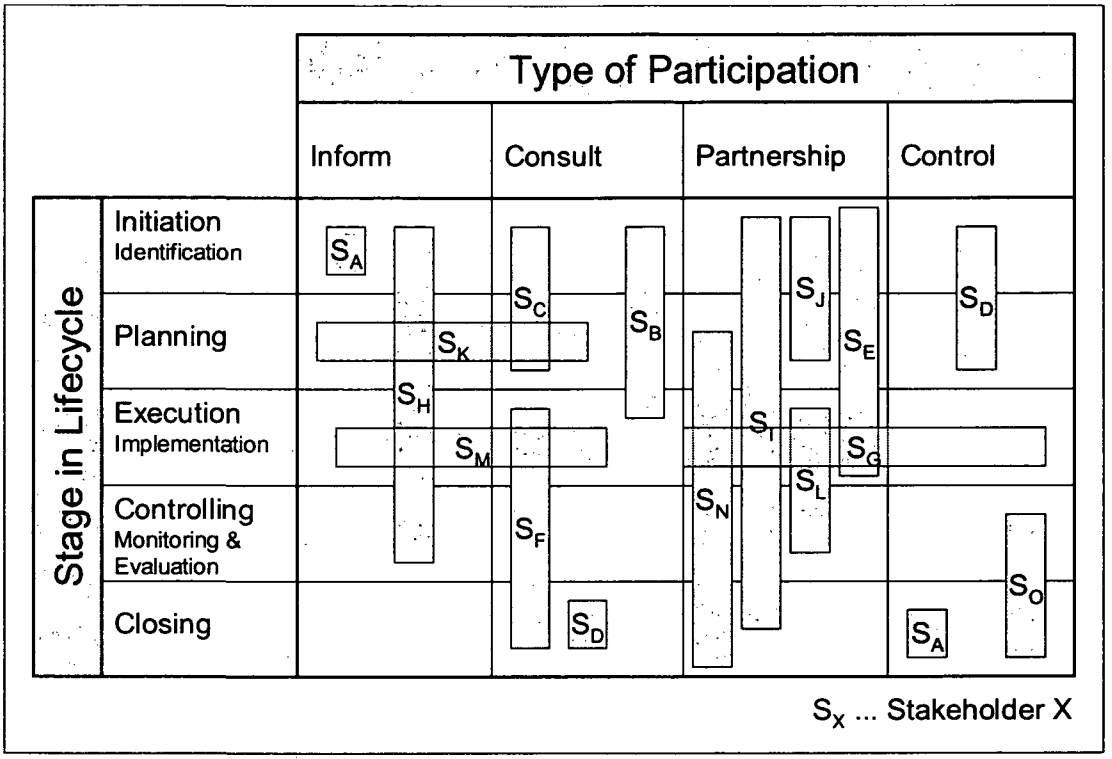


Abb. 9.7: Projekt-Partizipation

Parallel dazu wird auch der Informationsumfang, der Zeitpunkt der Informationsbereitstellung und das Format festgelegt, wie es jede Rolle zur Erfüllung der ihr zugedachten Funktion benötigt. Gegebenenfalls muss dieser Prozess während der Projektlaufzeit mehrmals durchlaufen werden. Abgesehen von regelmäßigen Controlling-Aktivitäten kann z.b. die „Frage nach dem Kunden" Anlass zum Anstoßen eines solchen Prozesses sein. Hierbei wird nachträglich analysiert wer aller Einfluss auf die Anforderungsliste (das Projektziel) hat. Mitunter nehmen Personen oder Organisationen Einfluss auf das Projektziel, die anfänglich nicht zum eigentlichen Kundenkreis gezählt wurden (z.b. einzelne Software-Entwickler).

10. Risiko-Management

„Wenn etwas schief gehen kann, dann wird es auch schief gehen.
Und wenn es mehr als eine Möglichkeit gibt, dass etwas schief geht,
so wird das schief gehen, was den größten Schaden anrichtet."

aus: Murphy's Gesetze

Der Zugang zu diesem Kapitel erfolgt über die Schlüsselfaktoren für erfolgreiches Software-Projektmanagement. Darauf aufbauend werden die den Schlüsselfaktoren zugeordneten Risken erläutert. Ein Schwerpunkt liegt in der Eingliederung des Risiko-Managements in den zugrunde liegenden Software-Entwicklungs-Prozess (vgl. 7-D-Modell nach Exl).

10.1 Einführung

10.1.1 Risikobegriff und Rahmenwerke

Gemäß der Definition in Gabler's Wirtschaftslexikon [1]) dient Risikomanagement der Erhaltung und erfolgreichen Weiterentwicklung der Unternehmung durch Bewusstmachung des Risiko-Phänomens bei allen Führungs- und auch Durchführungsprozessen und ist letztendlich risikobewusste Unternehmensführung. Die diesbezüglich anzustrengenden Aktivitäten werden zuallererst von den Planungs-, Steuerungs- und Kontrollabteilungen der betrieblichen Organisation wahrgenommen.

Aufgrund der Vielfältigkeit von Risken muss Risiko-Management als Teil der Unternehmensphilosophie und -kultur verankert werden. Im Rahmen der strategischen Planung bezieht es sich z.b. auf die Geschäftsfeld-, Organisations-, Rechtsform-, Informationssystem- und Führungskräfteplanung. Im operativen Bereich gilt es die Projekt-Programm-, Produkt-Programm- und Funktionsbereichs-Planung stets hinsichtlich der Risken zu beobachten und gegebenenfalls steuernd einzugreifen. Im Umfeld der Software-Entwicklung dient Risiko-Management in erster Linie der Vermeidung von unkontrolliertem Aktionismus. Wie in Kapitel 11 erklärt wird, sind Software-Organisationen sehr fragile Gebilde, die nur sehr bedacht veränderbar sind. Dies gilt insbesondere bei Terminnöten.

[1] Gabler-Wirtschafts-Lexikon, Wiesbaden, 1997

Im Projekt-Management wird unter dem Begriff: „Risiko" die Eigenschaft einer Situation verstanden, welche sich durch die Wahrscheinlichkeit des Eintritts eines unerwünschten Ereignisses und den im Eintrittsfall zu erwartenden Schaden charakterisiert. In Geldeinheiten ausgedrückt: Eintrittswahrscheinlichkeit multipliziert mit dem Schadensausmaß [2]). In einer Verallgemeinerung müssen die Überlegungen zum Risiko um jene der Chancen ergänzt werden. Denn jede Unternehmung ist im Grunde nur dadurch begründet, dass die Chancen – wie immer sie auch definiert sind – die Risken – welche auch immer – überwiegen.

Im Kapitalmarkt ist das Risiko unvergleichbar einfacher zu erfassen, sowohl in qualitativer als auch in quantitativer Hinsicht. Dies zeigt sich schon am universell einsetzbaren Maß der Volatilität [3]). Im Projekt-Programm-Management existiert dagegen eine Vielzahl an Risiko-Bewertungsgrößen, wie anhand der Aufstellung über die Risikoarten im weiteren Verlauf dieser Arbeit nachvollzogen werden kann. Die Schwierigkeit liegt dann darin begründet die unterschiedlichsten Chancen und Risken anhand eines allgemein gültigen Bewertungsmaßstabes quantitativ zu erfassen.

Rahmenwerke, die die Einbettung von Risiko-Management-Aktivitäten unterstützen und teilweise sogar bis ins Detail gehende Hilfestellung bieten, sind u.a.:

- ISO-9000-3 Norm:
  Sie definiert im Rahmen des Qualitäts-Managements einzelne Aspekte der Risiko-Thematik im Umfeld des Software-Projekt-Managements.

- MIL-STD-498 [4]):
  Dies ist ein US-Militärstandard, der den Einsatz von Risiko-Management-Praktiken während des gesamten Software-Entwicklungs-Prozesses spezifiziert.

- SEI-CMM bzw. kurz: CMM:
  Dies ist die weltweit bekannte Abkürzung für das Capability Maturity Model (CMM) des Software Engineering Institute (SEI) an der Carnegie Mellon University. Die Initiative zur Gründung des SEI kam vom US-Verteidigungsministerium mit dem Ziel den Software-Einkauf transparenter zu gestalten. Parallel dazu hielt es auch Einzug in die zivile Software-Industrie. Mittlerweile ist das CMM bzw. CMMI das wahrscheinlich weltweit bekannteste Referenzmodell zur Bestimmung der Prozess-Qualität.

- SPR Assessment:
  Das Ziel des SPR Assessments ist zum überwiegenden Teil identisch mit jenem des SEI-CMM-Assessments - eine Steigerung der Produktivität über die Verbesserung der Qualität. Gegründet wurde das SPR (= Software Productivity Research) Institute 1984 von Capers Jones, der ebenso wie Watts Humphrey (Gründer des SEI), ursprünglich für

[2] Patzak, Gerold, Rattay, Günter, Projektmanagement, Wien, 1998
[3] Steiner, Manfred, Bruns, Christoph, Wertpapiermanagement, Stuttgart, 2000
[4] Department of Defense, Software Development and Documentation, MIL-STD-498, AMSC no. N7069, 1994

die Firma IBM arbeitete [5]). Im Gegensatz zum Assessment durch das SEI verfolgt das SPR einen umfangreicheren Ansatz. Z.b. werden beim SPR-Assessment auch Themenkomplexe wie Gehaltsverteilung, Ausbildung des Managements, ergonomische Arbeitsplatzgestaltung, usw. untersucht. Ebenso wie beim CMM werden aufgrund des Assessment-Resultats konkrete Verbesserungsvorschläge für das Software-Projekt (bzw. Produkt)-Management abgeleitet.

- Darüber hinaus gibt es eine große Menge abgeleiteter Modelle, die auf den zuvor genannten Pionierleistungen aufbauen.

Zusammengefasst kann festgehalten werden, dass Risiko-Management Risken aller Art zu managen hat, die potentielle Auswirkungen auf das Software-Projekt, den Entwicklungs-Prozess oder das Produkt haben. Als zugrunde liegendes Modell für weitere Ausführungen zum Risiko-Management wurde aufgrund der häufigen Anwendung das CMM gewählt.

10.1.2 „Mind change"

Da aktives Risiko-Management in einem Softwarehaus im Gegensatz zum Finanz-Sektor komplexere Maßnahmen erfordert, bedarf es zur Einführung einer vergleichsweise größeren Anstrengung. Ein ernst gemeintes „Willkommen" ist oft ein guter Anfang für einen „Mind change". Dieser soll dahin führen, dass Chancen und Risken ebenso oft angesprochen werden wie Budgets, Terminpläne und technische Leistungsmerkmale. Ohne dieses Bewusstsein degradiert Risiko-Management zum Ausfüllen vorgefertigter Formulare, sofern es überhaupt als Thema adressiert wird.

E.M. Hall [6]) beschreibt die zu Beginn der Einführung eines aktiven Risiko-Managements nötige Energie mit den Worten: "It is policy, not process, that starts an organization on the path to institutionalizing a behavior. Policy is determined at the highest level in the organization". Und weiters: "Commitment is demonstrated top-down when the administration allocates resources to a task and bottom-up from the employees who support the task. ... Early on, commitment is based on trust and faith that the goal is worth the effort." Damit spricht sie ein generelles Problem an, nämlich den Irrtum, dass durch Prozess-Definitionen, selbst wenn sie nur korrekt und detailliert genug sind, die erforderlichen Tätigkeiten optimal ausgeführt werden. Insbesondere in Software-Organisationen ändern sich durch neue Tools, Outsourcing, oder einfach nur durch neue Produkte, die einzelnen Tätigkeiten mit einer derart hohen Geschwindigkeit, sodass zu detailreiche Prozesse schon zum Zeitpunkt der Bekanntmachung oft nicht mehr in einem ausreichendem Maße mit der zwischen den jeweils verantwortlichen Stellen abgestimmten Realität übereinstimmen und somit ignoriert werden.

---

[5] Jones, Capers, Assessment and Control of Software Risks, USA, 1994
[6] Hall, Elaine M., Managing Risk, USA, 2001

Aus eigener Erfahrung ist als Reaktion eine Neuauflage des Prozesses die Folge, wodurch ein Kreislauf von immer komplizierter gestalteten Prozess-Beschreibungen in Gang gesetzt wird, bis der Prozess durch einen anderen abgelöst wird. In diesem Zustand der kontinuierlichen Prozess-Neudefinition wird die Aufmerksamkeit derjenigen, für die der Prozess gelten sollte, immer geringer. Übrig bleibt die für jede Person individuell unterschiedliche Interpretation seiner Aufgabe. Diese Interpretation soll allerdings keinesfalls Anlass zur Behauptung geben, dass Prozesse keinen Nutzen haben. Sie soll lediglich andeuten, dass Prozesse ohne zugrundeliegende Motivation der beteiligten Personen rasch an Wirkung verlieren. An dieser Stelle sei angemerkt, dass die Thematik der Patterns (vgl. Kapitel 11) eine gute Alternativlösung zur Bewerkstelligung der angeführten Probleme bietet. Ein offensichtliches Zutagetreten der oben angeführten Diskrepanz zwischen Prozessplänen und Prozessdurchführungen ist insbesondere während Umstrukturierungs-Maßnahmen bzw. Personalanpassungen festzustellen.

Zusammengefasst kann festgehalten werden, dass für die Definition der Policy (= dem Zweck bzw. der Motivation zur Einführung von Risiko-Management) neben der Zurverfügungstellung von Budget, personellen Ressourcen, und einem Zeitplan, vor allem das Commitment des Top-Managements notwendig ist. Ohne diese Unterstützung ist eine Umsetzung der Policy praktisch zum Scheitern verurteilt. Ausnahmen bilden lediglich jene Bereiche, in denen die Praktiken des Risiko-Managements bereits Einzug in die tägliche Arbeit gefunden haben, wie z.b. im Finanzmanagement.

10.2 Dimensionen des Risiko-Managements

Um Risiko-Management umfassend und strukturiert zu betreiben empfiehlt sich als erster Schritt eine Diskussion der Dimensionen anhand derer Risiko-Management diskutiert wird. Grundsätzlich sollte man zwischen folgenden Dimensionen unterscheiden:

1. Gliederung nach Risiko-Arten
2. Erfolgsfaktoren für effizientes Risiko-Management
3. Aktivitäten zur Einführung und Verbesserung von Risiko-Management
4. Prozesse des Risiko-Managements
5. Verknüpfung von Risiko-Management- und Software-Entwicklungs-Prozessen

Die Nummerierung soll eine zu empfehlende Rangfolge aufzeigen um sich dem Thema des Risiko-Managements sinnvoll zu nähern. - Dies entspricht auch der Strukturierung des Kapitels.

10.2.1 Gliederung nach Risiko-Arten

Eine klassische Gliederung teilt die Risken des Software-Projekt-Managements in einen Teil, der vom Management zu verantworten ist, ein, und in einen, der den technischen Gegebenheiten unterliegt (Gliederung in 2 Risiko-Arten). Eine andere, sehr häufig anzutreffende Gliederung, unterteilt in Projekt-, Prozess- und Produktrisiko (Gliederung in 3 Risiko-Arten). Gemein ist beiden Gliederungs-Hierarchien die Anforderung, dass alle Risken der einen oder anderen Kategorie zuordenbar sein sollen. Im allgemeinen erheben die unterschiedlichen Kategorisierungen jedoch keinen Anspruch auf Orthogonalität zueinander. Während das Produktrisiko als technisches Risiko betrachtet werden kann, ist das Projekt-Risiko vorwiegend ein Management-Risiko. Und das Prozess-Risiko kann beides, ein Management- und ein Technik-Risiko, sein.

Im folgenden wird anhand der zuletzt genannten Gliederung eine Auflistung der in der US-Software-Community genannten Top-Risken, jeweils aus Software-Konsumenten- und Software-Produzenten-Sicht, gegeben [7]). Dies unter der Voraussetzung, dass der Kunde bzw. Konsument am Software-Projekt beteiligt ist. Hierbei kennzeichnet „K" ein Risiko für den Konsumenten mit entsprechender Wertung (1 ... höchste Priorität; 10 ... geringste Priorität). Die Top-10-Risken der Produzenten wurden mit „P" gekennzeichnet und ebenfalls entsprechend gereiht.

Als Beispiele für das Projekt-Risiko können genannt werden:

- Zu eng bemessene finanzielle Ressourcen (K1, P1)
- Unklare Rollen- und Verantwortlichkeits-Verteilung (K2)
- Unzureichende fachliche Qualifikation (K3)
- Abstimmungsprobleme hinsichtlich Zeitpläne, konkurrierender Leistungsmerkmale und technischer Abhängigkeiten (P4, K6)
- Terminrisiko (K9)

Beispiele für Prozess-Risken sind („M" ... Management bedingt; „T" ... technisch bedingt):

- Unzureichende Entwicklungsprozess-Definition (P3T, K4T)
- Unzureichende Projekt-Planung (K5M, P5M)
- Mängel hinsichtlich der Software-Entwicklungs-Umgebung (z.b. ungenügend geschultes Personal für einen effizienten Einsatz (P6T)
- Systems-Engineering-Mängel (z.b. Mängel hinsichtlich der Systemarchitektur, Benutzeroberfläche, Sicherheitseinrichtungen, Upgrademöglichkeit) (K7T)
- Management-Fehler (z.b. unzureichendes Reporting und Qualitäts-Management) (P8M)
- Kommunikationsprobleme zwischen Hierarchien und Teams (P9M)
- Unzureichende Systemtests (K10T)

[7] Hall, Elaine M., Managing Risk, USA, 2001

Beispiele für Produkt-Risken sind:

- Unzureichende Leistungs-Merkmal-Dokumentation (P2)
- Performance-Probleme (z.b. zu lange Systemantwortzeiten) (P7)
- Abweichung zwischen gewünschter und realisierter Funktionalität; auch hinsichtlich der MTBF (= Mean Time Between Failures) (K8)
- Mängel bei der Systemintegration mit Legacy-Systemen (Legacy-Systeme sind bereits beim Kunden in Betrieb befindliche Software-Systeme, die mit den neu zu liefernden Systemen kooperieren müssen) (P10)

Aufgrund funktionaler Organisationsstrukturen in der betrieblichen Organisation werden Risken sehr oft nach folgenden Kriterien unterteilt (mit Angabe zusätzlicher Beispiele für Risken aus dem Software-Projekt-Management):

- Technologische Risken
  z.b. Technologiesprünge: Da sich Software-Entwicklungen meist über Monate bis hin zu einigen Jahren erstrecken und die Produktzyklen in dieser Branche meist sehr kurz sind, kann ein Technologiesprung oder –wechsel bei Produkt-Entwicklungen zur völligen Unvermarktbarkeit führen. Ein klassisches Beispiel hierfür sind die teilweise konkurrierenden Betriebssysteme auf denen die Applikationsentwicklung aufsetzt. Ein anderes ist die laufende Weiterentwicklung von Schnittstellen-Standards.

- Abwicklungstechnische Risken:
  z.b. Inkludierung von OEM-Produkten (= Original Equipment Manufacturer; hiermit werden Fremdprodukte bezeichnet).

- Organisatorische Risken:
  z.b. Lernkurveneffekt: Eine zu späte Einbeziehung zusätzlicher Personal-Ressourcen bewirkt den gegenteiligen Effekt – das Projekt verzögert sich nochmals.

- Terminrisiken:
  z.b. Pönale aufgrund von Terminverschiebungen.

- Wirtschaftliche Risken:
  z.b. Enges Marktfenster: Durch den kurzen Lebenszyklus von Software-Produkten sind die Marktfenster meist relativ kurz.

- Juristische Risken:
  z.b. Vertragsgestaltung: Ohne auf bestimmte Punkte eingehen zu wollen, ist die detaillierte Ausformulierung des Leistungsumfangs von entscheidender Bedeutung. In vielen Projekten ist dieser dem Auftraggeber zu Vertragsabschluss selbst nicht zur Gänze bekannt. Eine wichtige Funktion übernehmen hierbei vertraglich geregelte Abnahmekriterien. Des weiteren sollte auch das Vorgehen bei Änderungswünschen (inkl. Finanzierungsthematik) im vorhinein geregelt sein.

10.2.2 Erfolgsfaktoren für effizientes Risiko-Management

Einen Übergang zur Strukturierung der Risken entsprechend den Erfolgsfaktoren zeigt Abb. 10.1 aus dem Buch: Managing Risk [8]), welche um den weiteren entscheidenden Faktor: „Self-Interests" erweitert wurde. Dieser Faktor berücksichtigt die Tatsache, dass handelnde Personen nicht ihren gesamten Einsatz ausschließlich den Zielen der Organisation bzw. dem Unternehmen widmen, sondern teilweise auch eigenen Interessen nachgehen.

Das Fischgräten-Diagramm spannt das Thema Risiko-Management in 4 Dimensionen auf, wodurch eine über die primitive Risiko-Arten Kategorisierung hinausgehende Strukturierung und Diskussion ermöglicht wird. Die 4 Dimensionen bzw. Kategorien, die die Fähigkeit einer Organisation Risiko zu managen bestimmen, sind (näheres hierzu in den folgenden Abschnitten):

1. Personen
2. Prozesse
3. Infrastruktur
4. Implementierung

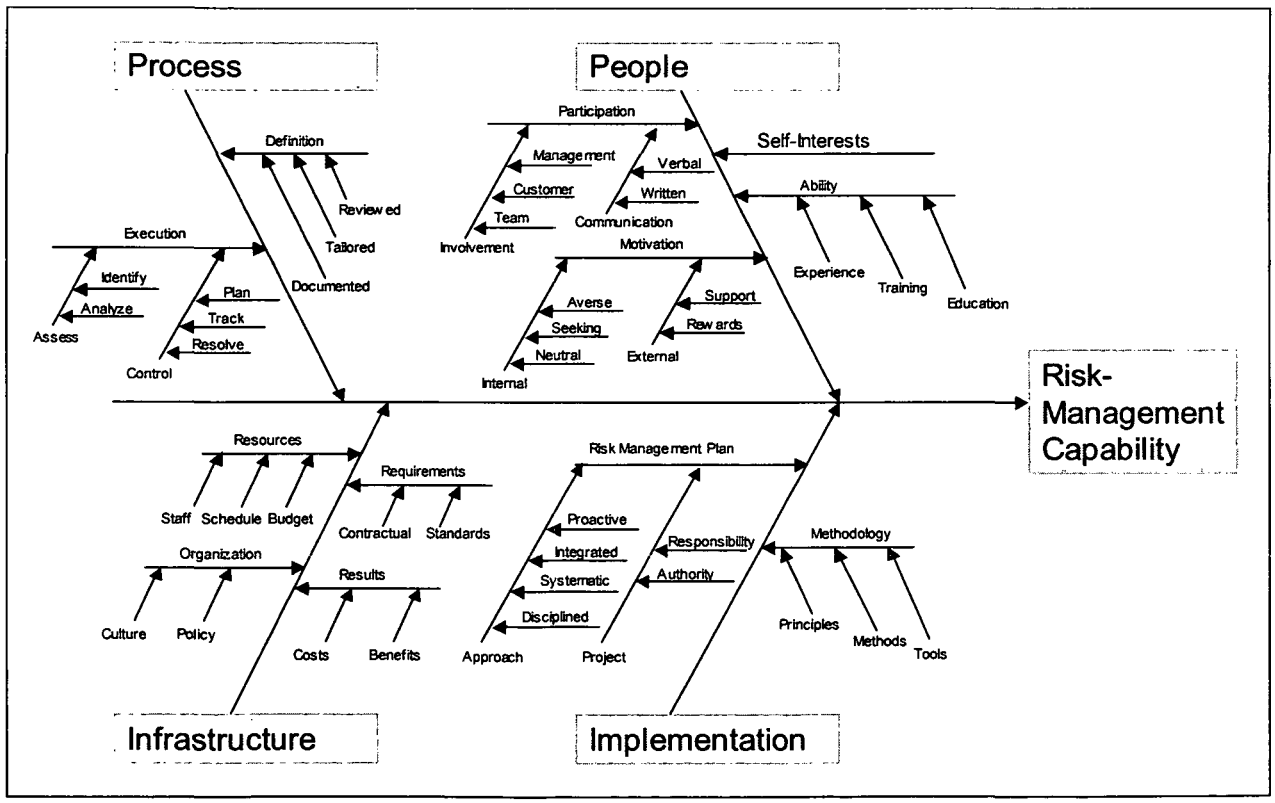


Abb. 10.1: Erfolgsfaktoren (Risken) des Software-Projekt-Managements

[8] Hall, Elaine M., Managing Risk, USA, 2001

10.2.3 Chancen und Risken als Teil des Software-Entwicklungs-Prozesses

Als Ergänzung zu dem in Kapitel 6 beschriebenen Spiral-Modell von Barry Boehm soll nun eine grundlegende Diskussion über den Regelkreismechanismus der Produktentwicklung unter Berücksichtigung relevanter Einflussfaktoren geführt werden. Hierbei wird neben dem Bezug zur Chancen-Risiko-Thematik auch eine Verknüpfung mit dem Capability Maturity Model des Software Engineering Institute an der Carnegie Mellon University (SEI-CMM; im weiteren mit CMM referenziert) hergestellt.

Bildet man den Software-Entwicklungsprozess auf den „Plan-Do-Check-Act"-Zyklus von W. Edwards Deming [9]) ab, kommt man zu einer Systemdarstellung, welche auch das Risiko adäquat in den Entwicklungs-Prozess integriert. Hierbei ist die wiederholte Anwendung der Zyklen die Grundlage zur Produkt-Optimierung entsprechend den Marktanforderungen.

Um eine Kontinuität zur Begriffswelt des CMM herzustellen, werden die Begriffe: „Do", „Check" und „Act" durch: „Work", „Measure" und „Improve" ersetzt. E.M. Hall [10]) hat dieses Regelkreis-Modell um zwei weitere Aktivitäten erweitert, die hinsichtlich der Diskussion über Chancen [11]) und Risken eine zentrale Rolle spielen. Diese sind: „Envision" und „Discover". (An dieser Stelle sei angemerkt, dass die Ausführungen hierzu in engem Zusammenhang mit jenen zum Strategischen Management stehen.)

Die einzelnen Aktivitäten der Abb. 10.2 lassen sich wie folgt charakterisieren:

Beginnt man die Diskussion mit jener Aktivität die mit „D" bezeichnet wird, steht man zunächst vor dem Rätsel über die Zukunft. „Enigma" bezeichnet die Wahrnehmung möglicher zukünftiger Entwicklungen und Ereignisse mit Relevanz zur betrachteten unternehmerischen Tätigkeit. In dieser Phase der Kontemplation wird der größte Anspruch an die Kreativität aller an der zukünftigen Strategie arbeitenden Mitarbeiter gestellt. Im Regelkreis wird diese Kreativitätsphase mit „D" (= „Discover") bezeichnet.

In einem nächsten Schritt werden die Ideen als Strategien und Ziele formalisiert – „E" (= „Envision").

Anschließend wird ein Plan erstellt, der mit den zur Verfügung stehenden Ressourcen die angestrebten Anforderungen abzudecken sucht. Für die Erstellung des Plans („P") sind die Ziele („Goals"), die Risken – die sich aus dem Verhalten der Umwelt („Risks") und den Aktivitäten innerhalb des Regelkreissystems selbst ergeben, die Abweichungen („Variance") vom ursprünglichen Plan, und die Verbesserungen – in Form von neuen Maßstäben („Metrics") relevant.

---

[9] Deming, Edwards, Out of the Crisis, Cambridge, MA, USA, 1986
[10] Hall, Elaine M., Managing Risk, USA, 2001
[11] Versteegen, Gerhard (hrsg.), Risikomanagement in IT-Projekten, Berlin, 2003

Im Prozessschritt der Erstellung („W", = „Work") werden alle jene Aktivitäten summiert, die der Erstellung des Software-Produkts dienen (z.b. Design-Dokumente, Testpläne, Codierung, Tests, usw.). Dieser Schritt bindet die meisten Ressourcen. Die Unsicherheiten, die während der Fertigstellung des Produkts aufkommen, und der Status während der Realisierung dienen als Input für die nachfolgenden Aktivitäten.

Der Prozessschritt der Messung („M", = „Measure"), z.b. in Form der Fortschrittskontrolle oder der Bestimmung der Produktqualität, stellt nach wie vor eine komplexe Herausforderung an die Software-Entwicklung dar. Dieser Schritt wird in Kapitel 8 diskutiert.

Um eine nachhaltige Konkurrenzfähigkeit zu gewährleisten müssen andauernd Anstrengungen zur Produkt- und Prozessverbesserung („I", = „Improve") unternommen werden (vgl. hierzu den Kaizen-Gedanken).

Nachdem im Schritt „M" die Plan- und Ist-Parameter des Produkts bzw. des Prozesses verglichen wurden, erfolgt in diesem Schritt ein Vergleich mit der System-Umwelt. Als Maßstab werden „Benchmarks" herangezogen die als Referenz („Metrics") in die Planung des nächsten Zyklus miteinfließen.

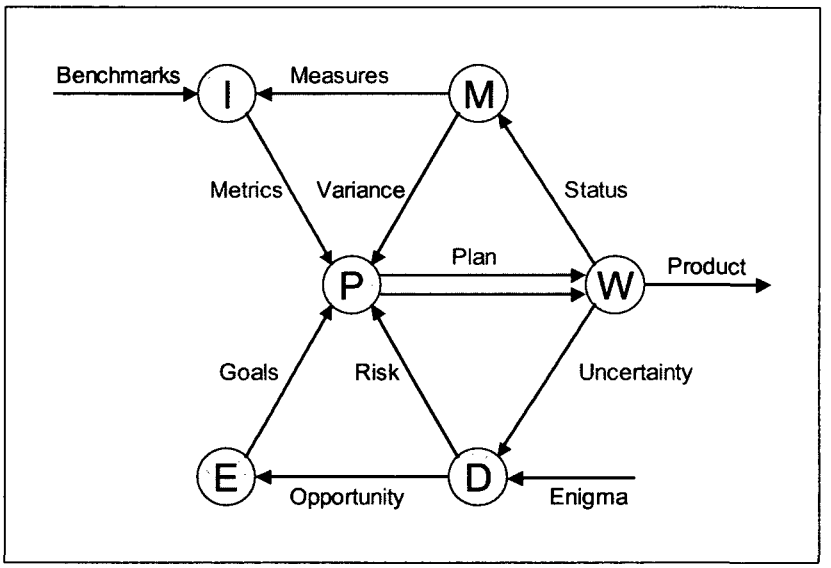


Abb. 10.2: 6-Disziplinen-Modell nach Hall

Bezieht man die Aktivitäten des Regelkreises auf die Reifegrade des CMM (= Capability Maturity Model) lassen sich grob folgende Zuordnungen treffen (hierbei bezeichnen die Ziffern die Stufen des CMM in denen die „Disciplines" [„Envision", „Plan", usw.] verankert sind):

- CMM 1: „Envision"
- CMM 2: „Plan"
- CMM 3: „Work"
- CMM 4: „Measure"
- CMM 5: „Improve"

[11] Versteegen, Gerhard (hrsg.), Risikomanagement in IT-Projekten, Berlin, 2003

Durch diese Zuordnung von Tätigkeiten zu einem CMM-Level wird gleichzeitig die Reihenfolge von Prozess-Verbesserungen angegeben. Diese Reihenfolge bringt zum Ausdruck, dass Verbesserungen in der Codierungsphase der Software-Entwicklung ohne vorhergehende Verbesserungen hinsichtlich der Zielsetzungen und Planungsaktivitäten nur wenig zur Produktverbesserung beitragen.

Der besondere Reiz dieses Regelkreis-Systems liegt in der gleichzeitigen Modellierung des Kaizen-Gedankens einerseits, und des Innovations-Gedankens andererseits. – Wie die Vergangenheit durch den Zyklus: „Planning -> Working -> Measuring -> Improving" Einfluss auf die kontinuierliche Produkt-Verbesserung nimmt, werden zukünftige Innovationen durch den Zyklus „Planning -> Working -> Discovering -> Envisioning" einzubinden versucht. Für die Software-Entwicklung bezieht sich der Begriff: „Enigma" nicht nur auf innovative Ideen im klassischen Sinn. Auch die Einbindung neuer OEM-Produkte oder die Einführung neuer Entwicklungsumgebungen muss als Innovation betrachtet werden, die sowohl Chancen als auch Risken mit sich bringt (vgl. hierzu die Ausführungen zum OEM-Prozess).

10.2.3.1 7-Disziplinen-Modell nach Exl

Obwohl das 6-Disziplinen-Modell (6-D-Modell) gegenüber jenem von Edwards Deming um zwei weitere Aktivitäten („Discover" und „Envision") erweitert wurde, soll auch dieses erweiterte Modell nicht darüber hinwegtäuschen, dass es eine modellhafte Abbildung der Realität ist. Seine Existenzberechtigung ist aber aufgrund der bisherigen Ausführungen ohne Zweifel gegeben. Weitergehende Adaptionen an die Realität können durch den Einbau von in den Patterns (vgl. Kapitel 11) dargelegten Sachverhalten erfolgen. Eine derartige Erweiterung des Modells ist Gegenstand des nächsten Absatzes.

Basierend auf eigenen Erfahrungen liegt eine Schwäche des 6-D-Modells in der fehlenden Berücksichtigung allfälliger menschlicher Verhaltensweisen („HB", = „Human Behaviour") bzw. Eigeninteressen oder Bedürfnissen. Wie aus den Diskussionen über Patterns hervorgeht spielen persönliche Ziele und zwischenmenschliche Beziehungen innerhalb von Teams bzw. über Teamgrenzen hinweg eine wesentliche Rolle, deren Berücksichtigung eine organisatorische Voraussetzung für erfolgreiche Software-Projekte ist.

Das um den Faktor der menschlichen Verhaltensweise („HB") ergänzte 6-D-Modell soll im weiteren Verlauf dieser Arbeit als 7-D-Modell nach Exl bezeichnet werden (vgl. Abb. 10.3). Die wesentliche Änderung gegenüber dem 6-D-Modell ist der Einfluss von Eigeninteressen auf das Ergebnis – das Software-Produkt. Die Eigeninteressen, die umso stärkeren Einfluss haben, je mächtiger die Rolle des Einzelnen im Software-Projekt ist, können sehr gut mit der Bedürfnis-Pyramide von Maslow modelliert werden. Dieser durch Eigeninteressen begründete Einfluss wirkt vor allem auf die Zielsetzungen und den Prozess-Schritt des Planens ein. Korrigiert wird er gegebenenfalls aufgrund der Resultate aus dem Benchmarking-Vorgang, dessen positives oder negatives Ergebnis u.U. auch persönliche Konsequenzen („Feedback") für den

verantwortlichen Manager mit sich zieht. Eine theoretische Basis für die Modellierung der Wirkung von Eigeninteressen innerhalb von Linien- und Projekthierarchien liefert die Principal-Agent – Theorie (vgl. hierzu Kapitel 3, Abschnitt 3.1.2).

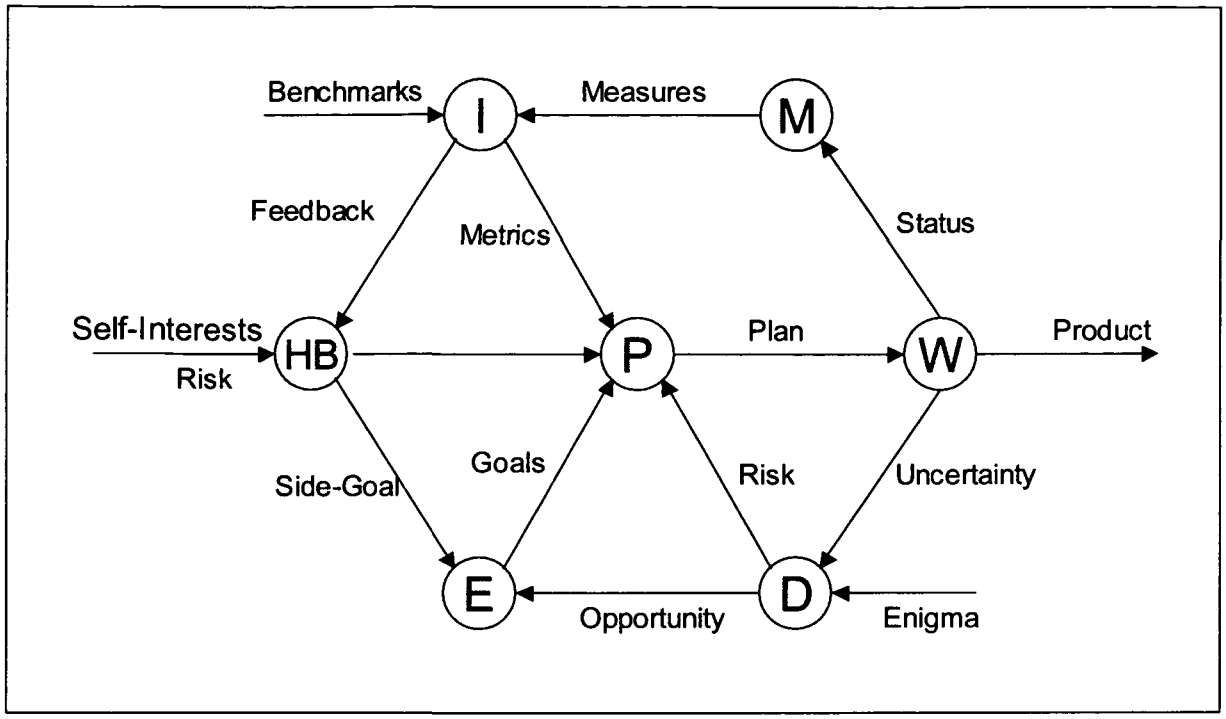


Abb. 10.3: 7-Disziplinen-Modell nach Exl

Der bereits als Erfolgsfaktor identifizierte Einfluss der Eigeninteressen kann sich im Rahmen der Software-Entwicklung auf vielfältige Weise äußern. Beispiele hierfür sind:

- Einflussnahme bei der Auswahl von OEM-Produkten
- Projekt-Team-Zusammensetzung aufgrund persönlicher Sympathien
- Strategie-Auswahl aufgrund individuell-attraktiver Perspektiven
- individuelle Schwerpunktsetzungen (z.b. Entwicklung von Software mit Funktionalitäten, die über die Anforderungen hinausgehen)
- Auswahl von Realisierungsmethoden bzw. Tools die dem letzten Stand der Technik entsprechen, jedoch noch nicht die für industrielle Anwendungen notwendige Stabilität aufweisen
- subjektive Bewertung von Alternativen aufgrund der für die jeweiligen Teams resultierenden Folgen (z.b. Projektabbruch und folglich Teamauflösung)

Der Einfluss der Eigeninteressen wirkt im oben beschriebenen Regelkreis sowohl auf die Entwicklung der Strategien („E") als auch auf die daran anschließende Entwicklung der Projektpläne („P"). Die Möglichkeit des Einflusses auf die anderen Aktivitäten bzw. Knoten in der Abbildung ist ebenfalls gegeben, jedoch marginal. Ein weiteres in der Praxis

vorzufindendes Element, das dem Knoten „HB" zuzurechnen ist, ist die individuelle Beherrschung der Kunst der Dialektik im Entscheidungsfindungs-Prozess. Oft werden in der Praxis Argumente ins Feld geführt, die keinesfalls die Antithese zu einem vorgebrachten Argument darstellen. Darüber hinaus wirken Überzeugungskraft, Auftreten, Overhead-Folien, uvm., die eine Entscheidung von der sachlichen Ebene abrücken können. Davon unabhängig ist die Möglichkeit einer aus Eigeninteressen begründeten Einflussnahme stets proportional dem Informationsmangel bei anliegenden Entscheidungen.

Eine weitere marginale Änderung im 7-D-Modell nach Christian Exl gegenüber dem 6-D-Modell von E.M. Hall ist die Zusammenfassung der Information über die Plan-Ist-Abweichung mit jener des Benchmarking bevor sie auf die Planungs-Dimension Einfluss nimmt. Dadurch werden die aus der Messung und von außen (der Umwelt) kommenden Metriken bereits in der Aktivität „Improve" („I") aufbereitet, bevor sie auf die Planungen der nächsten Iteration Einfluss nehmen.

10.2.4 Best-practice Strategie zur Einführung von Risiko-Management

Der Wille zum Risiko-Management ist die Voraussetzung um es zu etablieren. – Die schrittweise Einführung der Weg. Als theoretisches Fundament zur Beschreibung des Weges eignen sich die bereits dargelegten allgemeinen Erkenntnisse des Strategischen Managements (vgl. hierzu Kapitel 3), ergänzt um das für Software-Projekte entwickelte CMM.

Aus dem Strategischen Management leitet sich die grundsätzliche Vorgehensweise ab, wonach zu Beginn jedes Vorhabens eine Vision steht, die als solche formuliert und möglichst breit kommuniziert wird. Die Vision beschreibt den Ideal-Zustand, der durch das Erreichen einer überschaubaren Anzahl von Zielen verwirklicht werden soll. Durch die Zielformulierungen wird erreicht, dass anders als durch die Vision, jeder Einzelne auf konkretere und damit gemeinhin verständlichere Weise angesprochen wird. Damit ist für Teams – oder auch einzelne Personen – der Weg geebnet Strategien zur Erreichung dieser Ziele zu erarbeiten.

Um auf dem langen Weg der Verwirklichung einer Vision nicht an Motivation zu verlieren, sollten in weiter Ferne liegende Visionen über mehrere Stufen (mit jeweils klar formulierten Zwischenzielen) erreichbar sein. Dieses schon im Zusammenhang mit den Phasen eines Entwicklungs-Prozesses dargelegte Prinzip wird nun auch auf das Management der Risken angewandt. Dies bedeutet, dass mit jeder weiteren Stufe eine größere Fähigkeit zum Management der noch unbekannten Einflüsse auf dem Weg zur Vision erreicht wird. Spätestens jetzt stellt sich die Frage nach der Vision des Risiko-Managements, welche so beantwortet werden kann: „Die Vision ist die Umwandlung aller Risken in Chancen."

Als Vorbild zur Erreichung dieses Ziels dient die Vorgehensweise des bereits mehrmals erwähnten CMM. Auch spricht für das beim CMM zugrunde gelegte Prinzip die weltweite Akzeptanz (und Umsetzung in innerbetriebliche Prozesse) seitens der Entwicklungs-Abteilungen von Softwarehäusern. Bezogen auf die Aufzählung der oben erwähnten Erfolgsfaktoren von E.M. Hall lässt sich die stufenweise (1., ... 5.) Einführung von Risiko-Management-Aktivitäten wie in Abb. 10.4 darstellen:

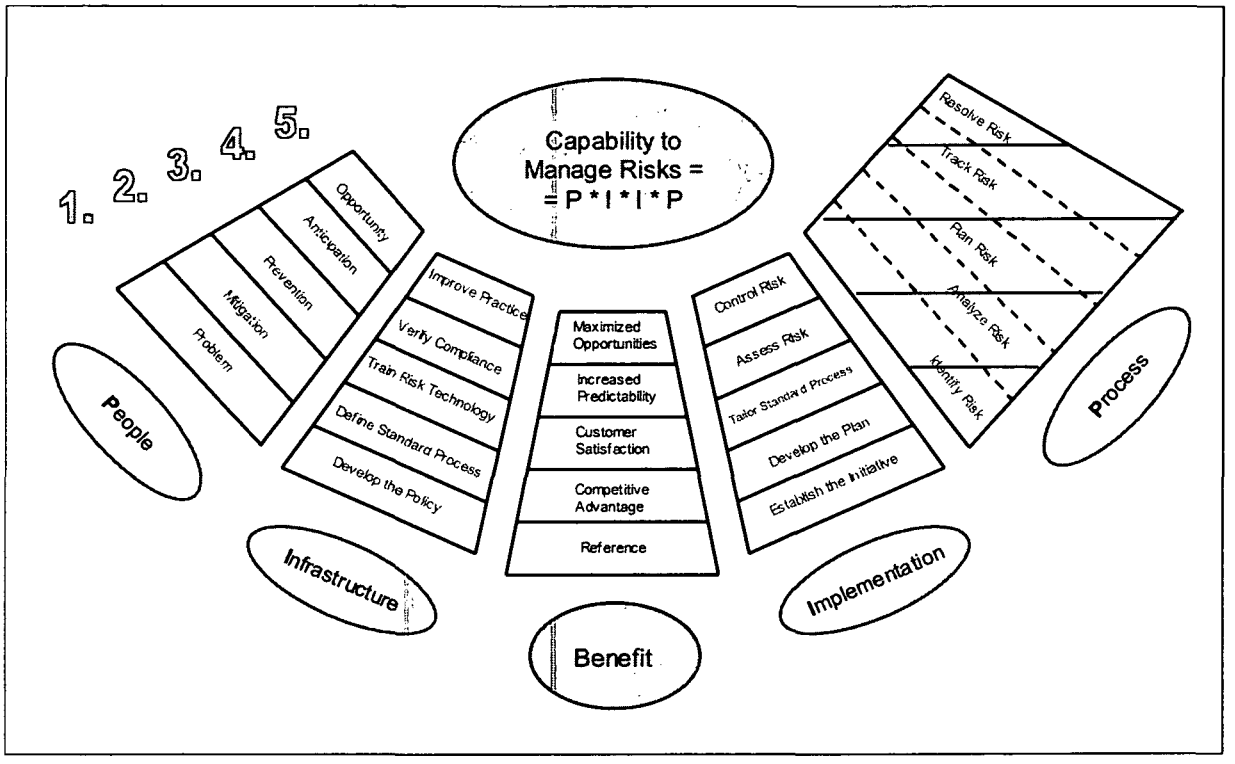


Abb. 10.4: „The Road to Risk-Management Capability"

In dieser Abbildung sind die vier wesentlichen Gruppen der Erfolgsfaktoren in Form von Trapezen visualisiert. Die Ziffern "1." bis "5." stehen äquivalent zum CMM-Modell für den erreichten Fortschritt bzw. die Reifestufe innerhalb jeder Erfolgsfaktorengruppe ("People", "Infrastructure", "Implementation" und "Process"). Mit jeder erreichten Stufe (max. "5.") innerhalb einer Faktorengruppe (z.b. "People") wächst die Beherrschung der Risken, welche dieser Faktorengruppe (vgl. Abb. 10.1) zugerechnet werden. Wird z.b. durch entsprechende Maßnahmen sichergestellt, dass der Einfluss auf Entscheidungen aufgrund von Eigeninteressen weitestgehend eliminiert ist, erreicht diese Faktorengruppe eine höhere Stufe des Reifegrades (unter der Annahme, dass andere notwendige Kriterien dieser Gruppe ebenfalls erfüllt werden).

Wegweisend an diesem Modell ist der Ansatz zur Quantifizierung des erreichten Grades an Risiko-Management in einer Organisation. Hierzu wird das Produkt über alle in der Organisation erreichten Reifestufen bzw. -grade der vier Faktorengruppen gebildet. Dies

impliziert jedoch, dass die Fähigkeiten zwischen den einzelnen Gruppen innerhalb einer Organisation nicht zu stark variieren dürfen, d.h. dass die Ausprägung der Fähigkeiten auf die Stufen: 1-2-3 oder 2-3-4 oder 3-4-5 verteilt sein sollte. Nur auf diese Weise kann ein sinnvoll begründetes Risiko-Management etabliert werden – wie die folgenden Überlegungen zeigen.

Unter der Annahme (Beispiel), dass eine Fokussierung hinsichtlich der Beherrschung der Risken ausschließlich auf die Faktorengruppe „Process" (mit maximalem Einsatz, wodurch ein Reifegrad der Stufe 5 erreicht wird) stattfindet, liefert die in der Abbildung angegebene Formel (Multiplikation der durch Ziffern charakterisierten erreichten Stufe innerhalb einer Gruppe) ein Ergebnis von 5 (-> 1*1*1*5 = 5). Werden hingegen geringe Anstrengungen zur Risiko-Beherrschung in jeder einzelnen Gruppe unternommen (Annahme, dass z.b. in jeder Gruppe die 2. Reifestufe erreicht wird) kann dies hinsichtlich der Beherrschung des Gesamtrisikos bereits merkliche Resultate zeigen (-> 2*2*2*2 = 16). Je größer dieser Wert, umso besser ist das Gesamtrisiko „unter Kontrolle" (max. 5*5*5*5 = 625). Wie die Beherrschung des Gesamtrisikos zu interpretieren ist wird durch das Trapez mit der Bezeichnung: „Benefit" beschrieben: „Capability to Manage Risks = P * I * I * P", wobei die Buchstaben P, I, I und P für: „People", „Infrastructure", „Implementation" und „Process" stehen. Das „ * " symbolisiert die mathematische Operation der Multiplikation.

Eine Ausnahme in der einheitlichen Darstellung der stufenweisen Rangfolge (vgl. Abb. 10.4) einzelner Fähigkeiten bilden die Prozesse, von denen jeder mehr oder weniger etabliert sein muss (zumindest ab dem 3. Reifegrad). Aus diesem Grund ist zwar eine gewisse Reihenfolge in der Einführung der einzelnen Prozesse (Identify-, Analyze-, Plan-, Track-, Resolve- Risk) zu berücksichtigen, jedoch ist keineswegs gefordert, dass z.b. der „Identify-Risk"-Prozess zu 100% etabliert sein muss bevor der nächste („Analyze Risk"-Prozess) eingeführt wird. Damit wird gegenüber der ursprünglichen Darstellung von E.M. Hall die strikte Rangordnung der Prozesse („Identify Risk" ... 1.Stufe; „Resolve Risk" ... 5.Stufe) zugunsten einer Ressourcen-Optimierung aufgegeben. Dem liegt zugrunde, dass bei gleichem Ressourcen-Einsatz eine parallele Optimierung der Prozesse ein besseres globales Optimum hinsichtlich der Beherrschung des gesamten Risiko-Management-Prozesses ergibt (gegenüber einer sequentiellen Optimierung). Überdies setzt sich der gesamte Risiko-Management-Prozess aus allen fünf Teil-Prozessen bzw. Prozess-Schritten zusammen (vgl. Abb. 10.11), wodurch der hier dargestellte Weg der Optimierung ebenfalls begründet wird.

Im weiteren Verlauf der Arbeit werden die vier Erfolgsfaktorengruppen gesondert diskutiert, wobei der Prozess-Thematik ein eigener Abschnitt gewidmet ist.

10.3 Erfolgsfaktoren: Mensch, Infrastruktur und Implementierung

Nachdem in vielen Publikationen zum Thema Risiko-Management fast ausschließlich der Risiko-Management-Prozess behandelt wird, sollen in diesem Abschnitt insbesondere jene Erfolgsfaktorengruppen dargestellt werden, die zur Beherrschung der Risiko-Thematik als Ganzes mindestens ebenso viel Einfluss haben: Menschen, Infrastruktur und Implementierung, bzw. "People", "Infrastructure" und "Implementation" (vgl. Abb. 10.4).

10.3.1 Erfolgsfaktor Infrastruktur

Die Infrastruktur bereitet das Umfeld auf in dem sich das Risiko-Management etablieren kann. Wie aus Abb. 10.5 ersichtlich ist, steht die Formulierung der Policy („**Develop the Policy**") – der Firmenpolitik bzw. Firmenkultur in bezug auf Risiko-Management – am Anfang aller Vorbereitungen. Eine wichtige Voraussetzung zur Schaffung einer diesbezüglich fruchtbaren Firmenkultur ist das Aufbereiten einer Risiko-Ethik [12]), die innerhalb der betreffenden Organisation jeden einzelnen Mitarbeiter dazu anhält:

- Verantwortung für Risken zu übernehmen,
- niemanden wegen Risken zu beschuldigen,
- Risken an andere zu kommunizieren,
- pro-aktives Engagement einzubringen, und,
- von unerwarteten Ereignissen zu lernen.

Weitere unbedingt notwendige Zutaten in der Initiierungsphase sind die Bereitstellung von Budget und das Suchen engagierter Pioniere, die sich gerne einem neuen Thema widmen. Weiters ist darauf zu achten, dass die Mitarbeit in Abstimmung mit dem jeweiligen Vorgesetzen stattfindet und mit anderen Projekten bezüglich der Ressource Zeit nicht kollidiert.

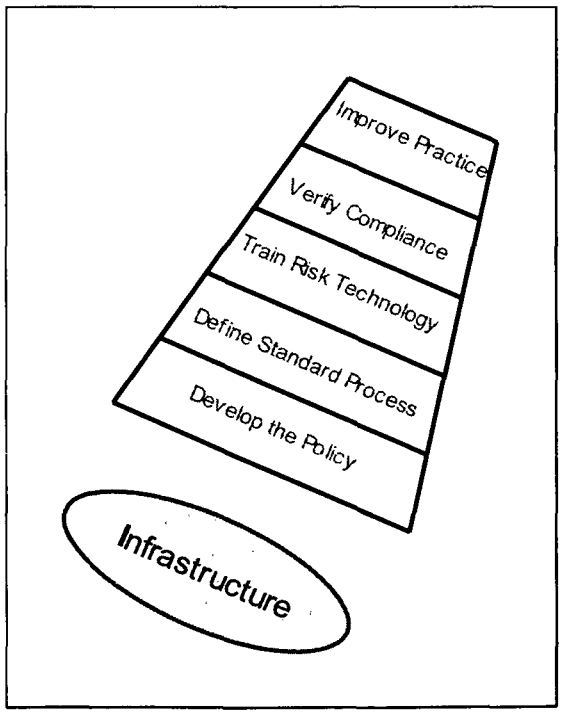


Abb. 10.5: Infrastructure

[12] Van Scoy, R., Problem or Opportunity, Technical Report CMU/SEI-92-TR-30, Pittsburgh, PA, USA, 1992

Um eine über mehrere Teilorganisationen gültige Policy zu erstellen sollten die daran arbeitenden Pioniere Erfahrungen aus den unterschiedlichen Bereichen mitbringen und sich auf eine gemeinsame Terminologie einigen. Inhalte der Policy sind u.a.:

- Beschreibung der Motivation für die Einführung der Risiko-Management-Aktivitäten
- Ziele nach der Einführung der Risiko-Management-Aktivitäten
- Definition der die Policy betreffenden Teilorganisationen
- Grob-Festlegung der Verantwortlichkeiten für Risiko-Management-Aktivitäten innerhalb der Organisation
- Erster Ansatz für den gesamten Risiko-Management-Prozess
- Referenz auf Standards, Dokumente, usw.

In einem anschließenden Review sollen möglichst alle Mitarbeiter der betroffenen Teilorganisationen die Möglichkeit bekommen, ihre Meinung zur Policy abzugeben. Dies ist ein wichtiger Schritt für das Erlangen weitreichender Akzeptanz. Nach Bestätigung der endgültigen Policy durch das Senior-Management wird die Policy an alle kommuniziert.

Während in der ersten Stufe (vgl. Abb. 10.5) ein gemeinsames Verständis über das Thema Risiko-Management geschaffen wurde, wird im nächsten Schritt ein Standard-Prozess („**Define Standard Process**“) festgelegt. Das Erarbeiten eines Standard-Prozesses soll als erster Schritt zum Entwurf des später in die Praxis umzusetzenden Risiko-Management-Prozesses verstanden werden. Hierfür sind dann die Ergebnisse aus der Diskussion über die Bedeutung der anderen Erfolgsfaktoren (aus den Gruppen: „People“, „Implementation“, „Process“) wesentlich. Zur inhaltlichen Vorbereitung des einzuführenden Risiko-Management-Standard-Prozesses empfielt sich das Anlegen einer Dokumenten-Sammlung, die Unternehmens-Prozesse, Unternehmens-Richtlinien, einschlägige Lehrmaterialien und Konferenzunterlagen, relevante Internetseiten, Best-Practice-Berichte und Sammlungen über Standards (ISO 9000-Reihe, SEI-CMM, IEEE, u.a. zum Thema Risiko-Management) in strukturierter Weise enthält.

Seine der Organisation entsprechende Individualität erlangt der Prozess durch die Adaptierung an die konkreten Gegebenheiten und Ziele, welche den Erfolgsfaktoren der Gruppe: „Infrastructue“ zuzurechnen sind:

- Reduktion bürokratischer Tätigkeiten
- Elimination von Doppelgleisigkeiten
- Schaffung von Kundennutzen
- Design eines innerhalb der Organisationsgrenzen möglichst universell einsetzbaren Prozesses
- Rücksichtnahme auf angewandte Verfahren
- Berücksichtigung der Anwendung von Tools (u.a. Einbeziehung von Tools aus dem Software-Entwicklungs-Prozess)

Das Erfüllen der Anforderungen der 2.Stufe der Faktorengruppe: „Infrastructure" wird nach einem breit gestalteten Review, der Bestätigung durch das Senior-Management und einer Verteilung des Prozess-Entwurfs an alle betroffenen Organisationsmitglieder offiziell.

Wenn alle Mitarbeiter einer Organisation über die grundsätzliche Thematik des Risiko-Managements informiert sind und ein auf die Organisation zugeschnittener Standard-Prozess existiert, empfiehlt sich die Vorbereitung und Durchführung entsprechender Trainingseinheiten („**Train Risk Technology**"). – Notwendig ist dabei immer die Abstimmung mit den Fortschritten in den anderen Schlüsselbereichen („People", „Process" und „Implementation").

Session 1	**Session 2**	**Session 3**	**Session 4**	**Session 5**
Crisis Management	Acceptable Risk	Causal Analysis	Cost-Benefit Analys.	Creativity
Loss	Choice	Corrective Action	Measurement	Opportunity
Rework	Consequence	Diversification	Metrics	Opportunity Cost
Risk	Decision	Proactive Risk M.	Quantitative Targets	Problem Prevention
Uncertainty	Estimation	Process	Risk Forecast	Risk Control
	Evaluation	Risk Action Plan	Risk Index	
	Probability	Risk Analysis	Risk Leverage	
	Risk Assessment	Risk Category	Risk Preference	
	Risk Checklist	Risk Context	Risk Scenario	
	Risk Exposure	Risk Database	ROI(RM)	
	Risk Identification	Risk Drivers	Threshold	
	Risk List	Risk Managem. P.	Trigger	
	Risk Management	Risk Planning	Utility Function	
		Risk Resolution		
		Risk Statement		
		Risk Tracking		

Tab. 10.1: Risk Management Training

Die Tab. 10.1 gibt einen Vorschlag zur Gestaltung der Reihenfolge in welcher die Risiko-Management-Begriffe trainiert werden können [13]). Darin werden die Kategorien: Konzepte (im wesentlichen in Session 1), Methoden (iwiS 2), Prozesse (iwiS 3), Metriken (iwiS 4) und Chancen/Risiko-Management (iwiS 5) abgedeckt. Die Sessions bezeichnen hierbei Trainingseinheiten, deren Inhalte sich auf die in der Tabelle genannten Begriffe beziehen. – Sie werden in den folgenden Abschnitten erläutert.

[13] Hall, Elaine M., Managing Risk, USA, 2001

Ohne auf die Details zum effizienten Gestalten von Kursen einzugehen sei auf zwei wesentliche Punkte für einen erfolgreichen Kurs (aus Teilnehmer- und Trainersicht) hingewiesen:

- Eine schon während des Kurses abgehaltene Kursreflexion hilft dem Trainer bei der inhaltlichen Themenauswahl und gibt den Teilnehmern die Möglichkeit der Mitgestaltung, was in erheblichem Maße zur Motivationssteigerung beiträgt.
- Einer der fatalsten Fehler ist ein „Motivation-Killing" durch einen unqualifizierten Trainer!

Während in den ersten 3 Stufen bzw. Reifegraden der Erfolgsfaktorengruppe: „Infrastructure" neue Rollen vergeben und Aktivitäten eingeführt wurden, muss daran anschließend überprüft werden, inwieweit die ursprünglichen Planungen auch in die Realität umgesetzt werden konnten. Hierfür werden in Audits mit Hilfe der aus der Qualitäts-Sicherung bekannten Verfahren die Rollen, Aktivitäten, Outputs, usw. anhand konkreter Projekte verifiziert. U.a. werden die Risiko-Management-Pläne auf:

- Vollständigkeit,
- Verständlichkeit,
- Detaillierungsgrad,
- Widerspruchsfreiheit,
- Anwendbarkeit,
- Zufriedenheit der Mitarbeiter,

hin überprüft ("**Verify Compliance**"). Erst nachdem auf diese Weise etwaige Unzulänglichkeiten beseitigt wurden kann ein Zyklus der Verbesserung unter Einbeziehung neuer Ideen und Sachverhalte in Erwägung gezogen werden.

Die Vorgehensweisen für den immer wiederkehrenden Verbesserungszyklus („**Improve Practice**") sind sehr vielfältig. Sie reichen von der Realisierung einzelner Verbesserungsvorschläge bis hin zur Infragestellung und Neugestaltung der gesamten Risiko-Management-Thematik. Die hierbei zum Einsatz kommenden Methoden werden am besten jenen des Qualitäts-Managements entliehen (vgl. hierzu auch die Beiträge zur Stufe 5 des CMM in Kapitel 8).

10.3.2 Erfolgsfaktor Mensch

Dieser Abschnitt widmet sich den Personen, deren Interessen, Einstellungen und Verhaltensweisen gegenüber einem Software-Projekt, das Risiko-Management-Methoden unterschiedlichen Reifegrades anwendet.

Wird eine Organisation zum ersten Mal mit dieser Thematik konfrontiert, werden zunächst die Probleme („**Problem**") in den Mittelpunkt der Diskussion gerückt. Es wird auf Erfahrungen aus vergangenen Software-Projekten Bezug genommen, ohne jedoch daraus strukturierte Methoden zur Risiko-Bewältigung abzuleiten. Dies ist kennzeichnend für die erste Stufe bzw. den ersten Reifegrad (vgl. Abb. 10.6).

Zur Erzielung von Fortschritten spielen in der ersten Phase externe Berater bzw. Mitarbeiter aus anderen Bereichen mit entsprechender Erfahrung eine zentrale Rolle. Zuallererst ist jedoch, wie bereits mehrfach erwähnt, ein Commitment des Senior-Managements eine notwendige Voraussetzung, ohne die Risiko-Management rein akademischer Natur ist und zudem zur Privatsache degradiert wird.

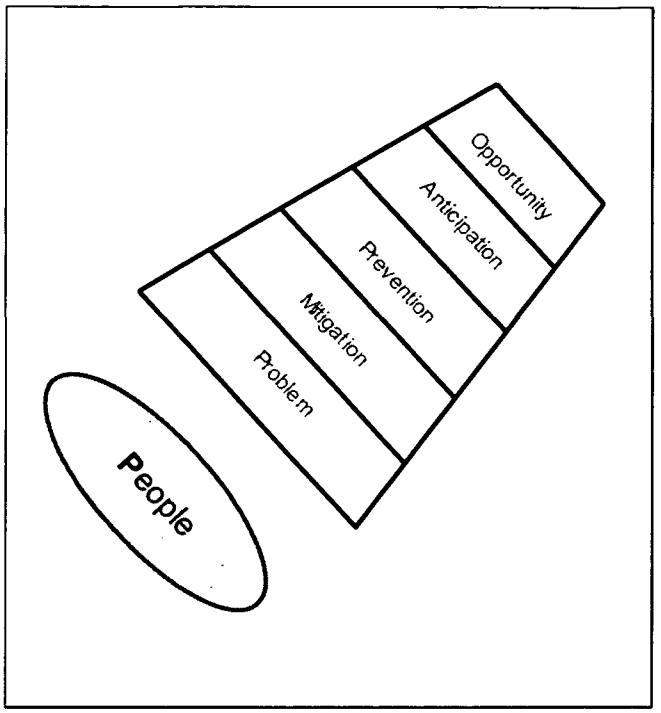


Abb. 10.6: People

Des weiteren sind noch folgende wichtige Rahmenbedingungen zu schaffen, die Teil der Erfolgsfaktorengruppe: „Infrastructure" sind:

- Bereitstellung des Budgets zur Verbesserung des Risiko-Managements
- Formulierung einer von allen verstandenen und akzeptierten Vision
- Zuweisung der für die Einführung von Risiko-Management wichtigen Rollen

Sind diese Voraussetzungen einmal verwirklicht, wird die Risiko-Thematik in Meetings und Seminaren erörtert, wobei zwecks Sicherstellung einer weiten Akzeptanz möglichst viele der betroffenen Mitarbeiter zu involvieren sind. Die Themenwahl für das erste Meeting darf nicht zu eng abgesteckt sein. Oftmals versuchen Mitarbeiter ihren aufgestauten Frust über das eine oder andere Ereignis in diesen Sitzungen kund zu tun, was als Katalysator zum erfolgreichen Erarbeiten der Thematik sehr hilfreich sein kann.

Charakteristisch für die Erfolgsfaktorengruppe „People" sind auch jene im 7-Disziplinen-Modell nach Exl (vgl. Abb. 10.3) genannten personenbezogenen Einflüsse. Aufgrund der weitreichenden Auswirkungen sind sie in jeder Reifegradstufe zu adressieren. Da die Risken in diesem Fall jedoch direkt mit einzelnen Personen in Verbindung zu bringen sind, bedarf es zu ihrer Thematisierung eines besonderen Einfühlungsvermögens.

Im Anschluss an die Problem-Visualisierung wird die momentane Projekt-Praxis mit ihren Schwächen und Stärken analysiert, um darauf aufbauend Pläne mit Maßnahmen zur Verbesserung abzuleiten. Nach der Festlegung der nächsten Schritte und eventuell weiteren Unterrichts-Einheiten werden die Vorhaben in die Tat umgesetzt.

Die Abb. 10.7 zeigt eine für diesen Einführungs-Prozess geeignete Organisations-Struktur. Während sich das Senior-Management mit Fragestellungen zur Vision und Finanzierung auseinandersetzt, ist das Steering-Board für die konkreten Zielsetzungen und die Umsetzung durch das Risiko-Management-Koordinations-Team (RM-Koordinations-Team) verantwortlich.

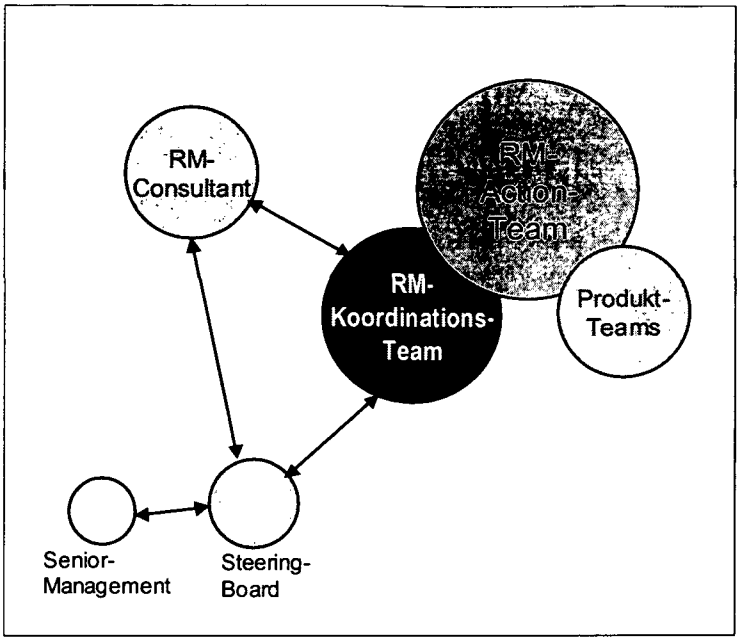


Abb. 10.7: Risiko-Management Einführung

In Abb. 10.7 symbolisiert die Größe der Kreise den Anteil der Risiko-Management-Aktivitäten an der gesamten Arbeitslast der jeweiligen Teams bzw. Personen. Der Abstand zwischen den Kreisen symbolisiert das Ausmaß der Interaktion zwischen den Teams bzw. Personen.

Das Team mit dem umfangreichsten Einblick in die durch die Risiko-Management Initiative gestarteten Aktivitäten ist das RM-Koordinations-Team. Es umfasst Personen aus möglichst allen durch die Aktivitäten betroffenen Bereiche. Es evaluiert den aktuellen Prozess, entwickelt den neuen Prozess, erarbeitet ein Metrik-Programm, usw. Da die Angehörigen dieses Teams weiterhin ihre zuvor ausgeführten Tätigkeiten weiterführen, muss ein gesondertes Team eingesetzt werden, das sich den Arbeiten zu 100% widmet – das RM-Action-Team. Die konkrete Implementierung in die tagtäglich ablaufenden Software-Entwicklungs-Prozesse wird von den Produkt-Verantwortlichen selbst wahrgenommen.

Bei Vorhandensein eines grundsätzlichen Verständnisses über Risiko-Management einerseits, und rudimentärer Risiko-Management-Aktivitäten innerhalb des Entwicklungs-Prozesses andererseits, kann in einer zweiten Stufe der Risiko-Management-Prozess in einem Assessment neu hinterfragt und verbessert werden. Die Fokussierung liegt dabei auf der Verringerung („**Mitigation**") der negativen Auswirkungen unerwünschter Ereignisse. Um dies zu erreichen werden Risiko-Kategorien erarbeitet und Checklisten vorbereitet, die im

- Risiko-Konsequenzen
- Risiko-Eintritts-Wahrscheinlichkeiten
- Zeitrahmen, innerhalb derer die Risken eintreten könnten
- Verantwortlichkeiten und Zuständigkeiten einzelner Risken

Ist das Risiko-Management bereits fester Bestandteil des Software-Entwicklungs-Prozesses kann mit einer eingehenden Analyse der Risken fortgefahren werden. Insbesondere wird nach Gründen und Zusammenhängen zwischen Risken, deren Ursachen, Verknüpfungen mit anderen Sachverhalten, usw. gesucht. Dazu werden alle möglichen Kreativitätstechniken und strukturierten Methoden angewandt. Das Ziel sind präventive Maßnahmen („**Prevention**") zur Vermeidung möglichst aller Risken. Charakterisierend für diese Entwicklungsstufe ist die weitgehende Einbindung betroffener Personen in Diskussionen zum Risiko-Management-Prozess, deren Methoden und Software-Tools. Es werden erste Anstrengungen unternommen die Risken zu quantifizieren um den Risiko-Management-Prozess weiter zu stabilisieren.

Zentrale Aufgabe der nächsten Stufe („**Anticipation**") ist die Quantifizierung aller relevanter Größen des Risiko-Management-Prozesses, bis hin zur Bestimmung des Return on Investment betreffend aller Risiko-Management-Aktivitäten. Dabei ist immer Wert auf die Verknüpfung mit den dafür verantwortlichen Rollen zu legen. Durch einen Vergleich der Plan-, Soll- und Istgrößen hinsichtlich des Software-Projekt-Fortschritts sollen Prognosen über das Projektende kalkulierbar sein. Die Metriken hierfür sind weitgehend identisch mit jenen aus dem Bereich des Qualitäts-Managements (z.b. aus dem CMM). Hierbei stehen jene Maße im Mittelpunkt die die Eintrittswahrscheinlichkeiten und die finanziellen Konsequenzen beziffern.

Bemerkung: Den Wert der Risiko-Quantifizierung kannte u.a. schon Sunzi (vor etwa 2500 Jahren) als er schrieb [14]): „Solange Du nur den Feind (entspräche dem Risiko; Anm. des Autors) genau kennst, brauchst Du den Ausgang von 1000 Schlachten (entspräche den Projekten; Anm. des Autors) nicht zu fürchten."

Als Krönung der Beherrschung des Risiko-Managements kann die bewusste Gestaltung von Chancen und Risken („**Opportunity**") bezeichnet werden. Ein gutes Beispiel hierfür liefern die durch Hedging-Strategien exakt kalkulierten Chancen und Risken im Derivativ-Segment des Wertpapierhandels. Aktuell unbekannte Determinanten werden als Chancen zukünftiger Geschäfte betrachtet. Obwohl sich kaum eine Organisation auf diesem Level der Risiko-Management-Implementierung befindet, sind sie durch die mittlerweile immer häufiger auf Fix-Preis-Basis gestalteten Verträge in eine Position gedrängt, wo gerade dieses Wissen über Gewinn oder Verlust entscheidet!

[14] Sunzi, Die Kunst des Krieges, zwischen 300 und 500 v.Chr., hrsg. von Clavell, James, München, 1988

10.3.3 Erfolgsfaktor Implementierung

Die Implementierung umfasst die Tätigkeiten die zur Einführung („**Establish the Initiative**") und Anwendung des Risiko-Managements notwendig sind. Wichtige Voraussetzungen für die Implementierung sind neben der Motivation der Beteiligten ein Risiko-Management-Plan („**Develop the Plan**"), Budget, Personal-Ressourcen, und die Bereitstellung von Methoden und Tools zur Durchführung der Aktivitäten (vgl. hierzu die Erfolgsfaktorengruppe: „Infrastructure").

Aufgrund der jeweils projektspezifischen Anforderungen muss gegebenenfalls der Risiko-Management-Plan entsprechend skaliert werden („**Tailor Standard Process**"). Wesentlich ist hierbei, dass er dem Kriterium der Kosten-Effizienz genügt, d.h. dass eine Balance zwischen den zu erwartenden Kosten im Eintrittsfall eines Schadens und den Kosten für Aufwände, die dies zu vermeiden suchen, gefunden wird.

Dabei wird von der Annahme ausgegangen, dass neben dem Risiko-Management-Prozess der eigentliche Entwicklungsprozess im Vordergrund steht. Dem gegenüber existieren jedoch auch andere Ansätze, wie z.b.: „Large-Scale Project Management is Risk Management" [15]). Dieser Ansatz basiert auf einer zentralen Stellung des Risiko-Management-Prozesses, um den herum der Software-Entwicklungs-Prozess aufgebaut ist. Eine Begründung für den neuen Stellenwert des Risiko-Managements kann aus den beiden Tatsachen abgeleitet werden, dass der Umfang der Software-Projekte kontinuierlich zunimmt und die Planbarkeit invers dazu abnimmt.

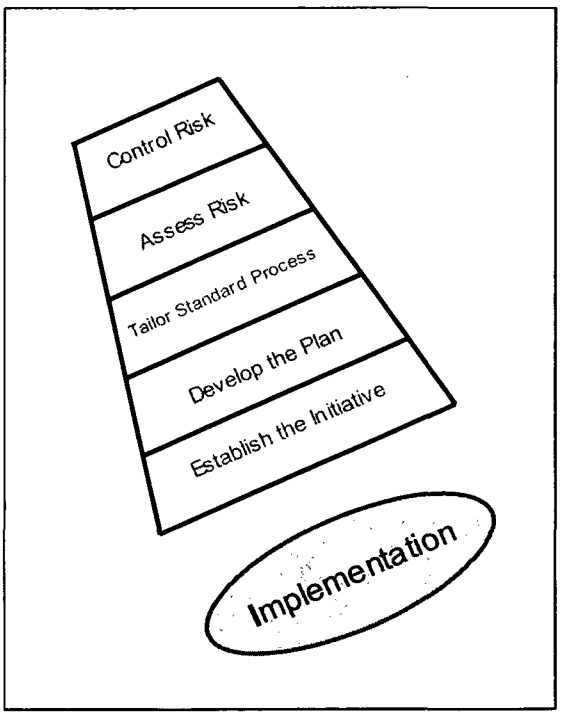


Abb. 10.8: Implementation

Ein anderes Modell, dem ebenfalls der Ansatz der zentralen Stellung des Risiko-Managements zu Grunde liegt, ist das Spiralmodell von B.W. Boehm. Dieses Modell basiert zudem auf einem iterativen Ansatz, wobei die periodische Risiko-Analyse Teil des Entwicklungs-Prozesses selbst ist [16]) (vgl. hierzu insbesondere die Ausführungen über das Spiralmodell in Kapitel 6). Damit ist dieser Prozess der derzeit einzige, der alle wichtigen Aspekte der Software-Entwicklung abdeckt.

[15] Charette, Robert N., Large-Scale Project Management is Risk Management, in: IEEE, July 1996, p. 110-117
[16] Feyhl, Achim W., Feyhl, Eckhardt, Management und Controlling von Softwareprojekten, Wiesbaden, 1996

Die in Abb. 10.8 dargestellten Reifegrade werden durch die Umsetzung der in den jeweils korrespondierenden Stufen der anderen Erfolgsfaktoren genannten Aktivitäten mitbestimmt. Darüber hinaus wird der Erfolg der Implementierung direkt durch die Personen bestimmt, die die einzelnen Aktivitäten des Risiko-Management-Prozesses ausführen.

Aufgrund der Erfahrungen die durch die Abhaltung von SPR-Assessments (als Beispiel für eine Realisierungsvariante der Stufe: „**Assess Risk**") gewonnen wurden, muss davon ausgegangen werden, dass die in der Praxis durchschnittlich erzielbaren Fortschritte erst über Jahre hinweg beurteilt werden können. Zwar umfassen SPR-Assessments alle Aspekte des Software-Entwicklungs-Prozesses, doch kann mit ziemlicher Sicherheit davon ausgegangen werden, dass die Risiko-Thematik als Teil-Aspekt in der Praxis (noch) keine überproportionale Priorität genießt. Interessant ist aber dennoch, dass die am höchsten entwickelten Software-Organisationen um etwa den Faktor 5 schneller lernen als die am wenigsten weit entwickelten Organisationen [17]).

Als Hilfestellung zur praktischen Umsetzung des Themas: „Implementation" ist das Buch: „Assessment and Control of Software-Risks" von Capers Jones [18]) zu empfehlen. Als Vorlage zur Gliederung des Buchinhalts diente ihm ein medizinisches Werk über ansteckende Krankheiten. Ohne es ausdrücklich zu erwähnen kommt es dem Pattern-Gedanken (vgl. hierzu Kapitel 11) sehr nahe! Die Tab. 10.2 gibt hierbei einen Überblick der diskutierten Patterns, während in Tab. 10.3 deren Gliederung dargestellt ist – anhand dessen jedes Risiko in strukturierter Weise diskutiert wird. Die beiden Tabellen können gegebenenfalls auch als Checkliste dienen. Im Vergleich zu den am Beginn dieses Kapitels dargestellten Risiko-Arten sind die bei Capers Jones erwähnten Risken nicht gegliedert (die inhaltliche Beschreibung jedoch schon, weshalb die Analogie zum Pattern-Gedanken sehr groß ist).

Der Erfolg der Implementierung hängt jedoch nicht zuletzt auch von der persönlichen Einstellung jedes einzelnen in der Organisation gegenüber Risken ab. Dieses kann von risikoavers über risikoneutral bis hin zu einer Risiko suchenden Einstellung variieren. Im letzteren Fall sollte der Ansatz zum Umgang mit Risken (z.b. die Firmen- oder Organisations-Policy) danach ausgerichtet werden, jene Chancen in den Mittelpunkt zu stellen, die ein kalkuliertes Risiko aufwiegen können („**Control Risk**").

[17] Jones, Capers, Assessment and Control of Software Risks, USA, 1994
[18] Jones, Capers, Assessment and Control of Software Risks, USA, 1994

The Most Common Software Risks	Inadequate Software Project Value Analysis
The Most Serious Software Risks	Inadequate Tools and Methods (Project Management)
Ambiguous Productivity or Quality Improvem. Goals	Inadequate Tools and Methods (Quality Assurance)
Artificial Maturity Levels	Inadequate Tools and Methods (Software Engineering)
Cancelled Projects	Inadequ. Tools and Methods (Technical Documentation)
Corporate Politics	Lack of Reusable Architecture
Cost Overruns	Lack of Reusable Code
Creeping User Requirements	Lack of Reusable Data
Crowded Office Conditions	Lack of Reusable Designs (Blueprints)
Error-Prone Modules	Lack of Reusable Documentation
Excessive Paperwork	Lack of Reusable Estimates (Templates)
Excessive Schedule Pressure	Lack of Reusable Human Interfaces
Excessive Time to Market	Lack of Reusable Project Plans
False Productivity Claims	Lack of Reusable Requirements
Friction between Clients and Software Contractors	Lack of Reusable Test-Plans, -Cases and -Data
Friction b. Software Managem. and Senior Executives	Lack of Specialisation
High Maintenance Costs	Long Service Life of Obsolete Systems
Inaccurate Cost Estimating	Low Productivity
Inaccurate Metrics	Low Quality
Inaccurate Quality Estimating	Low Status of Software Personnel and Management
Inaccurate Sizing of Deliverables	Low User Satisfaction
Inadequate Assessments	Malpractice (Project Management)
Inadequate Compensation Plans	Malpractice (Technical Staff)
Inadequ. Configuration Control & Project Repositories	Missed Schedules
Inadequate Curricula (Software Engineer)	Partial Life-Cycle Definitions
Inadequate Curricula (Software Management)	Poor Organization Structures
Inadequate Measurement	Poor Technology Investments
Inadequate Package Acquisition Methods	Severe Layoffs and Cutbacks of Staff
Inadequate Research and Reference Facilities	Short-Range Improvement Planning
Inadequate Software Policies and Standards	Silver Bullet Syndrome
Inadequate Software Project Risk Analysis	Slow Technology Transfer

Tab. 10.2: Software-Risk-Patterns

1	Definition	11	Product support
2	Severity	12	Consulting support
3	Frequency	13	Education support
4	Occurrence	14	Publication support
5	Susceptibility and resistance	15	Periodical support
6	Root causes	16	Standards support
7	Associated problems	17	Professional associations
8	Cost impact	18	Effectiveness of known therapies
9	Methods of prevention	19	Costs of known therapies
10	Methods of Control	20	Long-range prognosis

Tab. 10.3: Risk-Patterns-Notation

10.4 Prozess des Risiko-Managements

Im Gegensatz zu den anderen Erfolgsfaktorengruppen handelt es sich bei „Process" um ein zusammenhängendes Gebilde, das zwar ebenfalls stufenweise zu entwickeln ist, wobei aber den einzelnen Stufen („Identify Risk", usw.) nicht unmittelbar eine Entwicklungsstufe auf einer Skala der Fähigkeit (= Capability) zugeordnet werden kann. Stattdessen ist für das Erlangen eines Fortschritts die Beherrschung mehrerer Stufen bzw. Teil-Prozesse („Identify Risk", usw.) gleichzeitig notwendig. Dies wird durch die waagrechten Linien in der Abb. 10.9 verdeutlicht (vgl. hierzu insbesondere die Ausführungen zur Abb. 10.4).

Anders als in der hier dargestellten Gesamtprozess-Gliederung ist in der Literatur sehr häufig eine Strukturierung nach folgenden Prozess-Schritten zu finden:

- Risiko-Identifikation
- Risiko-Analyse
- Risiko-Gestaltung
- Risiko-Controlling

Im weiteren Verlauf wird an entsprechender Stelle auf diese Gliederung Bezug genommen.

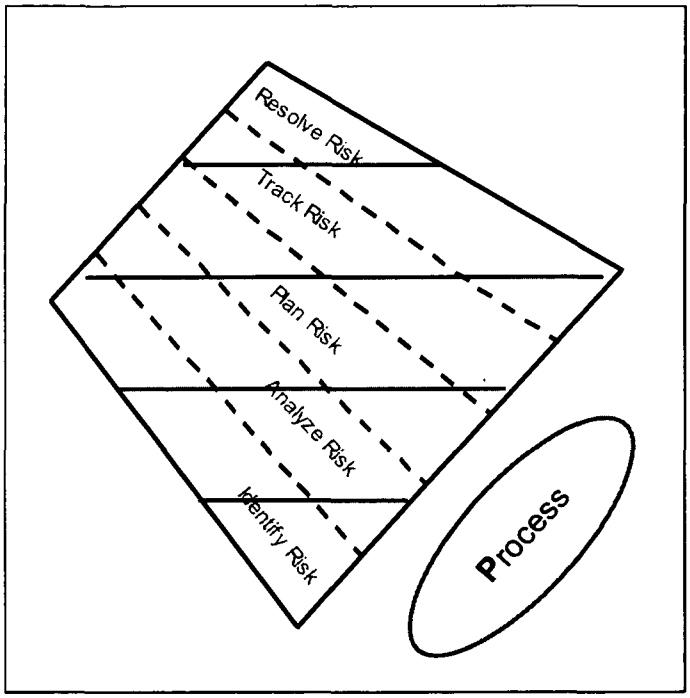


Abb. 10.9: Process

Eine Übersicht über die Verknüpfung der Prozess-Schritte wird in Abb. 10.11 gegeben, die gegenüber den Ausführungen von E.M. Hall [19]) hinsichtlich des Schritts „Resolve Risk" leicht modifiziert wurde. In dieser Abbildung ist der Prozess-Pfad durch die fette Linie symbolisiert. Die strichlierten, gepunkteten und punkt-strichlierten Pfade bilden den Regelkreismechanismus innerhalb des Risiko-Management-Prozesses (dadurch wird das während des Software-Entwicklungs-Prozesses kontinuierlich ablaufende Risiko-Controlling symbolisiert).

In der Abb. 10.10 wird die Notation eines Prozess-Schritts anhand der IDEF0 Prozess-Notation gezeigt. Hierbei charakterisiert „Control" die Nebenbedingungen für die Ausführung des Prozess-Schritts. „Mechanism" charakterisiert die im Prozess-Schritt zur Anwendung

[19] Hall, Elaine M., Managing Risk, USA, 2001

kommenden Methoden. „Input" kennzeichnet die Prozess-Schritt-Kriterien am Eingang, und „Output" jene nach erfolgreicher Prozess-Transformation (= Prozess-Schritt-Abarbeitung).

In der Abb. 10.11 werden die durch „Control" und „Mechanism" charakterisierten Nebenbedingen (gewissermaßen die Infrastruktur) bzw. Methoden durch dünne durchgehende Linien dargestellt. Unter „Project Resources" werden Kosten, Zeit und Personal-Ressourcen subsummiert. Unter „Project Requirements" werden die organisatorischen Standards und vertraglichen Projekt-Verpflichtungen subsummiert. Der „Risk Management Plan" subsummiert Angaben zur Prozess-Gestaltung und Rollen-Verteilung auf die Personen der Organisation.

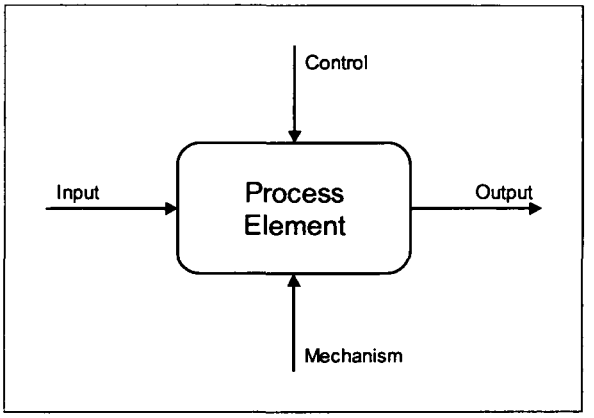


Abb. 10.10: Prozess-Element

Eine detaillierte Beschreibung der Abb. 10.11 erfolgt im Anschluss daran.

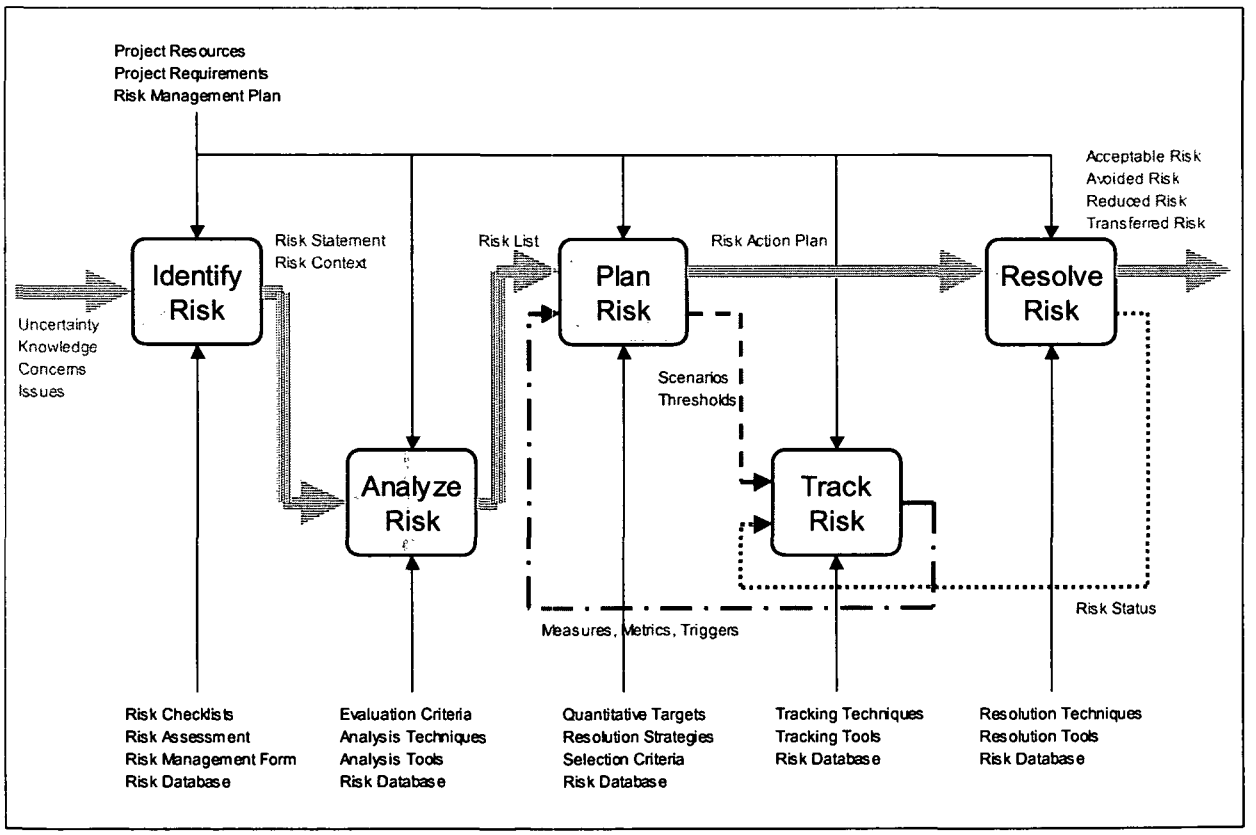


Abb. 10.11: Risiko-Management-Prozess

10.4.1 Risiko-Identifikation

Wesentliches Merkmal eines Risikos ist der Informationsmangel über zukünftige Ereignisse. Wie schon früher angedeutet darf Informationsmangel a priori nicht als negativ interpretiert werden. Ohne Risken würde jede Unternehmung zum gleichen Erfolg führen, die Selektion im darwinistischen Sinne würde nicht mehr wirken, und die Entropie gegen unendlich streben.

Obwohl vieles zu Beginn der Software-Entwicklung noch unbekannt ist („Uncertainty"), gibt es aufgrund der Erfahrung oder vertraglichen Gestaltung schon von Beginn an definierte Elemente („Knowledge"). - Und wie immer und überall auch Belange die teilweise bekannt sind und Anlass zur Sorge geben („Concerns") oder innerhalb der Organisation stets zur Diskussion stehen („Issues").

Das Ergebnis der Identifikations-Phase („**Identify Risk**") ist eine Liste von Risken („Risk Statement") die durch nähere Angaben über Ereignisse, Bedingungen, Annahmen, Umstände, in Beziehung stehende andere Themen, usw. näher beschrieben sind („Risk Context"). Als Methoden für die Risiko-Identifikation eignen sich bei Software-Projekten insbesondere („Risk Assessment"):

- Checklisten: Bei regelmäßiger Pflege der Checklisten eignen sie sich hervorragend zur Risiko-Identifikation ohne die Projektsituation regelmäßig erneut zu analysieren („Risk Checklists").
- Experten-Befragungen: 90% der Risken sind einem erfahrenen Projektleiter zumeist bekannt.
- Mitarbeiter-Befragung: Nur in den seltensten Fällen überraschen plötzlich eintretende Krisensituationen ein Software-Projekt. Aufgrund der unterschiedlichsten Erfahrungen der Projektteammitglieder ist ihre rechtzeitige Beteiligung empfehlenswert.
- Vertragsanalyse: Hierbei empfiehlt sich besonders die Verwendung von Musterverträgen und ein Review durch die juristische Abteilung.

Als Rahmen für die Identifikationsphase muss eine Prozess-Beschreibung existieren, die die Anwendung der einzelnen Methoden regelt. Dies ist eine wichtige Orientierungshilfe für die Teilnehmer an Risiko-Identifikations-Meetings. Darüber hinausgehend sollte der Prozess auch Bezug nehmen zu:

- Selektion und Training des die Meetings leitenden Personen
- Auswahlkriterien für die Teilnehmer an Meetings (entsprechend deren Rollen)
- logistische Angaben (Zeitvorgaben, Hilfsmaterial wie z.b. Kärtchen, usw.)
- Vorgaben zur Risiko-Dokumentation (z.b. vorgefertigte Formulare („Risk Management Form"))

Durch den Einsatz von Datenbanken kann eine Wissensbasis aufgebaut werden („Risk Database"), deren Umfang mit jedem Projekt wächst. Das Datenbank-Schema umfasst hierbei sämtliche während des Prozesses gemachten Angaben zum jeweiligen Risiko, wie:

- Risiko-Bezeichnung
- Name desjenigen, der dieses Risiko identifiziert hat
- Risiko-Kategorie
- Risiko-Eintritts-Wahrscheinlichkeit
- Angaben zu finanziellen Größen
- Verlauf der Priorisierung
- Schwellwerte und Triggermechanismen
- Rahmeninformationen (z.b. Projektname, Datum, usw.)

Die Datenbank muss darüber hinaus Bestandteil eines umfassenden Software-Entwicklungs-Prozesses sein, da sie nicht nur Informationen bereitstellt, sondern auch mit den Risiko relevanten Daten der jeweiligen Phase des Entwicklungs-Prozesses versorgt werden muss.

Die Risiko-Identifikations-Phase und jene der Analyse (siehe nächster Abschnitt) bilden zusammen die Risiko-Assessment-Phase.

10.4.2 Risiko-Analyse

Die Risiko-Analyse („**Analyze Risk**") soll die während der Identifikationsphase erkannten Risken bewerten und priorisieren. Dadurch sollen Indikatoren gefunden werden, die das Eintreten eines unerwünschten Ereignisses möglichst früh anzeigen. Der Prozess der Analyse sollte schon im Vorfeld definiert werden und gestaltet sich im Detail nach folgendem Schema:

Zunächst werden die aus der Identifikations-Phase stammenden Risken nach charakteristischen Merkmalen in Kategorien gruppiert. Wertvolle Zusammenhänge zwischen den Risken dürfen dabei nicht verloren gehen. Danach werden die Risiko-Treiber als hauptverantwortliche Determinanten für die einzelnen Risken identifiziert. Dies entspricht einer inhaltlichen Ergänzung der jeweiligen Risiko-Kontexte. Parallel zu den Treibern erfolgt in diesem Schritt auch die Bestimmung der Risiko-Ursache(n) – soweit möglich (zusammengefasst in der „Risk List").

Grundsätzlich gibt es für die Risiko-Analyse zwei Ansätze [20]):

1. Analyse jedes einzelnen Risikos (standardmäßiges Vorgehen)
2. Analyse verschiedener zukünftiger Szenarien (aufwendigere Variante)

[20] Hull, John C., Options, Futures, and other Derivatives, Toronto, 1997

Hierbei wird jedes Risiko hinsichtlich Ursachen, Folgen und unterschiedlicher Verkettungen analysiert (einzeln oder als Kombination in Form von Szenarien). Für die Analyse des einzelnen Risikos stehen u.a. folgende Methoden zur Verfügung („Analysis Techniques"):

- Kausal-Analyse:
  Dahinter verbergen sich wiederum unterschiedliche Zugänge des schrittweisen Erarbeitens der Risiko-Ursache(n). Als Repräsentanten seien das Fischgräten-Diagramm und die Methode des mehrmaligen „Warum" genannt.

- Entscheidungs-Analyse:
  Dieses Modell stellt die reale Welt als Abfolge von Entscheidungen dar. Die Basiselemente sind: Entscheidungen, unvorhersehbare Ereignisse und die Werte der Ereignis-Ausgänge. Zur visuellen Darstellung werden in der Regel Entscheidungs-bäume verwendet.
  Diese sind u.a. auch ein unverzichtbares Instrument zur Analyse von Finanzderivaten – aus dessen Umfeld die Prinzipdarstellung (vgl. Abb. 10.12) entlehnt wurde.

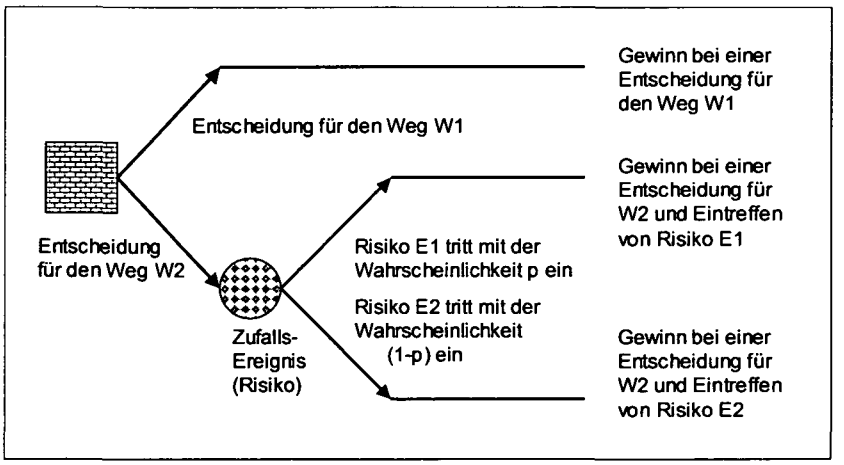


Abb. 10.12: Prinzip eines Entscheidungsbaums

- Gap-Analyse:
  Kennzeichnend für die Gap-Analyse sind Fragestellungen zum selben Thema an mehrere Teams bzw. Personen, wobei der (quantitative) Unterschied hinsichtlich der Antworten das gesuchte Ergebnis ist. Auf diese Weise kann z.b. sehr anschaulich der Unterschied zwischen einer persönlichen und einer Gruppenbewertung dargestellt werden. Graphisch geschieht dies meist in Form eines Balken- oder Radar-Diagramms.

- Pareto-Analyse:
  Ähnlich wie die Gap-Analyse sollte die Pareto-Analyse eigentlich als Prinzip aufgefasst werden. Das Prinzip, das hinter der Pareto-Analyse steht ist der häufig anzutreffende Sachverhalt, dass 80% der Probleme auf 20% aller möglichen Ursachen zurückzuführen sind. Durch eine Fokussierung auf die wesentlichen Ursachen wird vor allem die Effizienz und dadurch der Ressourcen-Einsatz optimiert.

- Sensitivitäts-Analyse:
  Darin werden alle auf einen Prozess einwirkenden Parameter in ihrer Größe variiert und mit dem sich daraus ergebenden Prozess-Output in Relation gesetzt. Oft genügt es die Input-Parameter auf ihre Maximalwerte zu verändern. Dabei ist es wichtig, dass immer nur ein Parameter verändert wird, während alle anderen mit ihrem realen Wert auf den Prozess Einfluss nehmen.

Das Ergebnis dieser Analyse ist eine Charakterisierung der Risken durch ihre

- Eintrittswahrscheinlichkeit (durch %-Angabe oder Ordinalskala), den
- Konsequenzen bei Risiko-Eintritt (projektspezifisch), und dem
- Zeitrahmen, innerhalb dessen auf das Risiko reagiert werden muss.

In einem letzten Schritt werden die im Detail analysierten Risken gereiht oder mit einem Kriterium verglichen und anschließend gereiht („Risk List"). Die Priorisierung der Risken erfordert dabei einen auf alle Ereignisse anwendbaren Maßstab („Evaluation Criteria"). Die klassische Bewertung der Risken in Geldeinheiten ist bei Software-Projekten im allgemeinen schwer anwendbar und daher nur von theoretischer Bedeutung. Besser eignet sich eine Punktebewertung (z.b. Ampelbewertung), die eine relative Rangordnung der Risken zum Ziel hat. Eine alternative Darstellung dazu ist das 4-Felder Schema, das Risken nach der Eintrittswahrscheinlichkeit und der Bedeutung für den Projekterfolg einteilt. Dies gestattet die in der Praxis übliche stufenweise und kostenorientierte Verminderung der Risken-Anzahl für die anschließende Risiko-Planungs-Phase. Sollte aus Gründen einer besseren Übersicht eine Zusammenfassung von Risken erfolgen, ist auf eine entsprechend sinnvolle logische Verknüpfung zu achten („UND", „ODER", „EXKLUSIV-ODER", usw.).

10.4.3 Risiko-Planung bzw. Risiko-Gestaltung

Die Risiko-Gestaltung („**Plan Risk**") stellt den eigentlichen Zweck des Risiko-Managements dar [21]). Ausgehend von einer Diskussion der Risiko-Szenarien werden unter Berücksichtigung der „Quantitative Targets" unterschiedliche Alternativen zur Risiko-Gestaltung erarbeitet. Parallel dazu werden jene Kriterien („Selection Criteria") festgelegt anhand denen die Risiko-Auflösungs-Strategie ausgewählt wird („Resolution Strategies"). Dabei sollten alle Projekte des Projekt-Programms Berücksichtigung finden, da durch eine geeignete Diversifikation das Gesamtrisiko minimierbar ist (vgl. Portfolio-Selection-Modell in Kapitel 3).

In einem Risiko-Aktions-Plan („Risk Action Plan") werden bezogen auf die einzelnen Szenarien die detaillierten Maßnahmen dokumentiert (vorzugsweise in einer Datenbank). Gekoppelt mit den Maßnahmen sollten auch Metriken („Metrics") mit dazugehörigen Schwellwerten („Thresholds") hinterlegt werden. Dadurch wird ein Trigger festgelegt bei dessen Aktivierung

[21] Patzak, Gerold, Rattay, Günter, Projekt Management, Wien, 1998

Maßnahmen zur Risiko-Auflösung ergriffen werden. Bezogen auf den Risiko-Management-Prozess ist der Trigger-Mechanismus („Tracking Techniques" und „Tracking Tools") der Controlling-Phase („Track Risk") zugeordnet. Ausgerichtet werden die Schwellwerte anhand der quantitativen Ziele („Quantitative Targets") bzw. Benchmarks.

Die Basis-Strategien zur **Risiko-Auflösung** sind:

- Risiko-Eliminierung („Avoided Risk"):
  Diese Strategie geht einher mit der Eliminierung von Chancen. In der Praxis bedeutet dies meist die Ablehnung von Aufträgen oder ein Angebot zu überhöhten Preisen.
- Risiko-Verminderung („Reduced Risk"):
  Ein wichtiges Instrument hierfür sind regelmäßige Projektdurchsprachen, die auf den zuvor erwähnten Analysen beruhen. In der Software-Entwicklung wird das Risiko z.b. durch die Verwendung bereits bekannter Entwicklungs-Tools, einer Prototyp-Entwicklung, eines iterativen Vorgehens oder Simulationen wesentlich verringert.
- Risiko-Überwälzung („Transferred Risk"):
  Dies ist gleichbedeutend mit dem Kauf zusätzlicher Sicherheit. Als Strategien stehen die entsprechende Vertragsgestaltung oder die Übertragung auf professionelle Risikoträger (Versicherung, staatlich unterstützte Exportkreditversicherung, u.a.) zur Auswahl.
- Risiko-Akzeptanz („Acceptable Risk"):
  Abgesehen von der unerkannten bzw. ungewollten Übernahme von Risken (z.b. bei zu geringer Marktmacht) darf nicht darüber hinweggesehen werden, dass Risken von Unternehmungen bewusst übernommen werden. Diese sind dann Teil des nach außen getragenen Images und somit inhärenter Teil der Kernkompetenzen einer Firma.

10.4.4 Risiko-Verfolgung bzw. Risiko-Controlling

Das Risiko-Controlling („**Track Risk**") erfüllt die Aufgabe eines Reglers innerhalb des Risiko-Management-Prozesses. Über regelmäßige und ereignisgesteuerte Projekt-Besprechungen werden die Anliegen, die aus dem Prozess resultieren, der Projektleitung mitgeteilt, die die Verantwortung für die gesetzten Maßnahmen trägt. Des weiteren liegt ein Teil der Risiko-Prozess-Verantwortung bei jenen die für die Qualitätssicherung zuständig sind. Die hauptsächliche Tätigkeit in diesem Prozess-Schritt („Track Risk") ist jedoch die Verwaltung von Risiko-Metriken („Risk Metrics") und den Aktivitäten die daraus abgeleitet werden. Über die im Software-Entwicklungs-Prozess eingesetzten Tools werden auch jene Maße („Measures") zur Verfügung gestellt, die für den Risiko-Management-Prozess von Relevanz sind.

Typische Risiko-Metriken sind:

- Anzahl der identifizierten Projekt-Risken
- Anzahl der identifizierten Projekt-Risken bezogen auf diverse Risiko-Kategorien
- Risiko-Maß (= Risk Exposure (RE)) als Produkt der Eintritts-Wahrscheinlichkeit und dem Schadensausmaß bei Eintritt
- Risk-Leverage (RL) definiert als (RE_{before} – RE_{after})/(Risk resolution cost) wobei RE_{before} das Risiko-Maß vor dem Einleiten Risiko minimierender Aktivitäten, RE_{after} das Risiko-Maß nach dem Einleiten Risiko minimierender Aktivitäten, und Risk resolution cost die Kosten der Risiko minimierenden Aktivitäten sind
- Risiko-Schwellwert kennzeichnet den Wert einer Metrik bei dessen Über- oder Unterschreiten RM-Maßnahmen aktiviert werden
- Risiko-Management-Index definiert als die (Summe aller REs) / (Gesamt-Projektkosten)
- Return on Investment als Verhältnis von [Summe von (RE_{before} – RE_{after}) über alle Risken] zu [den Gesamt-Kosten des Risiko-Management-Prozesses]

Basierend auf diesen Kenngrößen werden Trigger gesetzt, gelöscht oder vorübergehend deaktiviert. Diese starten, beenden oder suspendieren Aktionen entsprechend dem Risiko-Aktions-Plan. Hinsichtlich der Aktivierung eines Triggers gibt es grundsätzlich 2 Varianten:

- Über- oder Unterschreiten von Schwellwerten (z.b. Metriken, verstrichene Zeit, usw.)
- Ereignisse, die mit einem Trigger gekoppelt sind (Management-Meetings, regelmäßige Reports, Projekt-Reviews, usw.)

Im Falle schwerwiegender Risken empfiehlt sich zusätzlich das Überwachen von Szenarien.

10.4.5 Risiko-Auflösung

Die Phase der Risiko-Auflösung („**Resolve Risk**“) erfordert ein kreatives Vorgehen welches die Lösung der übriggebliebenen Risken als Herausforderung betrachtet. Die grundsätzlichen Methoden wurden bereits im Abschnitt: Risiko-Planung bzw. Risiko-Gestaltung aufgezählt („Resolution Techniques“).

Darüber hinaus sollte durch eine enge Kooperation aller Beteiligten (auch unter Einbeziehung des Kunden) eine möglichst ausgewogene Verteilung der Risken-, aber auch Chancen stattfinden. Eine einseitige Benachteiligung sollte zugunsten einer Win-Win-Situation vermieden werden. Das so auf gegenseitiges Vertrauen aufgebaute Projekt minimiert nicht nur die Risken, sonder schafft auch ein Klima, in dem Höchstleistungen erst ermöglicht werden.

11. Patterns

Wie in den vorhergehenden Kapiteln dargestellt, umfasst erfolgreiches Projekt-, Produkt- und Programm-Management die Beherrschung einer Vielzahl an Fähigkeiten, die zwecks Know-how - Transfer dokumentiert und archiviert werden sollten, um sie auch anderen Personen der betreffenden Organisation verfügbar zu machen. Dies ist eine Herausforderung, der durch das Schema des Musters begegnet wird. Diese zunächst in der Architektur angewandte Form der Wissensrepräsentation wird im folgenden auf das Gebiet des Projekt-, Produkt- und Programm-Managements angewandt.

11.1 Philosophische Grundgedanken

„Patterns", zu Deutsch Muster – eine Modeerscheinung oder eine zukunftsweisende Methode zur Gestaltung von Software-Entwicklungs-Organisationen?

Wie sehr oft in jenen Wissenschaften, die sich der Repräsentation von Wissen widmen, kam die Idee zu Patterns aus einem der Informatik völlig fremden Zweig – der Architektur. Christopher Alexander lieferte dazu die Grundlage. In seinem ersten Werk „Notes on the Synthesis of Form" [1]) behandelte er die Ästhetik und Nützlichkeit von Bauwerken. Sein besonderes Interesse galt dabei dem Bauwerk als Ganzes. Mit seinen Worten [2]): „Design is often thought of as a process of synthesis, a process of putting together things, a process of combination. But it is impossible to form anything which has the character of nature by adding preformed parts." Diese Aussage impliziert, dass alle Teile eines Bauwerks miteinander in Beziehung stehen. Oder anders formuliert - muss jedes dem Bauwerk neu hinzugefügte Artefakt auf alle bereits im Bauwerk integrierten Teile abgestimmt werden.

Wendet man diesen Grundgedanken auf Software-Organisationen an, folgt unmittelbar eine Schwerpunktsetzung auf die zwischenmenschliche Kommunikation aller am Entwicklungs-Prozess beteiligten Personen. Im Gegensatz zur Bewältigung einer Aufgabe durch eine Einzelperson spielen für die Beurteilung der Gesamtleistung einer Gruppe auch die Beziehungen untereinander eine wesentliche Rolle. Vervollständigt wird die ganzheitliche Betrachtung durch die Einbeziehung der Werkzeuge (z.b. Computer, Arbeitsmethodiken, usw.) und die durch sie geschaffenen Randbedingungen. Dies (die zwischenmenschliche Kommunikation im Team) ist ein Beispiel für einen Einflussfaktor, der in den Prozess-Beschreibungen zur Softwareentwicklung derzeit kaum Berücksichtigung findet.

---

[1] Alexander, Christopher, Notes on the Synthesis of Form, Harvard University Press, 1964
[2] Alexander, Christopher, The Timeless Way of Building, New York, 1979

11.1.1 Der Weg von der prozeduralen Programmierung zu Mustern

Bevor der Eindruck entsteht, dass die Idee der Muster eine logische Konsequenz der Objekt-Strukturierung ist, soll festgehalten werden, dass in der Software-Technologie der Begriff des Musters zwar chronologisch nach dem des Objekts auftritt, jedoch die Begriffe ansonsten nur insofern in Beziehung stehen, als sie sich gegenseitig ergänzen. Während durch das Objekt-Modell insbesondere die Teile eines Systems und deren Beziehungen untereinander formal gut beschreibbar sind, ermöglicht ein Muster-Modell bzw. eine Pattern-Language eine die wesentlichen Systemmerkmale herausstreichende Beschreibung (vgl. hierzu insbesondere Abschnitt 11.2.7).

In der Software-Technologie lag der Fokus lange Zeit auf der Formulierung von Algorithmen, die den Lösungsweg eines Problems in formaler Sprache (z.b. einer Programmiersprache) beschreiben. Dies war die Zeit in der die **prozedurale Programmierung** (Vertreter dieser Programmiersprachen sind z.b. Ada, BASIC, C, COBOL, Fortran, Modula-2, Pascal) hoch im Kurs stand. Mit der Zunahme an Code-Umfang und der damit einhergehenden Unübersichtlichkeit verlor sie jedoch an Bedeutung – u.a. auch dadurch, dass sich das Einsatzgebiet von der wissenschaftlichen Anwendung überwiegend auf die kommerzielle verschob. Verschiedene Überlegungen, die vor allem aus dem Wissenschaftszweig der Bionik kamen, führten zur Theorie der Objekte. Es wurden Mechanismen (z.b. Vererbung) implementiert, die wie andere Eigenschaften **objektorientierter Sprachen** (z.b. C++ oder JAVA) die Voraussetzung zur Bewältigung einer wachsenden Menge an Anforderungen erst bereit stellen. Eine weitere hinzukommende Anforderung ergibt sich aus der Tatsache, dass die Funktionalität der Software zu Beginn des Projekts meist nicht vollständig gegeben ist [3]). Somit resultieren als generelle Anforderungen an die Software neben der hohen Stabilität auch die Flexibilität gegenüber dem Einbau neuer Requirements (= Leistungsmerkmale).

Chronologisch nachfolgend, jedoch nicht unmittelbar auf die Objekt-Thematik aufbauend, begann sich die Pattern-Thematik zu etablieren. Die in diesem Zusammenhang angesprochenen Patterns sind exemplarische Algorithmen oder Codefragmente – im Gegensatz zu den in diesem Kapitel dargestellten, vorwiegend die Organisationsgestaltung betreffenden, Patterns. Bevor der Begriff „**Pattern**" weiter erläutert wird folgen einige charakterisierende Aussagen von James O. Coplien zu diesem Begriff [4]):
„... patterns are constrained by the rules of no single paradigm, which leaves them free to describe the incredibly rich structures of increasingly complex software systems. ... If we really take seriously the term "system" in "software system", we must consider facets of software development that range from technological to humanistic. ... Patterns are mostly about people and much less about houses, software, or design methods." – An anderer Stelle über ein Pattern das die Kriterien für einfach zu verstehenden Code beschreibt: „It is an intensely human pattern, woven together with words like „confidently", „stress", „secure", „culture", and

[3] Hickersberger, Arnold, Der Weg zur objektorientierten Software, Heidelberg, 1994.

„understand". It is intuitive because it speaks to our common human sense, but it is worth committing to literature because its precepts are so often ignored."

11.1.2 Anthropologie und die Grundlagen moderner Organisationen

Eine Wissenschaft, die sich schon sehr früh des Begriffs „Pattern" (im hier gebrauchten Sinne) bediente ist die Anthropologie – sie handelt vom Menschen uns seiner Entwicklung. In einem von A. L. Kroeber 1948 veröffentlichten Buch mit dem Titel: „Anthropology: Culture, Patterns, and Process" [5]) klassifiziert der Autor drei unterschiedliche Patterns (zu Deutsch: Muster), die das Verhalten einer Gemeinschaft bestimmen. Diese sind:

- Universale Muster: Allen Menschen gemeinsame Muster, die ähnlich der Instinkte von Tieren dem simplen Überleben dienen.
- Kulturelle Muster: Verhaltenweisen, die auf gemeinsamen historischen Überlieferungen beruhen.
- Gruppendynamische Muster: Verhaltensweisen, die sich aufgrund der Rollenverteilung in Gemeinschaften ergeben.

Ebenso motiviert - wie Charles Darwin, der die Evolution der organischen Lebensformen auf den allgegenwärtigen Überlebenskampf zurückführt, oder Stephen Hawking, der nach einer die Quantenfeldtheorie und die Allgemeine Relativitätstheorie vereinheitlichten, den gesamten Kosmos umfassende Formel sucht [6]) – sind auch die Absichten von Ch. Alexander zu interpretieren. So versuchte er u.a. auch in den Mustern von Teppichen eine universale Ästhetik zu entdecken. In einem Aufsatz von J.O. Coplien [7]) werden einige Vorstellungen Alexanders in dem kürzlich (im Jahr 2003) erschienenen Buchband [8]) bereits vorweg genommen:

Unter dem im Mittelpunkt stehenden Begriff „Nature of Order" versteht Alexander eine vereinheitlichte Theorie, die zu erklären versucht, wie sich „Things" [= Dinge, ganz allgemein] entwickeln und wachsen: „**Wholeness** (and life) emerges as a result of naturally occurring processes which are based on loci known as **Centers**." Abstrakt interpretiert handelt es sich dabei um eine philosophische Aussage, die im Zusammenhang mit dem Wachstum von zivilisatorischen Gebilden, beobachtete Muster auf eine Metaebene hebt. Nach Alexander „hat alles Wholeness" – ein abgegrenztes System mit einem bestimmten Zweck, und mit einem identifizierbaren Zentrum. – Mit den Worten J.O. Copliens:

---

[4] Coplien, James O., On the Nature of The Nature of Order, http://www1.bell-labs.com/user/cope , 1997
[5] Kroeber, A.L., Anthropology: Culture, Patterns, and Process, 1948
[6] Thorne, Kip S., Black Holes & Time Warps. Einstein's Outrageous Legacy, New York, 1993
[7] Coplien, James O., On the Nature of The Nature of Order, http://www1.bell-labs.com/user/cope , 1997
[8] Alexander, Christopher, The Nature of Order, 4-bändig, 2003

"The Nature of Order presents a unifying model of design for unfolding wholeness by a process of intensifying centers. A good center will reinforce the other centers around it. Existing centers may be augmented by adding adjacent centers, or by augmenting its adjacent centers, or the centers contained within it. The final result is a collection of recursive and hierarchical centers which are integrated at all levels of scale and which reinforce one another. One starts with basic, essential centers and then recursively unfolds the design to elaborate more centers without destroying the existing ones. An example stated by Alexander is that cells divide, and then differentiate themselves to reinforce existing centers. ... **Patterns**, then, are regional points or invariants in the design which exhibit deeper centers; they are configurations of centers that serve some cultural context which is germane to that particular culture."

Neben diesen bis jetzt statischen Aussagen rundet eine die Dynamik kennzeichnende Phrase das Weltbild Alexander's ab – nämlicher jene, die die Prozesse für die Entfaltung von „Wholeness" beschreibt: **„Structure preserving transformations"**. Ein Beispiel für die Anwendung eines derartigen Prozesses ist die Vergrößerung eines Software-Entwicklungsteams. Wird zudem noch dieses Team geteilt und ein zweiter Teamleiter aus dem ursprünglichen Team gewählt, kommt es zu interpersonellen Reaktionen, die eine positive, aber auch negative, Eigendynamik entfalten können. Dies ist ein typisches Umfeld, das in Form von einem oder mehreren Mustern bzw. Patterns – um den ursprünglichen Ausdruck dafür zu verwenden – beschrieben werden kann.

Während nun anhand der Muster (kurzfristige und langfristige) stabile Strukturen beschrieben werden, ist der Begriff der struktur-konservierenden Transformation nicht eigens abgebildet. Vielmehr werden die Transformationen als sehr schnell ablaufende Ereignisse angesehen, deren Wirkung erst anhand der sich einstellenden Ergebnisse beurteilt wird. Damit steht die Thematik der Muster diametral zu den Bestrebungen Aktivitäten in Form von Prozessen oder Abläufen zu beschreiben. J.O. Coplien selbst drückt es folgendermaßen aus: „The real process comes from instinct and avoids explicitly pre-arranged forms."

Unter Zugrundelegung der bisherigen Ausführungen kann die Anwendung von Patterns als Ergänzung zu den Prozessen im Software-Projekt-Management betrachtet werden. Während Prozesse den Weg schildern, und uns sagen, wie der zeitliche Ablauf der Aktivitäten stattfindet, gestatten uns Muster die Beurteilung einer augenblicklichen Situation. Da Muster – anders als Prozesse – nicht an einen inhaltlichen Kontext gebunden sind, bilden sie einen formalen Rahmen bzw. ein System, innerhalb dessen die unterschiedlichsten Sachverhalte formuliert werden können. Insbesondere unterstützen Muster die leichte Modellierbarkeit von Information aufgrund unserer (der menschlichen) Art des Denkens. Bemerkenswert ist hierbei, dass diese Form der Wissensrepräsentation u.a. auch der Bibel eigen ist, die menschliche Verhaltensnormen als Sammlung von Mustern abbildet.

11.2 Formale Grundlagen

11.2.1 Definitionen

Da Patterns ein weites Einsatzgebiet haben und einer aufgrund der Aktualität kontinuierlichen Fortentwicklung unterliegen, sind sich die derzeit existierenden Definitionen ähnlich aber nicht identisch. Ch. Alexander charakterisiert das Pattern in einer seiner ersten Publikationen [9]) durch: „Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution", oder kurz auch als "Rules of Thumb".

Eine andere gängige Definition aus der Pattern-Community [10]): „A pattern is a **named** nugget of instructive information that captures the essential **structure** and insight of a successful family of **proven solutions** to a **recurring problem** that arises within a **certain context** and **system of forces**." Die fett gedruckten Wörter unterstreichen die charakteristischen Merkmale, auf die später explizit eingegangen wird.

Eine Definition aus der Software-Community lautet [11]): „Patterns touch critical issues that are central to strategic software development; we believe that human communication is the bottleneck in software development. If the pattern literary form can help programmers communicate with their clients, their customers, and with each other, they help fit a crucial need of contemporary software development", und weiters: "Patterns focus more on the human activities of design than on transformations that can blindly be automated"

Das Wesentliche von Patterns mit eigenen Worten formuliert: Patterns sind wiederkehrende Problem-Lösungs-Paare. Sie beziehen sich auf Strukturen, Methoden, Praktiken, usw., die das Schlüssel-Know-how innerhalb eines grob umrissenen Systems in ihren wechselseitigen Beziehungen repräsentieren, und in Form einer eigens für Patterns typischen Syntax archiviert werden. Unter dem Begriff „**Problem-Lösungs-Paar**" wird die Verknüpfung eines bestimmten Problems mit einer bestimmten Lösung verstanden. Dadurch wird ausgedrückt, dass Patterns einen Reifungsprozess durchlaufen haben, wodurch ein bestimmtes Problem eine eindeutige Lösung (und umgekehrt eine bestimmte Lösung auf ein eindeutiges Problem zurückzuführen ist) besitzt. Im Gegensatz dazu ist bei einer 1:m, n:1 oder n:m Beziehung keine eindeutige Zuordnung zwischen einem Problem und einer Lösung gegeben.

---

[9] Alexander, Christopher, The Timeless Way of Building, Oxford University Press, 1979

[10] http://www.hillside.net , The Hillside Group is a nonprofit corporation dedicated to improving human communication about computers by encouraging people to codify common programming and design practice, 2004

[11] Coplien, James O., The Human Side of Patterns. C++ Report, 1996

Mit dem Begriff „Pattern" verwandte Begriffe – in Hinblick auf den Inhalt, der durch diese Begriffe verkörpert wird – sind:

- Paradigma
- Mikro-Architektur
- Rollen-Modell
- Prinzip
- FAQs (Frequently Asked Questions)
- Idiom (im Rahmen der Software-Technologie)
- Framework (im Rahmen der Software-Technologie)

Unter anderem werden durch diese Begriffe bestimmte Varianten von Patterns bezeichnet, die jedoch den eigentlichen Eigenschaften von Patterns nur teilweise entsprechen.

11.2.2 Notation

Um Patterns effizient und korrekt anzuwenden, muss ihre Darstellung bestimmte Informationen enthalten. Eine geeignete Notation dient insbesondere folgenden Zielen:

- übersichtliche Einführung des Lesers in das Problem
- Beschreibung des Umstands in welchem das Problem auftritt
- Analyse des Problems
- Darstellung von Lösungen

Die sich hierzu herauskristallisierte Notation, die gleichzeitig als Gliederung zu verstehen ist, umfasst folgende Inhalte:

- Name des Pattern:
  Der Name kann durch ein Substantiv, ein Verb, oder eine ganze Phrase gebildet werden. Er bezieht sich auf das Problem oder die Lösung. Teilweise werden auch Analogien als Bezeichner für Patterns herangezogen.
- Problem:
  Unter dieser Überschrift wird das Problem dargestellt.
- Kontext:
  Da Probleme meist im Zusammenhang mit einem bestimmten Kontext stehen, muss zusätzlich zum Problem auch dessen Umwelt beschrieben werden. Hierzu zählen Attribute, wie z.b. Angaben über den Markt, die Organisation, die angewandte Programmiersprache, usw. Wesentlich ist vor allem die Angabe von Bedingungen unter denen das Pattern gültig bzw. ungültig ist.

- Kräfte:
  Sie wurden ursprünglich von Ch. Alexander eingeführt, um eine Balance zwischen Ästhetik und Komfort einerseits, und physikalischen Strukturen von Städten und Gebäuden andererseits, zu erzielen. Kräfte spiegeln die Interessen und Motivationen wider, die die relevanten Einflüsse von den nebensächlichen differenzieren.

- Lösung:
  Meist wird die vollständige Problemlösung beschrieben. Eine abschnittsweise Lösung wird hingegen dann bevorzugt, wenn auf ergänzende Patterns verwiesen werden kann. Durch die Lösung sollen die auf das Problem einwirkenden Kräfte nach Möglichkeit vollständig aufgelöst, bzw. durch ergänzende Patterns neutralisiert werden.

- *Lösungsbegründung:*
  Um die Lösung zu plausibilisieren sollte eine Erklärung gegeben werden, welche eine bestimmte Lösung eindeutig mit einem bestimmten Problem verknüpft (= Problem-Lösungs-Paar). Eine Möglichkeit dies zu erklären kann z.b. durch Darlegung der Historie zur Entstehung des Pattern erzielt werden.

- Resultierender Kontext:
  Dieser gibt eine Zusammenfassung über die ausbalancierten Kräfte, welche neuen Probleme mit der Lösung verbunden sein könnten, oder welche anderen Patterns mit dem zu beschreibenden Pattern im Zusammenhang stehen.

Neben dieser auf J.O. Coplien zurückgehenden Notation [12]) gibt es je nach Anwendungsgebiet weitere Varianten. Eine auf das Design von objektorientierter Software ausgerichtete Notation ist die „GOF Form" [13]) welche folgende Sektionen aufweist (in Klammern jeweils eine Kurzbeschreibung der Sektion):

- Pattern Name and Classification (Name des Pattern)
- Intent (funktionale Beschreibung des Pattern)
- Also Known As (andere Namen unter denen dieses Pattern bekannt ist)
- Motivation (Bedeutung des Pattern)
- Applicability (Anwendbarkeit des Pattern)
- Structure (durch Graphiken der UML werden die Objektzusammenhänge erläutert)
- Participants (Aufzählung der im Pattern involvierten Klassen und Objekte)
- Collaborations (Interaktionen zwischen den Participants)
- Consequences (Konsequenzen bei Verwendung des betreffenden Pattern)
- Implementation (Hinweise für die Implementierung, d.h. Codierung)

[12] Coplien, James O., Software Patterns, http://www1.bell-labs.com/user/cope/Patterns/WhitePaper/ , 2002
[13] Gamma, B., et al., Design Patterns: Elements of Reusable Object-Oriented Software, 1995

- Sample Code (Beispiel für eine Implementierung in einer Programmiersprache)
- Known Uses (Systeme, in denen das Pattern angewandt wurde)
- Related Patterns (verwandte bzw. ähnliche Patterns)

Als letztes Beispiel einer weiteren Notations-Variante sei jene von Ch. Alexander aus dem Bereich der Architektur erwähnt. Sie wird oft auch als die „Original-Pattern-Form" bezeichnet und deckt sich weitgehend mit jener von J.O. Coplien, der darauf aufsetzte und sie fürs Software-Projekt-Management erstmals anwandte.

Gerne wird die Notation bzw. Struktur auch als „literarische Form von Patterns" bezeichnet. Dies kann direkt oder indirekt auf den „Wiederentdecker" der Pattern-Thematik (Ch. Alexander) zurückgeführt werden, der aufgrund seiner Beschäftigung mit architektonischen Fragestellungen auch dem Gebiet der Kunst sehr nahe steht.

11.2.3 Klassifikation

In der Sparte der Software-Technologie werden Patterns bereits für unterschiedliche Zwecke eingesetzt, wobei sie je nach Ausprägung bzw. Anwendungsgebiet eigene Namen tragen. Etablierte Varianten sind u.a.:

- Idioms:
  Dies sind sogenannte Low-level-Patterns, die von einer bestimmten Implementierung abhängig sind (z.b. von der Programmiersprache). Sie zählen zu den ersten publizierten Software-Patterns (etwa um 1992).

- Design Patterns:
  Sie werden erstmals im Jahr 1995 (von Gamma u.a.) im Zusammenhang mit ausgewählten Praktiken des objektorientierten Entwurfs erwähnt – jedoch unabhängig von einer bestimmten Programmiersprache. Sie werden auch als Mikro-Architekturen bezeichnet, da sie Strukturen beschreiben, die größer als Objekte, aber kleiner als Systeme sind.

- Framework Patterns:
  Diese Patterns sind auf System-Ebene angesiedelt und kombinieren Systemteile und Systemmechanismen. Abgesehen von der Dokumentation eines Frameworks durch Patterns können mit ihrer Hilfe auch die Mechanismen zur Erweiterung oder Reduktion eines Systemumfangs dargestellt werden.

Von untergeordneter Bedeutung und nur der Vollständigkeit wegen erwähnt sind:

- Anti-Patterns:
  Diese repräsentieren eine „literarische Form" der Darstellung von Sachverhalten die nicht funktionieren, bzw. destruktiv sind.

- Meta-Patterns:
  Diese können als Verallgemeinerung von Patterns angesehen werden, spielen jedoch praktisch – abgesehen unter philosophischen Gesichtspunkten - keine große Rolle. Dies steht auch im Widerspruch zum ursprünglichen Sinn von Patterns – nämlich der *unmittelbaren Anwendbarkeit*.

Um den Begriff des Pattern weiter abzugrenzen sollen nun noch einige Erläuterungen darüber gegeben werden, was Patterns nicht sind:

- Da Patterns im weitesten Sinne Erfahrungen in strukturierter Weise wiedergeben sind sie nur bedingt für innovative Vorgehen anwendbar.
- Patterns dienen der Repräsentation wichtiger Zusammenhänge – sie sind keine CASE-Tools.
- Patterns dienen (in der angewandten Praxis) nicht der Erreichung von absoluter Perfektion. Sie verhindern jedoch, dass grundsätzliche Zusammenhänge unberücksichtigt bleiben.
- Der Einsatz von Patterns setzt keine betriebliche Umorganisation oder Prozessänderung voraus. Jedoch ist eine Patterns-unterstützende Kultur unabdingbar.

11.2.4 Pattern Languages

Ein einzelnes Pattern dient der Lösung eines einzelnen Problems. Um ein System in Form eines Mosaiks beschreiben zu können benötigt man eine Vielzahl an Patterns, die sich gegenseitig ergänzen und aufeinander aufbauen. Jedes einzelne dient in diesem Gesamtsystem als Kristallationspunkt – oder in der Sprache des Ch. Alexander als „Center". Die Auflistung aller dieser um einen Kristallationspunkt entwickelten Patterns wird als „Pattern Language" bezeichnet. Der Ausdruck „Language" verdeutlicht die Analogie zu den lebenden Sprachen, dass nämlich jederzeit eine Erweiterung der Sprache um zusätzliche Ausdrücke – in unserem Fall Patterns – nicht nur möglich, sondern auch notwendig ist (vgl. hierzu die vielen neuen technischen Ausdrücke in der deutschen Sprache).

In Ansätzen lässt sich eine Pattern Language auch mit formalen Methoden beschreiben. So bilden die Patterns einer Pattern Language einen gerichteten azyklischen Graphen. Die Anzahl

der unterschiedlichen Wege durch den Graphen lässt einen großen Spielraum für die Adaption eines Pattern an die jeweiligen Bedingungen zu. Die Verbindungen zwischen den Patterns spielen hierbei eine nicht weniger bedeutende Rolle als die Patterns selbst. Angewandt auf den Aufbau von Organisationen bilden die Verbindungen die Kommunikationswege zwischen den Sub-Gruppierungen ab, die es zu optimieren gilt.

11.2.5 Pattern Programm

Um in der betrieblichen Praxis einen Nutzen aus Patterns ziehen zu können erfordert es einer sogenannten „Pattern-Kultur" bzw. eines Pattern-Programms. Dazu zählen:

- Pattern-Training
  Um aus einer Pattern Language zu profitieren, müssen die wichtigsten Patterns einer Domäne in Form eines Trainings vorgestellt werden, auf denen aufbauend eigene formuliert werden können.

- Pattern-Mining
  Da Patterns konkrete interne Strukturen von betrieblichen Organisationen abbilden, wird ein Großteil der Patterns nicht allgemein zugänglich sein. Dies bedeutet, dass sie zum wesentlichen Teil innerhalb einer Organisation entwickelt werden müssen. Sie stammen dabei von Leuten und – im übertragenen Sinne – von Organisationen und Produkten mit langer, erfolgreicher Historie. Sie bilden u.a. Schlüssel-Strategien ab, die die jeweilige Organisation wachsen ließen.

- Pattern-Publikation
  Für die (interne und/oder externe) Kommunikation von Patterns eignen sich insbesondere Medien des World Wide Web (WWW), wo durch die Verwendung von Hypertext-Sprachen (HTML) die für Patterns wichtige Verknüpfung in optimaler Weise erreicht wird. Hierzu sei noch eine Bemerkung von J.O. Coplien angebracht [14]): „More important than the format of publication is the review process", und weiters "good patterns are timeless literature ... with broad appeal that is easily understood, and which captures timeless design wisdom. It goes without saying that very little timeless literature has emerged from the engineering community."

- Pattern-Applikation
  Dies ist der wichtigste Aspekt. Ohne eine geeignete Anwendung werden Patterns für ein Unternehmen zum reinen Kostenfaktor und somit unrentabel.

[14] Coplien, James O., Software Patterns, http://www1.bell-labs.com/user/cope/Patterns/WhitePaper/ , 2002

11.2.6 Qualitätskriterien für Patterns

Die aufgrund der Reifung des Pattern-Gedanken formulierten Qualitätskriterien [15]) aus der Pattern-Community [16]) sind (unter Verwendung der Originalbezeichnungen):

- Encapsulation:
  Jedes Pattern kapselt ein Problem-Lösungs-Paar innerhalb einer Anwendungsdomäne ein.
- Abstraction:
  Patterns sind auf verschiedenen Ebenen konzeptueller Granularität innerhalb der entsprechenden Domäne anwendbar.
- Openness:
  Um Probleme unterschiedlicher Komplexität zu lösen, sollen Patterns um weitere Patterns erweiterbar sein.
- Variability:
  Ein Pattern soll durch unterschiedliche Implementierungen realisierbar sein, um mit anderen Patterns optimal kombinierbar zu sein.
- Generativity:
  Adressierung von Pattern Languages und dem Gedanken an ganzheitliche Lösungen („Wholeness"): „Each pattern, once applied, generates a resulting context which matches the initial context of one or more other patterns in a pattern language."
- Composability:
  Ein resultierender Kontext schafft die Basis für den Kontext des zu verknüpfenden Patterns; in diesem Zusammenhang wird auch von fraktalen Strukturen gesprochen: „But patterns are not simply linear in nature, more like fractals in that patterns at a particular level of abstraction and granularity may each lead to or be composed with other patterns at varying levels of scale."
- Equilibrium:
  Die Kräfte sollen in der Lösung in einem gleichgewichtigen Zustand sein.
- Syntax:
  Sie dient einer effizienten und korrekten Anwendung von Patterns.
- Accessibility and Usage:
  „Ohne Nutzung kein Nutzen !"

[15] Appleton, Brad, Patterns and Software: Essential Concepts and Terminology, http://www.bradapp.net/ , 2000
[16] http://www.hillside.com , 2003

11.2.7 Historie zu Patterns in der Software-Community

Wie schon zuvor im Abschnitt über die philosophischen Grundlagen dargelegt, werden Muster bzw. Patterns schon in früheren Publikationen in unterschiedlichster Weise erwähnt. Ebenso wurde dargelegt, dass Patterns – so wie sie in dieser Arbeit vorgestellt werden – auf Christopher Alexander zurückgehen. Er wurde 1936 in Wien geboren, erwarb in Cambridge das Master Degree der Mathematik und das Bachelor Degree der Architektur; in Harvard promovierte er zum Doktor (PhD) der Architektur. Die Grundlage zu seinen bekannten Werken legte er im Rahmen dieser Dissertation mit selbst entwickelten Computer-Programmen, die auf Basis einer logischen Analyse von Objektteilen neue Umgebungen analysieren und kreieren sollten. Danach widmete er sich der empirischen Forschung zu Fragestellungen der Architektur und Kunst, die ihn zur Theorie der Patterns führten, in der sich Logik und kollektive Erfahrung mischten. In den 80'er Jahren entwickelte er eine Theorie zum Thema: „Wholeness" – eine tiefgehende Diskussion über ewige Schönheit in der Architektur und im Leben als Ganzes.

Anfang bis Mitte der 80'er Jahre sprang der „Pattern-Funke" von der Architektur auf die Software-Technologie über. Kent Beck bemerkt hierzu [17]): „I first discovered patterns as an undergraduate at the University of Oregon. Many of the students in my freshman dorm ... were in the School of Architecture. Since I had been drawing goofy house plans since I was six or seven, they pointed me in the direction of Ch. Alexander. I read all of The Timeless Way of Building standing up in the university bookstore over the course of several months." Dieses Kommentar verdeutlicht in leicht verständlicher Weise die von T.S. Kuhn [18]) erkannte sprunghafte Entwicklung (Paradigmenwechsel) einer Wissenschaft (hier: der Informatik). Im Jahr 1987 präsentierten Ward Cunningham und Kent Beck im Rahmen der OOPSLA (Object-Oriented Programming, Systems, Languages, and Applications) in Orlando, USA ihre positiven Erfahrungen beim Entwerfen einer GUI unter Zuhilfenahme einiger selbst entworfener Patterns – das Interesse des Publikums war angeblich (fast) gleich Null [19]). In einschlägigen Workshops wurde die Idee anhand von „C++ patterns" verbreitet, welche unter der Bezeichnung: „C++ styles" oder „C++ idioms" in die Literatur Einzug hielten.

Im Jahr 1993 einigten sich einige der Autoren, die zu diesem Thema publizierten, auf eine Verschmelzung von Pattern- und Objekt-Thematik. Die aufgrund dessen interpretierbare Geburt der Patterns- aus der Objekt-Thematik ist dadurch zu erklären, dass zur Zeit des Aufkommens der Patterns die Objekt-Thematik in Hochblüte stand. Aus diesem Grund soll nochmals unterstrichen werden, dass obwohl die beiden Ansätze manche Analogien zeigen, unabhängige Wurzeln haben. Auf der OOPSLA 93' war das Thema: „Patterns" eines der zentralen Punkte auf der Agenda; und 1994 plante die „Hillside Group" die erste „Pattern Languages of Programming" (PLoP) Konferenz (im Jahr 2003 fand die 10. PLoP statt).

---

[17] http://c2.com/ppr , Portland Pattern Repository, 2004
[18] Kuhn, Thomas S., The Structure of Scientific Revolutions, University of Chicago, 1970
[19] Coplien, James O., Software Patterns, http://www1.bell-labs.com/user/cope/Patterns/WhitePaper/ , 2002

11.3 Beispiel einer Pattern Language

Basierend auf den Arbeiten von Ch. Alexander [20]) hat J.O. Coplien einige Muster von Software-Organisationen unter dem Titel „A Development Process Generative Pattern Language“ [21]) formuliert. Auf ähnliche Weise werden in den folgenden Abschnitten Patterns vorgestellt, die im Umfeld des Software-Projekt-Managements und der Software-Organisations-Gestaltung von Bedeutung sind. In ihrer Kombination repräsentieren sie eine „Pattern Language“.

Im ersten der beiden nachfolgenden Unterkapitel werden drei Patterns in der Notation von J.O. Coplien dargestellt. Sie bilden somit die ersten drei Bausteine der Pattern Language, welche im zweiten Unterkapitel um Ansätze für zusätzliche Patterns (Bausteine) erweitert werden. Diese Ansätze basieren auf eigener Erfahrung und sind aus Gründen der besseren Übersicht nicht in einer Pattern-Notation verfasst.

11.3.1 Patterns eines Softwarehauses in Coplien – Notation

Die im folgenden dargestellten drei Patterns sind in der Notation von J.O. Coplien verfasst. Zusätzlich wurde das dritte Pattern („Terminplanabstimmung“) beispielhaft auch hinsichtlich der Qualitätskriterien für Patterns analysiert.

11.3.1.1 Pattern: Entwicklungs-Standort-Verteilung als Faktor des Organisationsaufbaus

- Name:
  „Entwicklungs-Standort-Verteilung als Faktor des Organisationsaufbaus“
- Problem:
  Aufgaben- und Rollenverteilung über eine geographisch verteilte Organisation
- Kontext:
  Die Entwicklungsstätten befinden sich in unterschiedlichen Stockwerken, Gebäuden oder in weltweit verteilten Entwicklungszentren
- Kräfte:
  Die Kommunikationsmuster zwischen den Projektmitgliedern orientieren sich an deren geographischer Verteilung. Die Kommunikationsbereitschaft nimmt mit der räumlichen

[20] Alexander, Christopher, et al., A Pattern Language, New York, 1977
[21] Coplien, James O., A Development Process Generative Pattern Language, AT&T Bell Laboratories, http://www.bell-labs.com/~cope/Patterns/Process/index.html , 1995

Distanz ab. Die innerhalb einer organisatorischen Einheit anfallenden Aufgaben sind sich meist ähnlich, wodurch die Bereitschaft zur Kommunikation begünstigt wird.

- Lösung:
  Die Software-Architektur muss die geographische Verteilung der Entwicklungstätigkeiten widerspiegeln, um die Kommunikationsschnittstellen möglichst gering zu belasten. Mit der Tätigkeitsverteilung müssen auch die zugehörigen Kompetenzen lokal verteilt werden.

- Lösungsbegründung:
  Die Annahme, die der Lösung zugrunde liegt, ist, dass mit zunehmender Entfernung der Unterschied in den Zeitzonen, der Sprache, der Kultur, und anderen Kriterien zunimmt, was in Folge den Kommunikationsaufwand mit der Entfernungszunahme erhöht. Zudem lassen sich über geringe Distanzen persönliche Kontakte besser pflegen als über Kontinente hinweg. Daher basiert die Lösung auf einer Entfernungs-Minimierung möglichst vieler Kommunikations-Partner.

- Resultierender Kontext:
  Bei Anwendung dieses Pattern existieren Suborganisationen, die abhängig vom Markt oder anderen Kriterien weiter unterteilt werden können (vgl. z.b. mit Pattern „Marktbedingte Organisation"). Es ist unbedingt erforderlich jemanden zur Konfliktlösung zwischen den verteilten Standorten zu ernennen.

- Weiterführende Anmerkungen:
  Die Kommunikation zwischen entfernten Standorten muss nicht notwendigerweise schlecht sein – hierzu müssen allerdings folgende Kriterien erfüllt sein:

  - die Gesamtzahl der an einem Projekt beteiligten Entwickler aller Standorte ist verhältnismäßig gering (in Summe nicht mehr als etwa 7 – 12 Mitarbeiter)
  - Kommunikationsmittel wie das E-Mail oder Internet nehmen eine zentrale Stellung ein
  - die Projektmitglieder kennen sich persönlich, d.h. sie haben schon einige Zeit lang in unmittelbarer Nähe (z.b. im selben Büro) gearbeitet
  - die Projektmitglieder reisen gerne – vorausgesetzt die Notwendigkeit besteht dazu

  Andererseits können Markterfordernisse eine verteilte Entwicklung als Notwendigkeit voraussetzen, sodass alle anderen Wirkungen zweitrangig sind.

11.3.1.2 Pattern: Kundeneinbindung

- Name:
  „Kundeneinbindung"

- Problem:
  Sicherstellung der Kundenzufriedenheit

- Kontext:
  - Eine Qualitätssicherungs-Rolle existiert
  - Teil der QS ist auch die Kontrolle über die Requirements
- Kräfte:
  - Entwickler sind manchmal vergleichbar mit „Kanonen ohne Zielfernrohr"
  - Requirements ändern sich auch noch nach Beginn der Codierungsphase
  - Fehlende oder widersprüchliche Kundenrequirements sind ein ernstes Problem
  - Kunden wissen oft nicht was sie eigentlich wollen
  - Entwickler haben meist keinen Kontakt zu Kunden
  - Die Ideen der Kunden sind – wenn überhaupt – nur dem Projektleiter bekannt
- Lösung:
  Der Kunde muss Kontakt zu folgenden Rollen haben:
  - Vertrieb / Management des Softwarehauses
  - Architekt
  - Entwickler
  - Qualitätssicherung
- Lösungsbegründung:
  Da in der Mehrzahl der Software-Projekte die Kundenanforderungen zu Projektbeginn nicht vollständig vorliegen, ist eine langfristige Einbindung des Kunden während der gesamten Entwicklungsdauer nötig. Selbst nach Fertigstellung der Codierungsphase ist noch mit Kundenanforderungen zu rechnen. Um die Anzahl der nachträglichen Kundenwünsche zu minimieren, ist zu Projektbeginn auf eine möglichst vollständige Spezifikation der Anforderungen Wert zu legen. Aber selbst dann können vom Kunden in späterer Folge Änderungswünsche beantragt werden, die in den SEP einfließen müssen. Am besten installiert man hierzu die Rolle eines Claim-Managers, der die Vermittlung zwischen Kunden, Entwicklern, und gegebenenfalls, weiteren am Projekt Beteiligten verantwortet. Die Aufgaben des Claim-Managers sind die Claim-Vorsorge, -Erkennung und –Verfolgung, die u.a. folgende Aufgaben umfassen:
  - Mitwirkung bei der Vertragsgestaltung zwischen Auftragnehmer und –geber (Kunde)
  - Verwaltung von Kundenforderungen bezüglich Produkt- oder Projektumfang
  - Verwaltung von Kundenforderungen hinsichtlich Gewährleistung, Garantie und Schadensersatz
  - Durchsetzung eigener Claims gegenüber anderen
- Resultierender Kontext:
  Zu beachten ist auch der Querbezug zu den Patterns:
  - Problemdarstellung durch Szenarien
  - Prototypen
  - Firewall

11.3.1.3 Pattern: Terminplanabstimmung

11.3.1.3.1 Formulierung

- Name:
  „Terminplanabstimmung"
- Problem:
  Bestimmung des richtigen Terminplans
- Kontext:
  Die Software befindet sich in einer mittleren Phase des Entwicklungsprozesses
- Kräfte:
  - ein zu langgestreckter Terminplan schließt das Marktfenster
  - ein zu ehrgeiziger Terminplan überlastet die Entwickler
  - Software-Projekte ohne Terminplan tendieren (fast) generell zu einem Terminverzug; ein häufiger Grund dafür ist ein „Software-Overkill" gegenüber den Anforderungen
- Lösung:
  - Aufstellen einer Roadmap
  - Wöchentliche Kontrolle des kritischen Pfades
  - Erst bei größerer Verzögerung sollte der Plan neu überarbeitet werden
  - Unterschiedliche Terminpläne für die interne und externe Kommunikation
- Lösungsbegründung:
  Ein zu großzügig ausgelegter Terminplan widerspricht den Markterfordernissen. – Ein zu ambitioniert gestalteter Terminplan führt einerseits zu einer Vernachlässigung architektonischer Prinzipien und andererseits zu einer Überanstrengung personeller Ressourcen mit all ihren Folgen. Da es in der überwiegenden Mehrzahl der Software-Projekte zu Verzögerungen kommt, empfiehlt sich die Führung zweier Terminpläne: Einer, der mit dem Kunden, und ein anderer, der mit den Mitarbeitern abgestimmt ist. Hierbei ist zu beachten, dass die Unterschiede immer den Anforderungen entsprechend gestaltet sein müssen. Ein immer gleich gewählter Zeitpuffer würde dem Entwicklungsteam rasch bekannt werden, womit der interne Plan nutzlos wäre.
- Resultierender Kontext:
  - Existenz eines internen und externen Zeitplans
  - Anwendung geeigneter Methoden zur Leistungsstand-Beurteilung

11.3.1.3.2 Qualitätskriterien

Die in einem vorhergehenden Abschnitt aufgezählten Qualitätskriterien für Patterns werden im folgenden auf das Pattern „Terminplanabstimmung" angewandt:

- Encapsulation:
  Das Pattern kapselt das Problem-Lösungs-Paar (vgl. hierzu Abschnitt 11.2.1) innerhalb der Anwendungsdomäne des Software-Projekt-Managements ein
- Abstraction:
  Das Pattern ist sowohl für ein Teilprojekt, als auch für das Gesamtprojekt anwendbar
- Openness:
  Das Pattern behindert in keiner Weise eine Ergänzung durch andere Patterns (z.b. betreffend der Methodiken zur Feststellung des Leistungsfortschritts)
- Variability:
  Auch hierfür werden keinerlei Grenzen innerhalb des Pattern gesetzt
- Generativity:
  Die Ergebnisse aus der Anwendung des Pattern können die Ausgangsbasis für weitere Patterns darstellen
- Composability:
  Dass die Anwendung dieses Pattern im Verbund mit weiteren (innerhalb des Software-Projekt-Managements) geschieht ist offensichtlich
- Equilibrium:
  Durch die zwei unterschiedlichen Terminpläne werden sowohl die Anforderungen der Vertriebsseite, als auch jene des Entwicklungsteams berücksichtigt
- Syntax:
  Diese wird durch die Notation erreicht
- Accessibility and Usage:
  „Ohne Nutzung kein Nutzen !" – dies gilt für die Terminplanung in jeder Hinsicht

11.3.2 Ansätze für Patterns zur Erweiterung der Pattern Language eines Softwarehauses

Aus Gründen der besseren Übersicht und der Fokussierung auf die inhaltliche Seite von Patterns sind die im folgenden dargestellten Patterns nur ansatzweise, d.h. in keiner für Patterns charakteristischen Notation verfasst.

11.3.2.1 Abbildung von Domänen-Fachwissen auf Rollen

Dieses Muster zielt auf die Besetzung aller Rollen mit qualifizierten Personen ab. Diese Forderung ist eine der wesentlichsten für erfolgreiches Software-Projektmanagement. So wie sich im Laufe der Geschichte die Zünfte entwickelt haben – als nämlich sämtliche handwerkliche Fertigkeiten nicht mehr in einer Person vereinbar waren – nimmt auch das Wissen um die Software-Entwicklung rasch zu. Der Grund, warum wir noch immer allgemein von einem Software-Entwickler, und nicht von einem Software-Architekt, einem Software-Systemtester, einem Software-Konfigurations-Manager, usw., sprechen, sind die extrem kurzen Zyklen, in denen sich die Software-Technologie weiterentwickelt (alte Technologien defragmentiert und zu neuen integriert), wodurch auch die Anforderungen an die Software-Entwickler einer kontinuierlichen Veränderung – und somit Bezeichnung – unterzogen sind.

Beim Bilden der Rollen müssen neben den persönlichen Fähigkeiten auch die notwendigen Kommunikations-Ströme berücksichtigt werden. Dieses sich aus den einzelnen Strömen ergebende Kommunikationsmuster muss auch im Einklang mit der geographischen Raum-Ordnung der Mitarbeiter stehen. Diese Gedanken sind eng mit dem Muster „Lehrling" verknüpft, wonach neue Mitarbeiter in ein möglichst enges Arbeitsumfeld von Experten („Gurus" – der Begriff wird im nächsten Absatz erläutert) einzugliedern sind. – Hier kommt das Raumproblem besonders zur Geltung. Auf dem Niveau des individuellen Leistungsbeitrages – im Gegensatz zur Leistung eines gesamten Teams – ist das Fachtraining gegenüber dem Prozesstraining von erheblich größerer Bedeutung.

Als Garant für ein erfolgreiches Projekt ist ein „Anzahl an Gurus zur Gesamtzahl aller Mitarbeiter – Verhältnis" von mindestens 50% unbedingt notwendig. Hierbei bezeichnet der Begriff „Guru" einen Mitarbeiter

- der in der zu übernehmenden Rolle genügend Erfahrung gesammelt hat, um sie ohne externe Hilfe meistern zu können, und,
- die soziale Kompetenz besitzt, einen neuen Mitarbeiter in die Rolle einzuweisen, sodass dieser nach einer vernünftigen Zeitspanne (einige Monate) selbständig die Rolle ausfüllen kann.

11.3.2.2 Architektur bedingte Organisation

Wie schon im Pattern „Funktion bedingt die Form" dargestellt, bestimmen die einzelnen Aktivitäten des SP die Form der Organisation. Fragt man nach der grundlegenden Determinante für die Arbeitspakete, gipfelt die Antwort im Design der Architektur. Dieses Pattern ist auch unter der Bezeichnung „Conway's law" bekannt.

11.3.2.3 Architektur-Review

Eine alles berücksichtigende Software-Architektur ist die Basis für den Erfolg eines SP. Organisatorische Fehlentscheidungen führen in der Regel zu terminlichen Problemen, Fehler in der Architektur hingegen zum völligen Scheitern. Probleme, die durch Fehler in der Architektur ausgelöst werden, sind:

- Änderungen der Architektur während der Implementierungs-Phase führen zu aufwendigen Änderungen des Codes
- die Performance der Software ist durch die Architektur begrenzt, womit der Zielmarkt von vornherein eingegrenzt wird
- eine zu flexibel gestaltete Architektur führt zu Kosten- und Terminüberschreitungen
- eine zu wenig flexibel gestaltete Architektur macht eine schrittweise Software-Weiterentwicklung in Versionen ohne Architekturbruch unmöglich

Aufgrund dieser Argumente sind ausführliche Softwarearchitektur-Reviews unumgänglich. Hierbei sei auch darauf hingewiesen, dass die Kosten für eine Fehlerbehebung in der Anfangsphase eines SP am günstigsten sind – verglichen mit den Kosten einer Fehlerbehebung während des Systemtests.

11.3.2.4 Chef

Gegenüber der Rolle des Meisters unterscheidet sich die Rolle des Chefs in Bezug auf dessen Kernaufgabe – der Ausübung von Entscheidungs- bzw. Konflikt-Lösungs-Kompetenz. Obwohl andere Rollen ebenfalls über ein Attribut „Entscheidungs- bzw. Konflikt-Lösungs-Gewalt" verfügen, ist jenes des Chefs – hierarchisch betrachtet – im Vordergrund. Die Rolle ist vergleichbar mit jener des Champion im Change-Management-Prozess (in der Ethnologie würde diese Rolle als Königs- oder Vaterrolle bezeichnet werden). Die Aufgaben dieser Rolle sind u.a.:

- Wahrnehmung der Verantwortung für die Organisation
- schnellstmögliches Treffen von Entscheidungen (Gefahr der Verfolgung divergierender Lösungswege und der Produktivitätsminderung)
- Bewältigung von Hindernissen auf Projektebene
- Schaffung eines guten Betriebsklimas
- Wahrnehmung der zentralen Rolle für Prozess- und Organisations-Änderungen

Somit entspricht diese Rolle der klassischen Entwicklungsleiter-Rolle.

11.3.2.5 Code-Besitzer

Zentrale Motivation für eine explizite Zuweisung von Segmenten der Software (des Codes) an Personen ist das bekannte Phänomen: „Für etwas, für das jeder verantwortlich ist, ist letztlich niemand verantwortlich". Eine Nichtbeachtung dieses Pattern führt vor allem in großen SPs zu Problemen, wo kaum noch jemand einen Gesamtüberblick hat.

Die Zuweisung eines Code-Segments an einen Software-Entwickler bedingt auch, dass nur eine Person das Recht zu Code-Änderungen besitzt. Auf diese Weise wird sicher gestellt, dass es immer zumindest eine Person gibt, die diesen Software-Abschnitt bis ins kleinste Detail kennt. Notwendig wird dies aus dem Umstand heraus, dass der größte Teil des Design-Wissens im Code abgebildet ist, und eine Erarbeitung des Inhalts eines Code-Segments bei ungenügender Dokumentation erheblichen Aufwand bedeutet. In diesem Zusammenhang sollte jedoch darauf hingewiesen werden, dass das Muster des Code-Besitzes nicht zu mangelnden Dokumentations-Anstrengungen bzw. Kosteneinsparungen Anlass gibt.

11.3.2.6 Dokumente-Verfasser

Obwohl die Sinnhaftigkeit einer ausführlichen Dokumentation außer Frage steht, behindert sie sehr oft den Fortschritt der Entwickler. Die formalen Ansprüche an die Dokumentation und die sinnvolle Online-Organisation erfordern zu diesem Zweck eine eigene Rolle.

Viele objektorientierte Entwicklungs-Umgebungen unterstützen die Dokumentation durch das verbindliche entwicklungsbegleitende Erstellen von Kommentaren. In diesem Fall konzentriert sich die Rolle des Dokumenten-Verfassers auf die Administration des Dokumenten-Systems und die Einhaltung der Qualitäts-Standards hinsichtlich des Inhalts.

11.3.2.7 Entkopplung der Phasen

Um von einem Prozess überhaupt sprechen zu können, muss er formulierbar sein. Dies geschieht durch eine Charakterisierung der Teilaufgaben, die untereinander in eine chronologische Beziehung gebracht werden – die Phasen des SEP. Dadurch wird der Prozess steuerbar – die Voraussetzung für:

- Kosten-Controlling
- Termin-Controlling
- Qualitäts-Sicherung
- Ressourcen-Optimierung

- Prozess-Steuerung zu diversen Zwecken

Dabei wird zur Erfüllung oben genannter Zwecke die klare Abbildung des SEP in einfache Modelle (wie hier z.b. der Entkopplung der Phasen) umso wichtiger, je mehr Personen am Software-Projekt beteiligt sind.

11.3.2.8 Erfolgsbeteiligung

Jedes Software-Projekt verlangt von seinen Beteiligten hohe Motivation und Einsatz. Dies erfordert ein entsprechendes Entlohnungs-System, das auf die Erfolge der einzelnen SPs reagiert. Das klassische System der Beförderung ist aus verschiedensten Gründen nur sehr eingeschränkt anwendbar. Zudem muss bei SPs die Bedeutung des Einsatzes eines ganzen Teams berücksichtigt werden, was einer ausschließlich individuell ausgerichteten Belohnung widerspricht. Als Lösung ist in vielen Fällen eine Kombination aus individuellem und team-abhängigem Anteil erfolgversprechend.

11.3.2.9 Firewall

Manchmal kann es in Projekten vorkommen, dass durch unklare Kompetenz-Regelungen, oder generell, durch mangelnde Abschirmung des Projektteams von der Außenwelt, auf die Projekt-Mitarbeiter mehr Information einströmt als nützlich ist. Dies zu verhindern ist die Aufgabe der Rolle: Firewall. Dieses Muster ist eng verwandt mit dem Pattern „Gate keeper".

11.3.2.10 Flaschenhals

Werden zu viele Kommunikations-Ströme über eine einzige Person gebündelt, kann es zur Bildung eines „Flaschenhalses" kommen. In der Folge werden Rollen von wichtiger Information abgeschnitten, wodurch sich Kommunikations-Ströme neu ausrichten. Werden durch die Flaschenhals-Bildung auch Entscheidungen verzögert, ist die Wahrscheinlichkeit groß, dass sich weitere Entscheidungs-Zentren etablieren, die den bisherigen inhaltlich entgegenwirken können.

11.3.2.11 Funktionale Zentren gestalten die Organisation

Eine heutzutage größtenteils auf Prozessen basierende Software-Projekt-Strukturierung vernachlässigt die kontinuierliche Zuteilung der einzelnen Arbeitspakete auf Personen (eine Zuteilung zu Projektbeginn ist zumeist vorhanden). Die Schwierigkeit liegt vor allem bei der Aktualisierung der durch unvorhergesehene Ereignisse wechselnden Zuordnung. Hier empfiehlt sich eine Gruppierung eng miteinander in Beziehung stehender Aktivitäten. Dies sind jene Aktivitäten,

- die bezüglich der Realisierung zeitlich hintereinander liegen,
- von der Tätigkeit ähnlich sind,
- dasselbe Objekt betreffen, oder
- semantisch mit derselben Domäne verwandt sind.

Diese zu Gruppen zusammenfassbaren Aktivitäten werden abstrahiert und als Rollen interpretiert. Somit hat man neben der Strukturierung in Arbeitspakete auch eine Gruppierung hinsichtlich der eng miteinander in Beziehung stehenden geistig-manuellen personifizierbaren Tätigkeiten etabliert.

Ein Beispiel: Kommen in einem Objektmodell (Ausgangsbasis der objektorientierten Programmierung) einige fast identisch zu implementierende Beziehungen vor, die zudem gewisse Kenntnisse voraussetzen, könnte die Codierung einer Rolle, und in weiterer Folge einer bestimmten Person, zugeordnet werden. Dieses Beispiel zeigt auch die enge Verwandtschaft dieses Musters mit jenem („Abbildung von Domänen-Fachwissen auf Rollen"), wonach die Arbeitsaufteilung die Kenntnisse der Mitarbeiter berücksichtigen muss.

11.3.2.12 Gatekeeper

Abgesehen vom Wunsch zur Begrenzung der Kommunikation, kann in manchen – vor allem größeren Projekten – die Installation eines Gatekeepers notwendig sein. Diese z.b. zwischen Entwicklung und Marketing oder Unternehmens-Kommunikation angesiedelte Rolle hat nicht nur das Blockieren von unnötigen Nachrichten, sondern auch das gezielte Weiterleiten von essentieller Information zur Aufgabe.

11.3.2.13 Gestaltung der Kommunikationswege

Dieses Muster liegt vielen anderen zugrunde. Es betont die Bedeutung eines funktionierenden Interaktionsmusters zwischen den Rollen der Softwareorganisation. Grundsätzlich wird die

Kommunikation zum Zweck der Erfüllung rollenspezifischer Aufgaben bestimmt. Trotzdem treten oft Defizite zu Tage, die auf unzureichenden Informations-Austausch zurückzuführen sind. Durch die Verleihung von Titeln und Verantwortlichkeiten kann die Kommunikation in gewünschter Weise stimuliert werden.

11.3.2.14 Integration der Qualitäts-Sicherung in den SEP

Die Software-Qualitäts-Sicherung ist Teil eines Controlling-Prozesses, welcher die Kunden-Anforderungen hinsichtlich Qualitäts-Erfordernissen laufend überprüft und gegebenenfalls auch in den SEP eingreift. Aufgrund der Erfordernis möglichst fehlerfreier Software für kritische Anwendungen (z.b. Atomkraftwerke, Medizintechnik, Luft- und Raumfahrt, Militär, usw.) haben sich zahlreiche Standards diesbezüglich herausgebildet. Als Beispiel sei das Capability Maturity Model genannt (vgl. hierzu den Capability Maturity Model – Ansatz (CMM) in Kapitel 8).

11.3.2.15 Kommunikationsdichte

Jede Rolle innerhalb einer Organisation hat prinzipiell die Möglichkeit, mit jeder anderen zu kommunizieren (vollständige Kommunikation). Aufgrund der Aufgabenteilung ergibt sich jedoch ein asymmetrisches Interaktionsmuster, wobei eine Rolle langfristige und intensive Kontakte nur zu 3 bis max. 7 anderen Rollen sinnvoll unterhalten kann.

Als Kommunikationsdichte einer Organisation kann man das Verhältnis der genutzten Kommunikationspfade innerhalb der Organisation zur Gesamtzahl aller möglichen Kommunikationspfade definieren. Dabei weisen produktivere Organisationen ein höheres Verhältnis als weniger produktive aus.

11.3.2.16 Kontrollierte Unterbrechung

Aufgrund von Abhängigkeiten zwischen Arbeitspaketen kommt es zeitweise zu notwendigen Wartezeiten, von denen einzelne Entwickler betroffen sind. Zur Nutzung der unproduktiven Zeit steht entweder die kurzfristige Übernahme anderer Projektaufgaben oder eine neben dem Projekt parallel laufende andere Tätigkeit zur Auswahl. In jedem Fall aber sollte von vornherein mit möglichen Unterbrechungen von blockierten Tätigkeiten gerechnet werden um gezielt handeln zu können.

11.3.2.17 Kristallisationspunkte

Selbst voll funktionsfähige und einmal optimierte Organisationen können sich rasch ins Gegenteil verwandeln, sollten sie in ihrer Größe verändert werden. Um die mit raschen Expansionen und Kontraktionen verbundenen Aufwände zu minimieren, empfehlen sich identifizierbare Unterdomänen. Sie wirken wie Kristallisationspunkte, die bei Bedarf wachsen können, und dennoch ihre Struktur beibehalten (vgl. hierzu insbesondere Abschnitt 11.1.2).

Im Falle der Expansion ist für die Gestaltung der neu gebildeten Teilorganisationen (aufgrund der Teilung aus der ursprünglichen Organisation entstanden) wesentlich, dass sie an jenen Stellen voneinander getrennt werden, an denen es die losesten Bindungen gab. Rollen mit starkem Kommunikationsaufkommen sollten wieder innerhalb derselben Teilorganisation liegen.

11.3.2.18 Last-Ausgleich

Trägt man in einem quadratischen Raster entlang beider Achsen sämtliche Rollen auf, kann man an den Kreuzungspunkten zweier Rollen die Eigenschaften über deren Kommunikation (z.b. Kommunikations-Intensität) untereinander eintragen. Das so erhaltene Bild verhilft zu einem schnellen Überblick über das Kommunikations-Muster der gesamten Organisation. Ist z.b. die Kommunikations-Intensität gleichmäßig über die gesamte Fläche verteilt, findet keine strukturierte Kommunikation statt, was wiederum den Umstand verdeutlicht, dass die Organisation nicht optimal gestaltet ist.

Die Beachtung dieses Musters ist insbesondere bei großen Projekten notwendig, da der Kommunikationsaufwand überproportional mit der Anzahl der Mitarbeiter wächst.

11.3.2.19 Lehrling

Dieses Muster betont das klassische Meister – Lehrling – Verhältnis, wie es auch in der Softwarebranche anzufinden ist. In der Praxis sind nur wenige Fachhochschul- bzw. Universitäts-Abgänger in der Lage (innerhalb kürzester Zeit) eine tragende Rolle in SPs einzunehmen. Dies unterstreicht den nicht zu unterschätzenden Einfluss des „Learning by Doing". Wie bei den klassischen Lehrberufen niemand Meister nur aufgrund eines Schulbesuchs wird, gilt dieses Prinzip auch für die Software-Entwickler. – Es sei denn, jemand begann schon während seines Studiums mit dem Sammeln von Praxiserfahrungen. Im Durchschnitt sollte jedoch mit einer Trainingszeit von etwa einem Jahr gerechnet werden. Die

Methode des „Am besten lässt man den Neuling ins kalte Wasser springen" weist in der Praxis auf unzureichende Führungsqualitäten hin.

Eine im Zusammenhang mit diesem Muster noch stark ausbaufähige Unternehmens-Strategie ist eine schon während des Studiums beginnende Eingliederung des Studenten ins Unternehmen. Obwohl diese Strategie in einigen Punkten schon seit längerem realisiert ist (z.b. durch Ferialpraktika, periodische Unternehmens-Veranstaltungen), existiert noch beträchtliches Verbesserungspotential – z.b. bei der Eingliederung in die spezifischen Arbeitsabläufe.

11.3.2.20 Marktbedingte Organisation

Durch eine sich weltweit angleichende Nachfrage nach denselben Produkten einerseits, und den dennoch recht unterschiedlichen Kulturen andererseits, sind auch die Logistikprozesse einer Anpassung unterworfen. So fordert z.b. der chinesische Markt – auch im Industriesektor – eine Chinesifizierung der GUIs (Graphische Benutzer-Oberflächen); weiters ist eine Einbindung lokaler Firmen in die Serviceaktivitäten gewünscht. Diese Forderungen führen zu einer Verschiebung der Wertschöpfung hin zum Ort des Kunden. Was in den 70'er Jahren für die Automobil-Industrie gegolten hat, gilt mittlerweile auch für die Software-Industrie. – Und dies nicht nur wegen des Aspekts der Personal-Kosten-Minimierung.

Dieser Kraft zur lokalen Wertschöpfung steht die Forderung nach einem Standard-Produkt mit all seinen Vorteilen gegenüber. Die damit gestellten widersprüchlichen Anforderungen lassen sich jedoch mit den heute zur Verfügung stehenden objektorientierten Programmiersprachen entsprechend realisieren. Wichtig ist hierbei eine möglichst frühe Berücksichtigung marktbedingter Anforderungen schon während der Design-Phase zur Software-Architektur.

11.3.2.21 Personalaufbau

Die Kapazitätsplanung ist die Basis für den Mitarbeiteraufbau. Im Unterschied zum klassischen Projekt-Management ist der Know-how-Aufbau in Softwareprojekten besonders kritisch. Damit einhergehend ist die Möglichkeit, dass die Teamstärke in Abhängigkeit vom Aufwand temporär variiert, stark eingeschränkt. Eine typische Situation in SPs ist jene, dass nach 50% bis 80% der für ein Projekt anberaumten Zeit festgestellt wird, dass die noch ausständige Arbeitsmenge innerhalb der noch zur Verfügung stehenden Zeit einen Personalaufbau notwendig macht. Da dann in der Regel kurzfristig keine know-how-spezifischen Softwareentwickler zur Verfügung stehen, holt man unerfahrene Mitarbeiter ins Projekt, womit der Kollaps vorprogrammiert ist. Verursacht wird der Projektverzug durch vielerlei zusätzliche Aufwände, wie:

- Wissensaufbau bezüglich Projekt und Tooling, womit die wichtigsten Mitarbeiter im Projekt von ihrer Arbeit abgehalten werden (bedeutendster Faktor)
- Anschaffung der Tools (Hardware und Software) für die hinzukommenden Team-Mitglieder
- Eingliederung der neuen Team-Mitglieder in die Organisations-Struktur – und damit in das etablierte Kommunikationsmuster (Mail-Verteiler, Einführung in die „heimlichen Spielregeln" [22]) des konkreten Projekts, usw.)
- Eingliederung ins Konfigurations-Management (Berechtigungen, Arbeitsverwaltung, usw.)

All diese zusätzlich notwendigen Aktivitäten bremsen den Projektfortschritt derart, dass eine sinnvolle Projektbeschleunigung nur durch die Eingliederung von Experten möglich wäre. Dies wiederum führt innerhalb eines Projektportfolios zur Forderung der möglichst schnellen Wissensweitergabe von den Experten an neue Mitarbeiter, um im Krisenfall das Know-how breit gestreut vorzufinden.

Mit Referenz auf die eigenen Erfahrungen sollten bereits zu Projektbeginn möglichst viele Mitarbeiter ins Projekt aufgenommen werden, wobei sich ein Teil der Experten der Einschulung der neuen Mitarbeiter widmet. Ein in der Praxis oft begangener Fehler in diesem Zusammenhang ist das Fehlen von im Projekt-Plan eingetragenen Arbeits-Paketen, die den Schulungsaufwand korrekt widergeben.

11.3.2.22 Phasen-Kopplung

Eine Zunahme der Entwicklungs-Geschwindigkeit erfordert das Abrücken von der strikten Phasen-Trennung. Während dieses Muster für kurze Dauer durchaus erfolgreich ist, wirkt es sich über längere Zeiträume (länger als 1-3 Monate), und vor allem bei größeren Projekten, eindeutig negativ aus. Sowohl der Prozess als auch die Organisation verlieren hierdurch an Struktur, wodurch ein Abgleiten ins Chaos unvermeidlich wird. Hauptursache dafür ist der mit der Verzahnung der Phasen einhergehende erhöhte Kommunikationsbedarf.

11.3.2.23 Problemdarstellung durch Szenarien

Ähnlich wie die Tätigkeiten zur Entwicklung von Software durch Muster darstellbar sind, kann auch das Problem bzw. die Anforderungen an die Software durch Szenarien (vgl. hierzu Abschnitt 6.4) beschrieben werden. Abstrakt betrachtet, wird ein Szenario aus den Grund-Bausteinen:

[22] Scott-Morgan, Peter, Little, Arthur D., Unwritten Rules of the Game, McGraw-Hill, 1994

- Objekt und
- Relation bzw. Kommunikation zwischen den Objekten

gebildet. Dieselbe Motivation liegt auch den objektorientierten Programmiersprachen zu Grunde. Bernd Österreich formuliert es so [23]): „Die herkömmlichen Entwicklungsmethoden versagen bei anspruchsvolleren Aufgaben, weil sie durch ihren Deduktivismus die Beschränktheit der menschlichen Kommunikationsmöglichkeiten nicht berücksichtigt haben, was nicht heißen soll, dass die OO-Methodik nicht versagen kann. Das objektorientierte Vorgehen öffnet sich aber dem Bewusstsein einer kommunikativen Rationalität und leitet somit zu einer anderen Anschauung der Welt über."

11.3.2.24 Produktverantwortung des Architekten

Äquivalent zur Verantwortung des Chefs für die Organisation und des Entwicklers für den Entwicklungsprozess, ist der Software-Architekt für die gegenwärtige und zukünftige Gestaltung des Softwareprodukts verantwortlich. Das Adjektiv „zukünftig" unterstreicht die in der Praxis übliche Weiterentwicklung von Softwareprodukten. Somit obliegt dieser Rolle die Steuerung und Beratung der Entwickler in softwaretechnologischen Angelegenheiten. Die Bedeutung der Architektur wird auch im Pattern „Entwicklungs-Standort-Verteilung als Faktor des Organisationsaufbaus" hervorgehoben.

Die Aufgaben des Software-Architekten sind ähnlich denen des Entwicklers im frühen Stadium des SEP mit besonderer Fokussierung auf:

- Verständnis der Anforderungen
- Kreation einer flexiblen und erweiterbaren Architektur

11.3.2.25 Prototypen

Unabhängig von der Entwicklungsmethodik ist der Bau eines Prototypen für all jene SPs zu empfehlen, in denen die Anforderungen, die Konsequenzen einer bestimmten Architektur, oder aber die zu verwendenden Tools ungenügend bekannt sind. Während durch Use-Cases die Kundenwünsche hinsichtlich Funktionalität weitgehend abschätzbar sind, kann z.b. die Einhaltung der geforderten Performance nur anhand von realisierter Software überprüft werden.

[23] Oesterreich, Bernd, Objektorientierte Softwareentwicklung, München, 1998

11.3.2.26 Stabile Basis

Durch die parallel stattfindende Entwicklung von mehreren Modulen mit meist unterschiedlichen Fertigstellungsterminen kann das gesamte Softwaresystem erst am Ende der Entwicklung zum Laufen gebracht werden. Um aus verschiedenen Gründen schon zu einem früheren Zeitpunkt ein lauffähiges System zu erhalten, empfiehlt sich die Speicherung einer jederzeit fehlerfrei compilierbaren und linkbaren Version. Selbst wenn von einzelnen Modulen nur die Schnittstelle implementiert ist, können dennoch einzelne Teile des Gesamtsystems in Betrieb genommen werden.

Die Verfügbarkeit eines lauffähigen Systems ist insbesondere in Phasen des Systemtests und der Weiterentwicklung notwendig, in denen man einzelne Module auf ihre Systemverträglichkeit hin testet. In diesen Fällen sollten alle nicht zum Test anstehenden Module keine Fehler erzeugen. Die Rolle, die diese Aufgabe wahrnimmt, ist jene des Konfigurations-Managers. Obwohl sie bereits in den meisten SPs installiert ist, unterscheiden sich die Projekte in der Frequenz mit welcher neue bzw. korrigierte Komponenten dem Gesamtsystem hinzugefügt werden.

11.3.2.27 Verknüpfung von Produkt- und Testdesign

Software zu testen erfordert zumindest den gleichen Aufwand wie sie zu implementieren. Aufbauend auf den Kundenanforderungen müssen durch das Test-Design alle Szenarien berücksichtigt werden. Der frühestmögliche Zeitpunkt für die Implementierung der Test-Software ist durch die Stabilität der Schnittstellen gegeben.

Die Diskussion, wie weit die Rolle des Software-Testers von jener des Software-Entwicklers getrennt sein sollte, hängt im wesentlichen von der Stufe des Tests ab. Während ein Modultest sich auf den Code eines Softwareabschnitts bezieht, nimmt der Systemtest die Perspektive des Kunden ein (vgl. hierzu Abschnitt 6.1.2).

12. Resümee

In „**Software Projekt Produkt Programm Management & Patterns (SP^{4}M)**“ werden die Prinzipien des Projekt-Managements auf die Software-Entwicklung in einem Softwarehaus angewandt. Dabei wird das Projekt als Teil eines umfassenderen Portfolios (Programms) gesehen und auf die technologisch bedingte enge Verwandtschaft zwischen einem Software-Projekt und einem Software-Produkt Bezug genommen. Während ein Software-Projekt eine große Entwicklungs-Mannschaft über Jahre binden kann, ist der Aufwand zur Massenproduktion bzw. Vervielfältigung (im Zusammenhang mit der Vermarktung eines Produkts) auf das Kopieren der Datenträger beschränkt. Zudem wird - davon unabhängig - unter einem Software-Produkt auch das Ergebnis eines Software-Projekts verstanden. Ein umfassendes Managementkonzept zur Software-Entwicklung wird im Projekt-Management-Leitfaden (vgl. Kapitel 4) dargestellt.

Für die Betrachtung eines Projekt-Programms sind die strategischen Überlegungen hinsichtlich der optimalen Ressourcenverteilung wesentlich (vgl. Kapitel/Abschnitt 3.2). Das Optimum hängt nicht zuletzt davon ab, inwieweit die Software-Leistungs-Merkmale bzw. Features in anderen Projekten wiederverwendbar sind, wodurch die Entwicklungskosten auf mehrere Kostenträger verteilt werden können. Voraussetzung zur Wiederverwendung ist eine gut dokumentierte, dem Entwicklungsprozess zugrundegelegte Struktur, anhand der man die Modul-Funktionalität während der Entwicklung (auch im Detail) nachvollziehen kann. Zu diesem Zweck wurden eine Vielzahl an Qualitäts-Management-Systemen entwickelt (vgl. Kapitel 8), die zumeist über das Surrogat eines messbaren Prozesses die Einhaltung der Produkt- und Teilprodukt-Qualität gewährleisten.

Um in der Praxis der Software-Entwicklung Prozessverbesserungen durchzusetzen, müssen diese schrittweise mit messbaren Teilergebnissen erfolgen. Zu diesem Zweck, ursprünglich jedoch auch um Lieferanten objektiv beurteilen zu können, wurde das CMM-Konzept entwickelt. Zusätzlich zur Qualitätsverbesserung soll es einen Produktivitätszuwachs fördern. Unabhängig davon ist die Einhaltung eines Mindest-Qualitäts-Standards erst die Basis zum sinnvollen Erfassen von Metriken – Maßen mit denen der Software-Prozess und das Software-Produkt quantitativ charakterisierbar sind. Die Schwierigkeit der Messung ist hierbei durch den raschen Fortschritt der Software-Technologie und die in der Regel nur beschränkt vergleichbaren Projekte zu erklären. Dies trifft auf die Qualitäts-Metriken, in viel größerem Maße jedoch, auf die Produktivitäts-Metriken zu.

Da der Lebenszyklus von Software-Produkten kontinuierlich kürzer wird und Eigenpersonal zugunsten von Fremdpersonal reduziert wird, gewinnt der Prozess zur Einbindung von OEM-Software seit einigen Jahren zunehmend an Bedeutung. Damit werden im Gegensatz zur Eigenentwicklung zusätzliche Kenntnisse benötigt – insbesondere hinsichtlich der Software-Produkt-Auswahl und der Vertragsgestaltung (vgl. Kapitel/Abschnitt 5.5). Darüber hinaus ändern sich die Software-Requirements seitens des Kunden mit zunehmender Häufigkeit, wodurch es notwendig wird, den Kunden während der gesamten Software-Entwicklung miteinzubinden (vgl. u.a. Kapitel/Abschnitt 4.6).

Aufgrund der stark variierenden Produktivität ist ein Software-Projekt zu Beginn nur mit großen Ungenauigkeiten bezüglich des Aufwands abzuschätzen. Mit Fortschreiten der Entwicklungstätigkeit werden zwar die Schätzungen stets genauer, die Möglichkeiten zur Projektgestaltung hingegen immer geringer. Diesem Umstand wird durch die Aufwandsabschätzung auf Basis der zu realisierenden Features (bzw. Leistungsmerkmale) Rechnung getragen (vgl. Kapitel/Absatz 7.1.3). Des weiteren muss während des gesamten Projektverlaufs eine Aussage über den Sachfortschritt möglich sein. Dies führt zur Problematik der Bezugsgröße der Fortschritts-Messung. Als Lösung wird hierzu das Verfahren der kleinsten Arbeitspakete nach Exl vorgestellt (vgl. Kapitel/Abschnitt 7.2.2.1). Dadurch wird die schwierige Beurteilung des Fertigstellungsgrades vermieden, welche darin besteht, dass oft 80% des Aufwands für die zuletzt realisierten 20% des Programm-Codes erforderlich sind (Pareto-Prinzip; vgl. Kapitel/Abschnitt 10.4.2).

Trotz aller Vorbereitungen und Maßnahmen für eine qualitativ hochwertige Software-Entwicklung ist ein Software-Projekt durch eine Vielzahl an Risken bedroht. Zur Bewältigung dieser müssen anhand einer strukturierten Vorgangsweise (Prozess) die Erfolgsfaktoren definiert und deren Schwachpunkte (Risken) analysiert werden. Was die Verbesserung des Risiko-Management-Systems betrifft, muss dieses ebenso wie das QM-System in den Entwicklungsprozess eingebunden sein. Um bei der Vielfalt an Gestaltungsparametern den zugrundeliegenden Entwicklungsprozess auf seine wesentlichen Gestaltungsfaktoren zu reduzieren, dient das aus dem Deming-Regelkreis weiterentwickelte 7-Disziplinen-Modell nach Exl (vgl. Kapitel/Abschnitt 10.2.3.1).

Aufgrund des hohen Faktoranteils an Personalressourcen in der Software-Entwicklung ist eine adäquate personelle Organisationsgestaltung von fundamentaler Bedeutung. Hierbei müssen die Interessen aller Stakeholder berücksichtigt werden, wobei sie weniger als Risken, sondern vielmehr als Gestaltungs-Faktoren in die Projektplanung und Realisierung miteinfließen sollten (vgl. Kapitel/Abschnitt 9.4).

Bei Erstellung der Spezifikation (= Pflichtenheft) ist darauf zu achten, dass Anwender und Entwickler eine identische Vorstellung über die Geschäftsprozess-Implementierung haben. Hierbei ist die Kommunikation zwischen Entwickler und Anwender der kritische Erfolgsfaktor, damit die Requirements (in funktionaler, qualitativer, terminlicher und budgetärer Hinsicht) korrekt realisiert werden. Zu diesem Zweck eignet sich die Unified Modeling Language in optimaler Weise. – Wesentliches Instrument sind hierbei Graphiken, die die Zusammenhänge in unterschiedlicher Weise abbilden (z.b. hierarchische Objektstruktur, Sequenzdiagramme, uvm.; vgl. Kapitel/Abschnitt 6.4).

Alle bisher genannten erfolgskritischen Thematiken, die in den Kapiteln 2 bis 10 detailliert dargestellt werden, sind in der Hektik der Praxis nur selten gleichermaßen berücksichtigbar. Sehr oft wird der eine oder andere Erfolgsfaktor zugunsten der Lösung eines akuten Problems vernachlässigt. Um in diesen (in der Praxis recht unterschiedlich in Erscheinung tretenden) Szenarien den besten Weg der Projekt-Realisierung zu finden, wird im Kapitel 11 eine neue Variante der Darstellung des Erfahrungs-Wissens vorgeschlagen, wie sie in der Literatur des Projekt-Managements bisher noch nicht erwähnt wurde – das Schema des Pattern.

In Form eines einfachen Schemas werden Probleme (mit ihren Randbedingungen) und den zugehörigen Lösungen beschrieben. Hierbei können die Probleme beliebigen fachlichen Disziplinen zuordenbar sein. Voraussetzung ist, dass sich die Lösungen der Probleme in der Praxis bewährt haben und somit ohne großes Risiko anwendbar sind. Sie dienen im Gegensatz zu hochentwickelten QM-Systemen nicht der Perfektionierung der einen oder anderen Methode, sondern sollen verhindern, dass grundlegende Fehlentscheidungen wiederholt getroffen werden. Dadurch bieten sie im Fall, dass das identifizierte Problem bereits in Form eines Pattern analysiert und beschrieben wurde, eine Lösungshilfe zur Entscheidungsfindung. Ein in letzter Zeit, auf diesem Grundsatz basierendes, in den USA neu entwickeltes Modell ist „Extreme Programming" (vgl. Kapitel/Abschnitt 6.1.4). Diesem Ansatz zum Software-Engineering liegen Patterns zugrunde, die aufgrund von mehrjährigen einschlägigen Erfahrungen aus der Programmierpraxis gewonnen wurden.

Besonders begünstigt wird die Wissensaufbereitung in Form von Patterns durch die mittlerweile durchgehende innerbetriebliche Datenvernetzung und Nutzung der HyperText Markup Language zur Verknüpfung von Information.

13. Anhang

13.1 Key Practices der einzelnen Key Process Areas des SEI-CMM (Ergänzung zu Kapitel 8, Abschnitt 8.2.4)

Im folgenden wird eine Übersicht über die Key Practices der einzelnen Key Process Areas des SEI-CMM gegeben [1]). Da die Zählweise der Stufen bei „1" beginnt, existieren erst ab Level 2 Kriterien.

13.1.1 CMM-Level 2: „Repeatable"

13.1.1.1 Allgemeine Merkmale von Level 2:

- „Intuitiv"
- Anforderungen sind festgeschrieben und abgestimmt
- Definition von Standards und Richtlinien sowie Überwachung ihrer Einhaltung
- Anwendung von Methoden, Techniken und Verfahren, die in vergleichbaren früheren Projekten erfolgreich eingesetzt wurden
- Schätzungen auf der Basis von Erfahrungswerten
- Projektverfolgung bezüglich Terminen, Kosten und Qualität

13.1.1.2 Zu erfüllende Kriterien in den einzelnen Key Process Areas von Level 2:

Level 2.1: Anforderungsklärung und –definition:

- Vereinbarungen zwischen Auftraggeber und Auftragnehmer
- Requirements enthalten Anforderungen an Produkt
- Requirements enthalten Anforderungen an Prozess (z.b. Termine)
- Anforderungen sind Grundlage für Planung und Durchführung des Projekts
- Anforderungen erfüllen bestimmte Kriterien: klar, eindeutig, überprüfbar, dokumentiert
- Gesamtheit der Anforderungen = Anforderungskatalog
- Umsetzung der Anforderungen in Pläne und Ergebnisse, insbesondere Entwurf
- Review und Abnahme des Anforderungskatalogs durch SW-Entwicklungs-Team
- Anwendung eines geeigneten CR-Verfahrens für Anforderungen, insbesondere Prüfung der Auswirkungen neuer Anforderungen und Ermittlung davon betroffener Ergebnisse

[1] Siemens AG, Zentralabteilung Technik, 1998

Level 2.2: Projektplanung:

- Ermittlung aller Arbeitspakete zur Umsetzung der Anforderungen
- Realistische Schätzung der Mengen der Ergebnisse von Arbeitspaketen nach einem *festgelegten Verfahren*
- Realistische Schätzung der Aufwände für die Arbeitspakete nach einem festgelegten Verfahren
- Festhalten der Randbedingungen, Einflussfaktoren und Vereinbarungen für die Schätzung
- Dokumentation der Pläne als Basis für die Ausführung der Arbeiten und für die Überwachung des Projektfortschritts sowie für Vereinbarungen mit dem Auftraggeber
- Projektplanung als projektbegleitender, permanenter Prozess mit Schwerpunkt in den ersten Phasen
- Anwendung eines dokumentierten Verfahrens für die Projektplanung
- Einsatzmittelplanung für Tools, Hilfsmittel, Einrichtungen usw.
- Ermittlung von Risken bezüglich Technik, Kosten, Kapazität und Terminen, incl. Abschätzung ihrer Auswirkungen

Level 2.3: Projektverfolgung:

- Vergleich der aktuellen Ist- mit den dokumentierten Plan-Werten (bezüglich Ergebnissen und Produktivität)
- Aktualisierung der Pläne auf der Basis der jeweiligen Ist-Situation
- Dokumentation der Ist-Werte, Soll-/Ist-Vergleiche und Planaktualisierungen
- Steuerungsmaßnahmen bei Planabweichungen bez. Ergebnissen, Kosten u. Terminen
- Kontrolle des Projektfortschritts anhand definierter Meilensteine
- Überwachung des Projektfortschritts (Erledigung der Arbeitspakete) durch Projektleiter und Management
- Berücksichtigung von CRs in einem geregelten Verfahren, d.h. nach Abstimmung unter allen Betroffenen
- Verfolgung der technischen, Kosten-, Kapazitäts- und Terminrisken

Level 2.4: Lieferantenmanagement:

- Festlegung der in Auftrag zu gebenden Arbeiten
- Auswahl von Unterauftragnehmern (UAN) entsprechend ihrer Eignung zur Erfüllung der Aufgaben nach einem festgelegten Verfahren (incl. Assessment)
- Treffen von schriftlichen Vereinbarungen zwischen Auftragnehmer und UAN über die zu erfüllenden Aufgaben unter Berücksichtigung der entsprechenden Anforderungen und unter Angabe relevanter Restriktionen (z.b. Zeit, Kosten) als Basis der Zusammenarbeit
- Koordination/Abstimmung der Aktivitäten des UAN (nach dessen eigenen Plänen) mit den anderen Aktivitäten
- Verfolgung und Kontrolle der Ergebnisse und Leistungen des UAN auf der Basis der schriftlichen Vereinbarungen und der Pläne des UAN
- Änderungen (der Aufgabenstellung, der Randbedingungen oder anderer Vereinbarungen) des Vertrages entsprechend einer festgelegten Prozedur unter Beteiligung aller Betroffenen des Auftragnehmers und des UAN
- Regelmäßige Besprechungen von Management und Ausführenden (technische Aspekte) von Auftragnehmer und UAN
- Überprüfung der Aufgabenerfüllung und der Ergebnisse des UAN anhand bestimmter Meilensteine

- Überwachung der QM-Aktivitäten des UAN durch die QS des Auftragnehmers nach einem festgelegten Verfahren
- Überwachung der KM-Aktivitäten des UAN durch das KM des Auftragnehmers nach einem festgelegten Verfahren
- Abnahme der Ergebnisse des UAN durch den Auftragnehmer entsprechend eines festgelegten Verfahrens (incl. Abnahmetest)
- Periodische Wiederholung der Assessments des UAN

Level 2.5: Qualitätssicherung:

- Erstellung eines QS-Plans mit den durchzuführenden QS-Maßnahmen
- Teilnahme der QS-Stelle an der Ausarbeitung und Verabschiedung von Plänen, Standards und Richtlinien sowie Verfahrensbeschreibungen
- Durchführung von Reviews von Ergebnissen zur Sicherstellung der geforderten Qualität (d.h. Erfüllung der festgelegten Kriterien) der Ergebnisse
- Berichterstattung über Prüfergebnisse an Entwicklung und Management
- Dokumentation von Abweichungen/Fehlern und Bearbeitung entsprechend eines festgelegten Verfahrens
- Überprüfung der QS-Aktivitäten und der QS-Ergebnisse (Prüfergebnisse, QS-Berichte)
- QS-Funktion ist unabhängig von Entwicklung und Projekt-Management

Level 2.6: Konfigurationsmanagement:

- Erstellung eines KM-Plans (KM-Verfahrensbeschreibung) als Basis für die Durchführung der KM-Aktivitäten
- Einrichtung einer Projektbibliothek (DV-gestützt) zur Speicherung der Ergebnisse
- Bestimmung der Konfigurationen (Identification)
- Bearbeitung von CR/FM entsprechend eines festgelegten Verfahrens (Control)
- Verfolgung von Änderungen an Konfigurationen entsprechend eines festgelegten Verfahrens (Audit): Entwicklungsschritte und CR/FM
- Überprüfung der Ergebnisse auf Vollständigkeit
- Freigabe von Baseline-Ergebnissen (i.d.R. Meilensteine) entsprechend eines festgelegten Verfahrens
- Dokumentation des Status von Konfigurationen bzw. von CR/FM entsprechend eines festgelegten Verfahrens (Accounting)
- Dokumentation der KM-Aktivitäten und eindeutige Beschreibung der Baseline-Ergebnisse sowie Verteilung der entsprechenden Informationen an Betroffene

13.1.2 CMM-Level 3: „Defined"

13.1.2.1 Allgemeine Merkmale von Level 3:

- "Qualitativ"
- Prozessmodell ist institutionalisiert (als Standard im Bereich) und wird nach individueller Anpassung in den Projekten eingesetzt
- Integration von Software-Engineering- und Management-Techniken
- Ständige Prozessbetrachtung durch "Process Group"
- Gezielte Steuerung von Kosten, Terminen, Funktionalität und Qualität

13.1.2.2 Zu erfüllende Kriterien in den einzelnen Key Process Areas von Level 3:

Level 3.1: Prozess-Fokus (auf Bereichsebene):

- Entwickeln eines Verständnisses über Abläufe und Zusammenhänge bei der Software-Entwicklung, insbesondere Erkennen der Stärken und Schwächen (= Reflexion über Projektablauf)
- Ermittlung und Koordination von Aktivitäten zur Beschreibung und Verbesserung der Abläufe und Zusammenhänge, insbesondere systematisches Begegnen der Schwächen
- Einrichtung einer "Process Group" als zentrale Anlaufstelle für alle prozessrelevanten Fragen, die für die Erstellung eines Standard-Software-Prozessplans, dessen Anwendung und permanente Verbesserung zuständig ist
- Periodische Beurteilung des SW-Entwicklungsprozesses
- Verabschiedung eines Plans zur Erstellung und Verbesserung des Entwicklungs-Prozessplans
- Zentrale Koordination aller Aktivitäten bezüglich Definition, Umsetzung, Messung und Verbesserung des Entwicklungsprozesses, incl. der entsprechenden Datenhaltung
- Bewertung neuer Methoden und Techniken und Einsatz entsprechend ihrer Eignung
- Schulung der Prozesstechnologie für Software-Entwicklung, die im Bereich oder im Projekt Verwendung findet
- Beteiligung bzw. Information aller Betroffenen an bzw. über Aktivitäten zur Definition oder Verbesserung des Prozessplans

Level 3.2: Prozessdefinition (auf Bereichsebene):

- Erstellung eines Standard-Software-Prozessplans für den Einsatz in den SW-Projekten des entsprechenden Bereichs als einheitliche Grundlage aller Entwicklungs- und Wartungs-Projekte zur Gewährleistung und Verbesserung der Produktivität
- Definition der wesentlichen Prozess-Schritte (Phasen) und Errichtung einer Grundlage für Analysen und Messungen der "Performance" des Prozesses
- Entwicklung des Standard-Prozessplans entsprechend eines festgelegten Verfahrens (vgl. L3.1) und unter Berücksichtigung der spezifischen und eingeführten Bereichs-Standards
- Benutzung von bewährten SW-Life-Cycle-Modellen und von in früheren Projekten verwendeten Prozessbeschreibungen

- Einrichtung und Nutzung einer Datenbank für Zwecke der Erstellung und Anwendung des Prozessplans

Level 3.3: Trainingsprogramm:

- Identifikation des Schulungsbedarfs für die Mitarbeiter gemäß den in Projekten auszuführenden Aufgaben und den dazu erforderlichen Kenntnissen und Fähigkeiten
- Entwicklung eines Schulungsprogramms entsprechend des Bedarfs und periodische Anpassung des Programms; beides entsprechend festgelegter Verfahren
- Entwicklung oder Beschaffung von Kursen zur Abdeckung des Schulungsbedarfs unter Berücksichtigung der Besonderheiten des Bereichs
- Feststellung, inwieweit die in den Kursen vermittelten Kenntnisse und Fähigkeiten bei den einzelnen Mitarbeitern schon vorhanden sind
- Durchführung der Kurse zur Vermittlung der erforderlichen Kenntnisse und Fähigkeiten, sowie um die Mitarbeiter in die Lage zu versetzen, die vorhandene oder geplante Arbeitsumgebung effektiv zu nutzen
- Erstellung von Protokollen über durchgeführte Trainingsmaßnahmen

Level 3.4: Integrierter SW-Entwicklungs- und Management-Prozess:

- Projektspezifische Anpassungen des Standard-Software-Prozessplans ("Tailoring") unter gleichzeitiger Berücksichtigung des Technischen (Entwicklungs-) und des Management-Prozesses (als Grundlage der Projektdurchführung)
- Bestimmung der Aktivitäten zum Management (insb. Schätzungen, Planung, Verfolgung, Steuerung) eines Projektes auf der Basis des Standard-Prozessplans
- Berücksichtigung der besonderen Eigenschaften des Projektes bei der personellen Besetzung, beim Training der Mitarbeiter und beim Management des Projektes
- Nutzung von Erfahrungswerten (Prozessdatenbank) aus anderen - früheren und aktuellen - Projekten bei der Planung und Verfolgung der Projekte
- Schätzung des Umfangs der zu entwickelnden Software nach einem festgelegten Verfahren, das auch Überprüfungen und Aktualisierungen der Schätzungen sowie die Überprüfung und Anwendung des Verfahrens selbst durch dafür jeweils zuständige Stellen umfasst
- Planung und Überwachung von Kosten auf der Basis von Arbeitspaketen und Phasen nach einem festgelegten Verfahren
- Planung und Überwachung der kritischen Ressourcen nach einem festgelegten Verfahren, das auch die Überprüfung und Aktualisierung der Pläne umfasst
- Ermittlung und Ausweis kritischer Abhängigkeiten zwischen Arbeitspaketen und des kritischen Pfades sowie Verfolgung der entsprechenden Arbeitspakete nach einem festgelegten Verfahren
- Identifikation, Bewertung, Dokumentation, Planung und Überwachung von Risken nach einem festgelegten Verfahren
- Periodische Durchführung von Projektanalysen, um die Projektabwicklung an die jeweils aktuellen (ursprünglich festgelegten oder infolge einer neuen Marktsituation bzw. anderer Kundenwünsche geänderten) Projektziele anzupassen

Level 3.5: Systematischer Software-Erstellungs-Prozess:

- Ermittlung, Analyse, Konkretisierung und Dokumentation der Software-Anforderungen (insb. hinsichtlich Funktionalität, Leistung und Schnittstellen) als Ausgangsbasis der Entwicklung

- Erstellung, Dokumentation und Prüfung einer SW-Architektur und eines SW-Designs (Grob- und Feinentwurf) auf der Basis der Anforderungen und als Grundlage für die Codierung
- Codierung und Test der Komponenten auf der Basis von SW-Architektur und -Design
- Test des Produktes (i.S.v. System- und Abnahmetest) auf Erfüllung der definierten Anforderungen auf der Basis eines dokumentierten und geprüften Testplans
- Review und Abnahme der Dokumentation für Betrieb und Wartung des Systems
- Gewährleistung der Konsistenz aller Aktivitäten zur SW-Erstellung über alle Phasen hinweg (vor allem Konsistenz zwischen Aktivitäten zur Planung und zur Ausführung von Entwicklungs- und Testaufgaben), insbesondere durch Dokumentation der Ergebnisse, Einbeziehung des Konfigurationsmanagements und ein effizientes Änderungswesen
- Flexible und kontrollierte Anpassung von Anforderungen, Design und Code an Änderungen während der Entwicklung
- Sammlung und Analyse von Fehlerdaten aus Reviews und Tests nach einem festgelegten Verfahren
- Verwendung geeigneter, aktueller, effizienter Tools und Methoden für Software-Entwicklung und -Test bei der Ausführung jedes Entwicklungsschritts
- Schulung der Entwickler hinsichtlich der Methoden, Tools, Standards und Richtlinien, die für den jeweiligen Tätigkeitsbereich (z.b. Anforderungsanalyse, Entwurf, Codierung, Test) maßgeblich sind

Level 3.6: Interdisziplinäre Zusammenarbeit:

- Gemeinsame Erarbeitung der Anforderungen durch alle am Projekt beteiligten Stellen (incl. Auftraggeber)
- Beteiligung aller Stellen an Arbeiten auf Systemebene (Anforderungsklärung, Aufteilung der Anforderungen auf Subsysteme, Überprüfung von Architektur und Design durch Reviews, Steuerung von Änderungen)
- Kenntnis der Anforderungen bei allen Projekt-Mitarbeitern
- Kenntnis der Zuständigkeiten einer jeden Stelle bei den jeweils anderen Gruppen (incl. Kenntnis der jeweils verwendeten Prozesse, Methoden und Standards)
- Definition der Schnittstellen zwischen den verschiedenen Projektstellen, incl. Review der Ergebnisse von übergebenden Stellen durch die übernehmenden Stellen
- Anwendung von Gruppenarbeits-Techniken (incl. Training der Führungskräfte in diesen Techniken)
- Laufende Kommunikation zwischen den Projektbeteiligten

Level 3.7: Reviews:

- Review-Planung: Festlegung aller einem Review zu unterziehenden Ergebnisse
- Festlegung konkreter Review-Verfahren für die Prüfung der Ergebnisse, incl. Angaben für Vorbereitung und Durchführung der Reviews, Ermittlung und Bewertung der gefundenen Fehler, Freigabe der Ergebnisse
- Dokumentation und Verfolgung der infolge von Reviews durchzuführenden Aktivitäten bis zu ihrem Abschluss
- Spezielle Schulung von Reviewern und Moderatoren von Review-Sitzungen

13.1.3 CMM-Level 4: „Managed"

13.1.3.1 Allgemeine Merkmale von Level 4:

- "Quantitativ"
- Metriken zur Messung von Produktivität und Qualität
- Bewertung von Projekten (d.h. Produkte und Prozesse) auf der Basis dieser Metriken
- Erkennen von Abweichungen in Leistung und Produktivität
- Treffen von Korrekturmaßnahmen bei Abweichungen
- Kenntnis und Beherrschen von Projektrisken
- Relativ genaue Vorhersage von Projektverlauf und Produktivität

13.1.3.2 Zu erfüllende Kriterien in den einzelnen Key Process Areas von Level 4:

Level 4.1: Quantitative Prozess-Steuerung:

- Festlegung der Zielgrößen für die Messungen (insb. Produktqualität, Produktivität, Entwicklungszeiten) entsprechend ihrer Bedeutung für Projekte und den Bereich
- Festlegung der entsprechenden Aktivitäten, incl. der Verantwortlichen und der dafür einzusetzenden Werkzeuge, sowie Abschätzung des Zeitbedarfs für die Messungen und Analysen
- Ermittlung der entsprechenden Daten und Selektionen auf der Basis der Hauptaktivitäten, der wichtigsten Ergebnisse und der Meilensteine des Standard-Prozessplans
- Speicherung der ermittelten Daten für Auswertungen und zur Verwendung in zukünftigen Projekten, z.b. zur Planung
- Statistische Analyse der ermittelten Daten, inklusive Mittelwert und Varianz bei jeder Kennzahl
- Ermittlung quantitativer Abhängigkeiten zwischen Produktqualität, Produktivität, und Entwicklungszeiten

Level 4.2: Qualitätsmanagement:

- Festlegung und Priorisierung messbarer Qualitätskriterien für Produkt- und Prozessqualität, gemeinsam mit Auftraggeber und Endnutzer
- Nutzung der Prozessmessungen, um eine quantitative Basis für das Qualitätsmanagement zu erhalten
- Gestaltung von Plänen, Design und Prozess entsprechend der festgelegten Qualitätskriterien, um Produkt- und Prozessqualität mit großer Zuverlässigkeit vorhersagen und realisieren zu können
- Einbeziehung der Unterauftragnehmer: Weitergabe der Qualitätskriterien bezüglich Produktqualität
- Festlegung und Verfolgung quantitativer Qualitätskriterien für:
    - Requirements
    - Entwurf
    - Code

    - Tests
- Vergleiche der tatsächlichen Qualität der Ergebnisse mit den Qualitätskriterien des jeweiligen Ergebnisses
- Bei Abweichungen in Produkt- und Prozessqualität: Analyse der Abweichungsursachen und Treffen von Korrekturmaßnahmen

13.1.4 CMM-Level 5: „Optimizing"

13.1.4.1 Allgemeine Merkmale von Level 5:

- "Kontinuierliche Prozessverbesserung"
- Problemanalyse: Identifikation von Problem-/Schwachstellen im Prozess
- Behebung und Vermeidung von Fehlern und Problemen (Defect Prevention)
- Gezielte Verwendung des statistischen Datenmaterials aus den Prozessmessungen für Verbesserungen des Prozesses (Feedback)
- Prüfen neuer Ideen und Technologien bezüglich SW-Entwicklung auf deren Eignung zur Verbesserung des Prozesses
- Systematische Einführung neuer Technologien (Innovationen)

13.1.4.2 Zu erfüllende Kriterien in den einzelnen Key Process Areas von Level 5:

Level 5.1: Defect Prevention:

- Erkennen und Analyse von - wiederholt auftretenden - Fehlern in Projekten
- "Kickoff-Meeting" der Team-Mitglieder zu Beginn von Aufgaben zur Vorbereitung auf die Ausführung mit besonderer Berücksichtigung von Fehlervermeidungsaspekten
- "Causal Analysis-Meetings" während und nach der Aufgabenausführung zur Feststellung von Problemen und Schwachstellen sowie zur Analyse ihrer Ursachen und Auswirkungen
- Ermittlung von Maßnahmen, um zukünftiges Auftreten der Fehler zu vermeiden
- Überarbeitung von standard- und projektspezifischen Prozessplänen entsprechend den vorgeschlagenen Maßnahmen
- Einrichtung von Teams auf Bereichs- und Projektebene, die für Defect-Prevention-Aufgaben verantwortlich sind
- Einrichtung einer Datenbank für Defect-Prevention-Aufgaben

Level 5.2: Management des Technologiewechsels:

- Ermittlung neuer Technologien, die für Zwecke des Bereichs und dessen Projekte in Frage kommen
- Vergleich der neuen Technologien mit den im eigenen Bereich eingesetzten Technologien
- Analyse des Standard-Prozessplans im Hinblick auf Bereiche, in denen der Einsatz neuer Technologien, Methoden und Tools den größten Nutzen bringt

- Bewertung neuer Technologien im Hinblick auf deren Nutzen beim Einsatz im eigenen Bereich zur Erhöhung der Produktivität und Verbesserung der Qualität
- Auswahl und Beschaffung neuer Technologien entsprechend eines festgelegten Verfahrens, das *Anforderungen und Nutzen berücksichtigt*
- Piloterprobung der neuen Technologien vor ihrer breiten Einführung
- Einführung der geprüften, geeigneten neuen Technologien und Berücksichtigung in standard- und projektspezifischen Prozessplänen entsprechend eines festgelegten Verfahrens
- Einrichtung einer "Technology Support Group", die für Ermittlung, Bewertung, Auswahl und Einführung neuer Technologien zuständig ist
- Aufstellen eines Plans für die Einführung neuer Technologien, der auch strategische Aspekte der Prozessverbesserung beinhaltet
- Unterstützung des Managements bei der Einführung der Technologien

Level 5.3: Management der Prozessverbesserung:

- Festlegung quantitativer, messbarer Ziele für Prozessverbesserungen und Verbesserungsschritte (gerichtet auf Verbesserungen der Produktqualität und Erhöhung der Produktivität) unter aktiver Beteiligung von Mitarbeitern und Management
- Koordination aller Aktivitäten zur Prozessverbesserung durch die "Process Group"
- Erstellung und regelmäßige Überprüfung geeigneter Trainingsprogramme und Anreizsysteme für Mitarbeiter und Management zur Begünstigung der schrittweisen Prozessverbesserung (z.b. Belohnung für die Abgabe von bedeutsamen Verbesserungsvorschlägen)
- Abgabe von Verbesserungsvorschlägen durch einzelne Mitarbeiter oder Teams nach einem festgelegten Verfahren
- Prüfung, Genehmigung, Planung, Umsetzung und Verfolgung von Verbesserungsvorschlägen nach einem festgelegten Verfahren, das auch zu erwartende Nutzen der Vorschläge und deren Priorisierung umfasst
- Bildung von Arbeitsgruppen, Qualitätszirkeln u.ä. unter aktiver Beteiligung von Mitarbeitern und Management mit dem Ziel, Verbesserungen für jeweils konkrete Bereiche des Entwicklungsprozesses zu erarbeiten
- Einarbeitung von Prozessverbesserungen, über deren Einführung nach Abschluss aller zu ihrer Entstehung vorgenommenen Arbeiten (incl. eventueller Piloterprobungen) entschieden worden ist, in standard- und projektspezifische Prozesspläne nach einem festgelegten Verfahren
- Regelmäßige Information (Feedback) der Mitarbeiter und des Managements über Status und Ergebnisse der Prozessverbesserungsschritte

14. Literaturverzeichnis

Alexander, Christopher, Notes on the Synthesis of Form, Harvard University Press, 1964

Alexander, Christopher, The Timeless Way of Building, New York, 1979

Alexander, Christopher, et al., A Pattern Language, New York, 1977

Alexander, Christopher, The Nature of Order, 4-bändig, 2003

Alexander, Christopher, The Timeless Way of Building, Oxford University Press, 1979

Al-Laham, A., Strategieprozesse in deutschen Unternehmungen. Verlauf, Struktur und Effizienz, Wiesbaden, 1997

Ansoff, H.I., The New Corporate Strategy, New York, 1988

Appleton, Brad, Patterns and Software: Essential Concepts and Terminology, http://www.bradapp.net/ , 2000

Balzert, Helmut, Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Heidelberg, 1998

Balzert Helmut, Lehrbuch der Software Technik: Software-Entwicklung, Heidelberg, 1996

Barny, J.B., Gaining and Sustaining Competitive Advantage, New York, 1997

Basili, V. R., Applying the Goal / Question / Metric Paradigm in the Experience Factory, in: Software Quality Assurance & Metrics, Chapman & Hall, London, 1994

Boehm, B.W., Verifying and Validating Software Requirements and Design Specifications, in: IEEE Software, 1984, p. 75-88

Boehm, B.W., Improving Software Productivity, in: IEEE Computer, 1987

Boehm, B.W., A Spiral Model of Software Development and Enhancement, in: Computer, May 1988, p. 61-72

Boehm, B.W., Using the Win Win Spiral Model: A Case Study, IEEE Computer, July 1998, p. 33-44

Boehm, B.W. & Bose, P., A Collaborative Spiral Software Process Model Based on Theory W, Proceedings, ICSP 3, IEEE, October 1994

Brooks, F.P., The mythical Man-Month, 1975

Bröhl, A.-P., Dröschel, W. (hrsg.), Das V-Modell, München, 1993

Bruhn, M. (hrsg.), Balanced Scorecard: Wertorientierte Unternehmensführung, Wiesbaden, 1998

Burghardt, Manfred, Einführung in Projektmanagement, Publics-MCD-Verl., München, 1995

Charette, Robert N., Large-Scale Project Management is Risk Management, in: IEEE, July 1996, p. 110-117

Clark, B., Boehm, B.W. (hrsg.), Knowledge Summary: Focused Workshop on COCOMO 2.0, USC-CSE, 1995

Cleland, David I. (hrsg.), Field Guide to Project Management, USA, 1998

Copeland, T.E., Koller, T., Murrin, J., Valuation: Measuring and Managing the Value of Companies, New York, 1994

Coplien, James O., On the Nature of The Nature of Order, http://www1.bell-labs.com/user/cope , 1997

Coplien, James O., Software Patterns, http://www1.bell-labs.com/user/cope/Patterns/WhitePaper/ , 2002

Coplien, James O., A Development Process Generative Pattern Language, AT&T Bell Laboratories, http://www.bell-labs.com/~cope/Patterns/Process/index.html , 1995

Coplien, James O., The Human Side of Patterns. C++ Report, 1996

Coplien, James O., Software Patterns, http://www1.bell-labs.com/user/cope/Patterns/WhitePaper/ , 2002

Cusumano, Michael A., Harvard Business Manager 3/98: Gipfelstürmer Microsoft, Massachusetts, 1998, p.55-66

Dangel P., Hofstetter U., Otto P., Analyse von Jahresabschlüssen nach US-GAAP und IAS, 2001

Darwin, Charles R., On the Origin of Species by Means of Natural Selection, London, 1859

DeMarco, Tom, Lister, Timothy, Wien wartet auf Dich! Der Faktor Mensch im DV-Management, München, 1999

Deming, Edwards, Out of the Crisis, Cambridge, MA, USA, 1986

Department of Defense, Software Development and Documentation, MIL-STD-498, AMSC no. N7069, 1994

Duden, 1996

Ebers, M., Gotsch, W., Institutionenökonomische Theorien der Organisation, in: Organisationstheorien, hrsg. v. A. Kieser, Stuttgart-Berlin, 1999

Ebert, Christof, Dumke, Reiner, (hrsg.), Software-Metriken in der Praxis, Berlin, 1996

Europäische Norm EN ISO 9000-3: 1997

Florac, William A., Carleton, Anita D., Measuring the Software Process, SEI Series, Massachusetts, 1999

Feyhl, Achim W., Feyhl, Eckhardt, Management und Controlling von Softwareprojekten, Wiesbaden, 1996

Gabler-Wirtschafts-Lexikon, Wiesbaden, 1997

Gamma, B., et al., Design Patterns: Elements of Reusable Object-Oriented Software, 1995

Gartner Group, The Software Engineering Strategies Scenario, Stanford, 1990

Garvin, D.A., Managing Quality: The Strategic and Competitive Edge, New York, 1988

Gassmann, Oliver, Internationales F+E Management, München, 1997

Götz, Herbert, Entwicklung eines Modells eines konvergenten Informations- und Kommunikationssystems an Hand eines internationalen Unternehmens der Telekommunikationsindustrie, Dissertation an der TU-Wien, 2002

Grady, R.B., Practical Software Metrics for Project Management and Process Improvement, 1992

Grady, R.B., Caswell D.L., Software Metrics: Establishing a Company-Wide Program, 1987

Gray, Lewis, in: Crosstalk The Journal of Defense Software Engineering, Sept. 1998

Grove, Andrew S., Nur die Paranoiden überleben, Frankfurt/Main, 1997

Hall, Elaine M., Managing Risk, USA, 2001

Halstead, M.H., Elements of Software Science, New York, 1977

Halstead, Maurice H., Elements of Software Science, New York, 1977

Hansen, Wilfried J. (hrsg.), Spiral Development: Experience, Principles, and Refinements, Special Report, CMU/SEI-2000-SR-008, 2000

Henderson, Bruce D., Oetinger, Bolko von, (hrsg.), Das Boston Consulting Group Strategie-Buch, Düsseldorf, 2000

Heß, G., Marktsignale und Wettbewerbsstrategie, Stuttgart, 1991

Hickersberger, Arnold, Der Weg zur objektorientierten Software, Heidelberg, 1994

Holmes, Lori A., Evaluating COTS Using Function Fit Analysis, in: CrossTalk, Vol. 13 No. 2, Febr. 2000, p. 26-30

Horx, Matthias, Die acht Sphären der Zukunft, Wien, 2000

Hostettler, S., Das Konzept des Economic Value Added (EVA), Bern, 1997

Huber, R., Überwindung der strategischen Diskrepanz und Operationalisierung der entwickelten Strategie, St. Gallen, 1985

Humphrey, Watts S., Managing the Software Process, Addison-Wesley, Reading, Massachusetts, 1989

Humphrey, Watts S., A Discipline for Software Engineering, Addison-Wesley, Reading, Massachusetts, 1995

Hull, John C., Options, Futures, and other Derivatives, Toronto, 1997

Hüttemann, H.H., Anreizsysteme in schrumpfenden Unternehmungen, Wiesbaden, 1993

IBM, Die Function Point Methode, IBM Deutschland GmbH, 1985

IEEE, A methodology for collecting valid software engineering data, IEEE Vol. SE-10, 6, 1984, p. 728-738

Jeffries, Ronald E., The Four Project Values, http://www.xprogramming.com/Practices/PracValues.html , 2004

Jones, Capers, Applied Software Measurements, Assuring Productivity and Quality, New York, 1991

Jones, Capers, Assessment and Control of Software Risks, USA, 1994

Kaplan, R.S., Norton, D.P., Balanced Scorecard: Strategien erfolgreich umsetzen, Stuttgart, 1997

Kellner, Hedwig, Projekt-Mitarbeiter finden und führen, München, 2003

Kemerer, C.F., Reliability of Function Points Measurement – A Field Experiment, in: Communications of the ACM, Febr. 1993, p. 85-97

Kieback, A., et al., Prototyping in industriellen Software-Produkten, in: Informatik-Spektrum 15, 1992, p. 65-77

Kieser, A. (hrsg.), Woywode, M., Evolutionstheoretische Ansätze, in: Organisationstheorien, Stuttgart, 1999

Kralicek, Peter, Bilanzen lesen, Wien, 1999

Krcmar, H., Informationsmanagement, Heidelberg, 2000

Kroeber, A.L., Anthropology: Culture, Patterns, and Process, 1948

Kruchten, P., The Rational Unified Process, MA, 1998

Kuhn, Thomas S., The Structure of Scientific Revolutions, University of Chicago, 1970

Leonard-Barton, D., The Factory as a Learning Laboratory, in: SMR, 33.Jg., 1992

Levy, Haim, Introduction to Investments, Cincinnati, Ohio, 1999

Lewis, T.G., Steigerung des Unternehmenswerts, Landsberg, 1995

Litke, Hans-Dieter, DV-Projektmanagement, München, 1996

Litke, Hans-Dieter, Projektmanagement, München, 2004

Mandl, Gerwald, Rabel, Klaus, Unternehmensbewertung, Wien, 1997

McCabe, T.J., A Testing Methodology Using the McCabe Complexity Measure, in: IEEE Computer Society Press, 1983

McCabe, T.J., A Complexity Measure, Structured Testing, in: IEEE Computer Society Press, 1983, p. 3-15

Metha, N., MBASE EPG, USC-CSE, LA, CA, http://sunset.usc.edu/research/MBASE/EPG , 1999

Miller, Roy W., http://www-106.ibm.com/developerworks/java/library/j-xp0813/?loc=dwmain , 2004

Mintzberg, H., et al., Strategy Safari. A Guided Tour Through The Wilds Of Strategic Management, NY, 1998

Mintzberg, H., Die Mintzberg-Struktur: Organisationen effektiver gestalten, 1992

Neumann, I. von, Morgenstern, O., Spieltheorie und wirtschaftliches Verhalten, 1961

Noth, T., Kretzschmar, M., Aufwandsschätzung von DV-Projekten, Berlin, 1986

Nölting, A., Hebel-Wirkung, in: manager magazin, Nr.5/1998

Nutt, P.C., Planning Process Archetypes and their Effectiveness, in: Decision Science, 15.Jg., 1984

Oesterreich, Bernd, Objektorientierte Softwareentwicklung, München, 1998

Patzak, Gerold, Rattay, Günter, Projektmanagement, Wien, 1998

Paulk, M., Chrissis, M., Curtis, B., Capability Maturity Model for Software, USA, 1993.

Picot, A., Dietl, H.M., Franck, E., Organisation. Die ökonomische Perspektive, Stuttgart, 1997

Popper, Sir Karl R., Die offene Gesellschaft und ihre Feinde, Neuseeland, 1944

Pulford, Kevin, et al., A Quantitative Approach to Software Management, 1996

Rasiel, Ethan M., The McKinsey Way – Using the Techniques of the World's Top Strategic Consultants to Help You and Your Business, USA, 1999

Roos, Alexander, Stelter Daniek, Oetinger, v. Bolko, hrsg., Das Boston Consulting Strategie-Buch, Düsseldorf, 2000

Royce, W.W., Managing the development of large software systems, in: IEEE, 1970, p. 1-9

Rubey, R.J., Hartwick, R.D., Quantitative Measurement of Program Quality, 23. National Conference ACM; 1968

Scott-Morgan, Peter, Little, Arthur D., Unwritten Rules of the Game, McGraw-Hill, 1994

Severin, Roman, Vortrag bei Siemens in Wien, Mai, 2002

Sharpe, W.F., Capital Asset Prices: A Theory of Equilibrium under Conditions of Risk, in: Journal of Finance, Vol. 19, 1964

Siemens AG Österreich – Intranet, SEM Softwareentwicklungsmethoden, Copyright © Siemens AG Österreich 1997

Simon, H.A. (hrsg.), A Behavioural Model of Rational Choice, in: Models of Man, New York – London, 1957

Smith, Larry W., Project Clarity Through Stakeholder Analysis, in: CrossTalk, http://www.stsc.hill.af.mil , Dec. 2000

Sneed, H. M., Software-Management, Köln, 1987

Stadler, Wilfried (hrsg.), Beteiligungsfinanzierung, Wien, 1997

Staehle, W., Management, München, 1990

Steiner, Manfred, Bruns, Christoph, Wertpapiermanagement, Stuttgart, 2000

Stewart, B., Announcing the Stern Stewart Performance 1000: A New Way of Viewing Corporate America, Journal of Applied Corporate Finance, Vol. 3, 1990

Sunzi, Die Kunst des Krieges, zwischen 300 und 500 v.Chr., hrsg. von Clavell, James, München, 1988

Takahashi, Muneo, Kamayachi, Yuji, An Empirical Study of a Model for Program Error Prediction, in: IEEE Transactions on Software Engineering, Vol. 15/1, January 1989

Tavolato, Paul, Software-Projekt-Management, Wien, 1993

Thaller, Georg E., Software-Metriken einsetzen, bewerten und messen, Berlin, 2000

Thorne, Kip S., Black Holes & Time Warps. Einstein's Outrageous Legacy, New York, 1993

US Air Force, CrossTalk, http://www.stsc.hill.af.mil/crosstalk/ , 2004

Van Scoy, R., Problem or Opportunity, Technical Report CMU/SEI-92-TR-30, Pittsburgh, PA, USA, 1992

Versteegen, Gerhard (hrsg.), Risikomanagement in IT-Projekten, Berlin, 2003

Welge, M.K., Al-Laham, A., Kajüter, P., (hrsg.), Praxis des strategischen Managements, Wiesbaden, 1999

Witte, E., Hauschildt, J., Grün, O., (hrsg.), Innovative Entscheidungsprozesse – Das Ergebnis des Projekts Columbus, Tübingen, 1988

Wojda, Franz, Projektorganisation-Projektmanagement, Wien, 2001

Wojda, Franz, Lernende Organisation durch Projekt-Programm-Management, in: Office Management 1-2, 1996

Wojda, Franz, Organisation und Führung, TU-Wien, 1998

Wojda, Franz, Buresch, M., Seghezzi (hrsg.), Ganzheitliche Unternehmensführung, Stuttgart, 1997

Wojda, Franz (hrsg.), Waldner, B., Innovative Organisationsformen, Stuttgart, 2000

Zillmer, Hans-Joachim, Darwins Irrtum, München, 2000

Internetadressen ohne Hervorhebung eines Autors:

http://c2.com/ppr , Portland Pattern Repository, 2004

http://dev.panopticsearch.com/xp-notes.html , 2004

http://www.butlergroup.com , IBM's Übernahme der Firma Rational Software Corp., 2004

http://www.extremeprogramming.org/ , 2004

http://www.extremeprogramming.org/map/project.html , 2004

http://www.extremeprogramming.org/rules/testfirst.html , 2004

http://www.hillside.net , The Hillside Group is a nonprofit corporation dedicated to improving human communication about computers by encouraging people to codify common programming and design practice, 2004

http://www.iso.ch/iso/en/stdsdevelopment/tc/tclist/TechnicalCommitteeStandardsListPage.TechnicalCommitteeStandardsList?COMMID=40 , Standards and/or guides of JTC 1/SC 7, 2004

http://www.itq.ch/ , Qualität und Informatik, Capability Dimension von SPICE/ISO 15504, Zürich, 2004

http://www.questhouse.com/excerpt1.htm , Engineering a Better Software Organization, 2003

http://www.sei.cmu.edu/sema/welcome.html , Software Engineering Measurement & Analysis (SEMA), 2004

http://www.sei.cmu.edu/cmmi/general/ , CMMI General Information, 2004

http://www.sei.cmu.edu/cmmi/models/ , CMMI-Models, CMMI-SE/SW/IPPD/SS, V1.1, 2004

http://www.sqi.gu.edu.au/spice/ , SPICE presented by SQI, 2004

http://www.stsc.hill.af.mil/crosstalk/ , CrossTalk: The Journal of Defense Software Engineering, 2004

http://www.tantara.ab.ca/info.htm , Informational Hotlist (for software process improvement, software quality assurance,...), 2004

http://www-306.ibm.com/software/rational/info/literature/lifecycle.jsp , IBM's Rational Unified Process, UML, 2004

15. Abkürzungsverzeichnis

AMI	Application of Metrics in Industry
AP	Arbeitspaket
APV	Adjusted Present Value
BSC	Balanced Scorecard
CAPM	Capital Asset Pricing Model
CASE	Computer Aided Software Engineering
CC	Carbon Copy
CFROI	Cash Flow Return on Investment
CIP	Chancen-Identifizierungs-Prozess
CM	Konfigurationsmanagement, Configuration Management
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
COTS	Commercial off the shelf
CR	Change Request
CRC	Class-Responsibilities-Collaboration
D	Discover
DCF	Discounted Cash Flow
DoD	US-Department of Defense
DV	Datenverarbeitung
E	Envision
EBIT	Earnings before Interest and Taxes
EGT	Ergebnis der gewöhnlichen Geschäftstätigkeit
EVA	Economic Value Added
FCF	Free Cash Flow
FCM	Factor Criteria Metrics
FP	Function Point
GQM	Goal Question Metric
HTML	HyperText Markup Language
HW	Hardware
I	Improve
IDEF	Integrated Computer Aided Manufacturing Definition
iwiS	im wesentlichen in Session
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IFPUG	International Function Point Users' Group
IOC	Initial Operational Capability
IPO	Initial Public Offer
IPPD	Integrated Product and Process Development

ISO	International Organization for Standardization
KM	Konfigurationsmanagement
LCA	Life Cycle Architecture
LCO	Life Cycle Objectives
LM	Leistungsmerkmal
LOC	Lines of Code
LSB	Leistungsstand-Beurteilung
M	Measure
MBASE	Model-Based Architecting and Software Engineering
MM	Mann-Monat(e)
MR	Modification Request
MTA	Meilenstein-Trend-Analyse
MTBF	Mean Time Between Failures
MTTF	Mean Time To Failure
MVP-L	Multi View Process Modeling Language
NASDAQ	National Association of Securities Dealers' Automated Quotations system
NOP(L)AT	Net Operating Profit Less Adjusted Taxes
OEM	Original Equipment Manufacturer
OEMP	Original Equipment Manufacturer-Prozess
OO	Objekt-Orientierung
OOA	Objektorientierte Analyse
OOD	Objektorientiertes Design
OOP	Objektorientierte Programmierung
OOPSLA	Object-Oriented Programming, Systems, Languages, and Applications
P	Plan
PA	Principal-Agent
P-CMM	People Capability Maturity Model
PBP	Produkt-Bereitstellungs-Prozess
PLoP	Pattern Languages of Programming
PM	Projekt-Management
PMP	Produkt-Management-Prozess
PPM	Projekt-Programm-Management
QFD	Quality Function Deployment
QIP	Quality Improvement Paradigm
QS	Qualitätssicherung
RE	Risk Exposure
RL	Risk Leverage
RM	Risiko-Management
SCP	Structure Conduct Performance
SE	Systemerstellung, System Engineering
SEI	Software Engineering Institute / Carnegie Mellon University Pittsburgh, USA
SEP	Software-Entwicklungs-Prozess

SGE	Strategische Geschäfts-Einheit
SGF	Strategisches Geschäfts-Feld
SP	Software-Projekt
SPs	Software-Projekte
SPI	Software Process Improvement
SPICE	Software Process Improvement and Capability dEtermination
SPM	Software-Projekt-Management
SPR	Software Productivity Research
SQM	Software Quality Metrics
SW	Software
SWOT	Strength-Weakness-Opportunity-Threat
SW-CMM	Capability Maturity Model for Software
TQM	Total Quality Management
UAN	Unterauftragnehmer
UML	Unified Modeling Language
W	Work
WACC	Weighted Average Cost of Capital
WWW	World Wide Web
XP	Extreme Programming

Lebenslauf

Christian Exl
Geb. am 13.02.1969 in Wien
verheiratet; Sohn Philip

Bildungsweg:

1975 – 1976	Volksschule in Wien 7
1979 – 1983	Hauptschule in Wien 7
1983 – 1988	HTL Schellinggasse in Wien 1, Nachrichtentechnik und Elektronik
1988 – 1995	Studium der Elektrotechnik mit Spezialisierung: Informatik und Betriebswirtschaftslehre an der TU-Wien; ein Semester Politikwissenschaft an der UNI-Wien; Diplomarbeit: „Gesundheitswesen - Technologiefolgen & Kosten“
2000 - 2004	Doktoratsstudium am Institut für Betriebswissenschaften, Arbeitswissenschaft und Betriebswirtschaftslehre an der TU-Wien zum Thema: „Software Projekt Produkt Programm Management & Patterns“

Berufsweg:

1983 – 1995	Ferialpraktika, u.a. bei Schrack, ORF, Einzelhandel
1991 – 1992	Siemens AG Deutschland in der Funktion: Softwareentwickler im Bereich Automationstechnik in München
1996	Präsenzdienst
1997	Assistent des Direktors des European Universities Continuing Education Network in Liège & Bruxelles
seit 1997	Siemens AG Österreich in den Funktionen: Projektmanager, Produkt und Business Line Manager in Wien und München