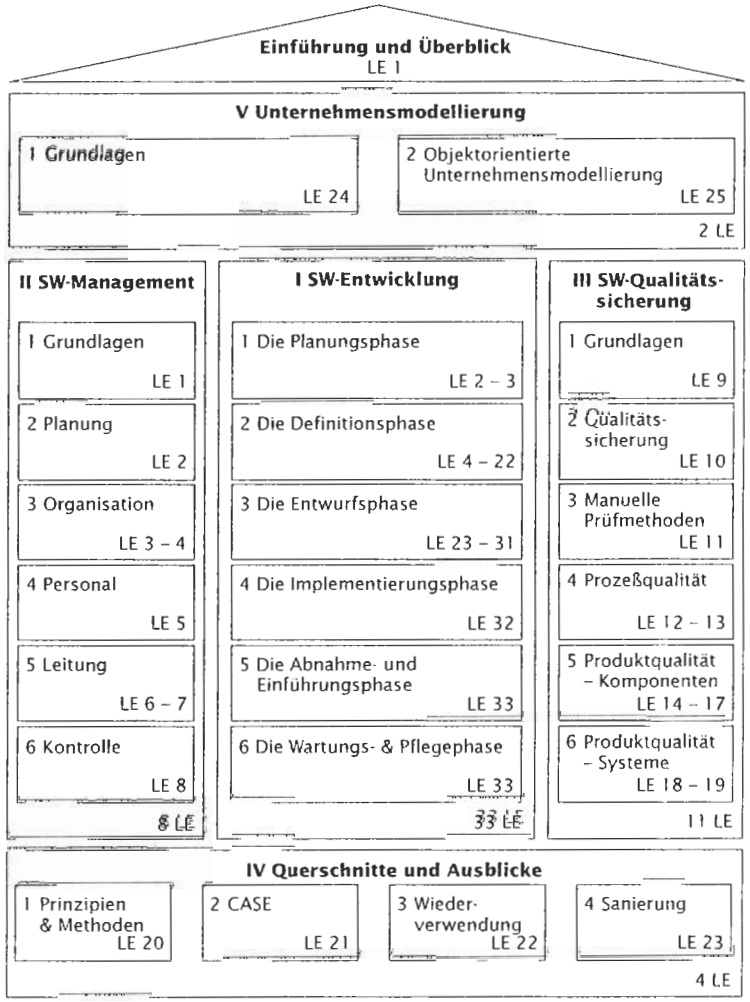
Inhalt

II Software-Management



Legende: LE = Lehreinheit (für jeweils 1 Unterrichtsdoppelstunde)

LE 1

1 Grundlagen

wissen

- Mögliche Software-Geschäftsstrategien und ihre Charakteristiken aufzählen können.
- Die fünf Managementfunktionen und die Hauptaktivitäten des Managements nennen können.
- Angeben können, welche Einflußfaktoren wie stark die Produktivität beeinflussen.
- Maßnahmen zur Produktivitätssteigerung aufzählen können.
- Die Unterschiede zwischen einer Software-Entwicklung und Produktentwicklungen in anderen Ingenieurbereichen erklären können.
- Definitionen für die Software-Produktivität und ihre Problematik aufzeigen können.
- Den Zusammenhang zwischen Produktivität und Qualität darstellen können.
- Für gegebene Szenarien notwendige Managementaktivitäten identifizieren können.
- Für gegebene Szenarien Produktivitätsverbesserungsmaßnahmen, geordnet nach Prioritäten, vorschlagen und begründen können.

Tom DeMar
* 1940 in Ha
ton, Pennsyl
USA, Erfinde
Strukturierte
Analyse (SA),
Buch: *Structu*
Analysis and
tem Specifica
1978, bedeut
Beiträge zum
Software-Ma
ment, Bücher
Peopleware 1
(Coautor:
T. Lister), *Co*
trolling Softw
Projects 1982
BS *Electrical*
neering von
Cornell Unive
MS in Elektro
nik von der
lumbia Unive
heute: *Princi*
of The Atlanti
Systems Guil
New York; 19
Warnier Prize
Excellence in
mation Scien

1.1 Einführung

Eine erfolgreiche Software-Erstellung hängt wesentlich von der Güte des Software-Managements ab. Wesentliche Ziele des Software-Managements bestehen darin, die Produktivität zu erhöhen, eine definierte Qualität sicherzustellen und die Kosten zu senken.

Ziele: Produktivität, ualität, Kosten

In /Grady 92, S. 22/ werden drei primäre Managementstrategien für Software-Unternehmen unterschieden:

1 Maximierung der Kundenzufriedenheit,
2 Minimierung des Aufwands und der Zeit der Software-Erstellung,
3 Minimierung von Fehlern.

3 Strategien

Die Hauptcharakteristika dieser Strategien zeigt Tab. 1.1-1.

ptcharakteristika	Max. Kundenzufriedenheit	Min. Aufwand & Zeit	Min. Fehler
ptgeschäfts-tegie	Marktanteile erlangen	Konkurrenz erfordert neue Produkte oder Kostenkontrolle	Halten oder Vergrößern des Marktanteils
zient ...	... beim ersten Markteinstieg	... bei mehreren Konkurrenzprodukten, oder wenn man profitablere Produkte verkauft	... bei konkurrenzfähigen Eigenschaften und falls ein adäquater Marktanteil gehalten wird
rakteristische enschaften	Kommunikation mit dem Kunden, schnelle Reaktion	Fokus auf Auslieferungsdatum und Aufwand	Analyse und Entfernen von Fehlerursachen

Tab. 1.1-1: Charakteristika von Software-Geschäftsstrategien (in Anlehnung an rady 92, S. 24/)

Besonderheiten

Software-Management unterscheidet sich vom Management anderer Ingenieurbereiche aus folgenden Gründen:

- Das Produkt ist immateriell.
  Man sieht es nicht. Man kann es nicht berühren. Der Manager ist abhängig von der Dokumentation, um den Entwicklungsfortschritt zu überprüfen.
- Der Entwicklungsfortschritt ist objektiv nicht zu ermitteln.
  Es ist schwer festzustellen, ob ein Teilprodukt fertiggestellt ist oder nicht. Ein Software-Entwickler kann fast jederzeit behaupten, er sei mit einem Teilprodukt fertig, d.h. es existiert ein Dokument oder ein Quellprogramm. Um festzustellen, ob dieses Dokument oder Quellprogramm überhaupt verwendbar ist, ist eine genaue Untersuchung durch einen Experten erforderlich. Um die Qualität zu überprüfen, muß man fast denselben Aufwand betreiben wie bei der Entwicklung. Der Software-Manager ist also voll auf die Aussagen seiner Mitarbeiter angewiesen. Es fehlen in der Regel einfache und billige Kontrollmittel. Sogar gute Entwickler glauben oft, fertig zu sein und sind es nicht, da die Aufgabe nicht eindeutig definiert wurde.

  Natürlich gibt es auch genug Entwickler, die den wahren Stand ihrer Arbeit verdecken, um Zeit zu gewinnen. Bei Großprojekten wird die Verschleierung des Projektzustandes oft von den Managern der mittleren Ebene geschickt fortgesetzt, so daß es noch schwieriger wird, den wahren Zustand zu ermitteln /Sneed 87, S. 203/.
- Eine Software-Entwicklung verläuft nicht-deterministisch.
  Neue Erkenntnisse während der Entwicklung haben Auswirkungen auf die bisherigen Ergebnisse. Deshalb ist ein bestimmtes Teilprodukt immer nur bedingt fertig. Diese entwicklungsinternen Zyklen müssen bei der Projektplanung mitberücksichtigt werden. Oft wird die Hälfte der Arbeit zur Überarbeitung bereits erstellter Teilprodukte benötigt /Sneed 87, S. 203f./.
- Es gibt noch kein klares Verständnis vom Entwicklungsprozeß.
  Andere Ingenieurdisziplinen haben eine lange Historie. Die Entwicklungsstufen dort sind verstanden und vorhersagbar.
- Große Software-Systeme tendieren dazu, einmalige Entwicklungen zu sein.
  Sie unterscheiden sich von früheren Entwicklungen. Gemachte Erfahrungen sind von begrenztem Wert, um vorherzusagen, wie das Management dieser Entwicklungsprozesse aussehen sollte.
- Unteilbarkeit der Arbeit.
  Um ein Software-Problem zu lösen, muß sich der Entwickler intensiv mit dem Problem beschäftigen. Dadurch wird die Übertragung von Aufgaben an einen anderen Mitarbeiter teuer, da der neue Mitarbeiter die Einarbeitungszeit des alten Mitarbeiters wiederholen muß.
  Aufgaben sind also nicht voll austauschbar. Nicht jeder Mitarbeiter kann die Aufgaben eines anderen übernehmen, denn dafür sind die Aufgaben oft viel zu spezialisiert. Fällt ein Experte aus, dann kann man ihn nur durch einen vergleichbaren Experten ersetzen. In Netzplänen wird jedoch oft von der vollen Austauschbarkeit der Mitarbeiter ausgegangen /Sneed 87, 204f./.

Kapitel 2.5

- Die Software-Technik ist keine Naturwissenschaft.
  Als ein künstliches Produkt des menschlichen Erfindungsgeistes basiert Software nicht auf physikalischen Prinzipien und wird daher auch nicht durch diese begrenzt.
- Hoher Grad an Abstraktion, bei gleichzeitig niedrigem Grad an Normierung /Sneed 87, S. 205/.

Diese Unterschiede gegenüber Entwicklungen anderer Disziplinen sind für einen Außenstehenden nur schwer zu erkennen. Daher scheitern viele Manager aus anderen Branchen, wenn sie das Management von Software-Entwicklungen versuchen.

Viele Software-Manager sind von Managementideen geprägt, die aus typischen Produktionsprozessen stammen. Wegen der Besonderheiten einer Software-Entwicklung ist mehr Management als bei

Produktionsprozessen erforderlich, nicht weniger. Viele Software-Entwicklungen werden zu spät fertiggestellt, die Kosten und die Termine werden überschritten. Jede Entwicklung eines großen Software-Systems ist ein neues und technisch innovatives Projekt. Viele Ingenieurprojekte, die ebenfalls innovativ sind, z.B. neue Transportsysteme, neue Brücken, neue Tunnel, haben ebenfalls Zeit- und Kostenprobleme.

1.2 Aufgaben

Die drei grundlegenden Elemente, mit denen sich ein Manager befaßt, sind Ideen, Dinge und Menschen. Das Management dieser drei Elemente steht in direktem Zusammenhang mit konzeptionellem Denken (Planung ist ein wesentlicher Teil davon), Verwalten und Führen von Mitarbeitern. Eine Organisation benötigt dementsprechend den Planer, den Verwalter und den Leiter. Nicht jeder Manager ist gleichzeitig ein guter Führer von Mitarbeitern und umgekehrt.

Literaturhinweis: Diese Ausführungen orientieren sich an /Mackenzie 69/.

Management läßt sich folgendermaßen definieren:

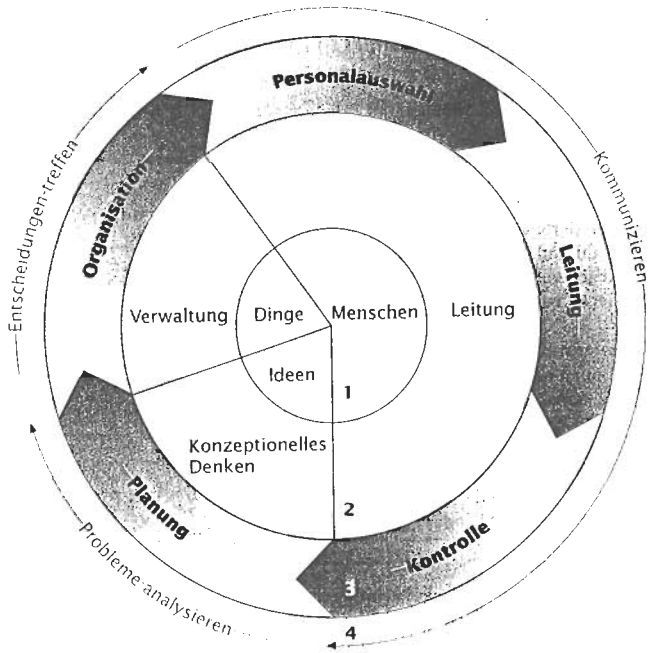
Management umfaßt alle Aktivitäten und Aufgaben, die von einem oder mehreren Managern durchgeführt werden, um die Aktivitäten von Mitarbeitern zu planen und zu kontrollieren damit ein Ziel oder der Abschluß einer Aktivität erreicht wird, die durch die Mitarbeiter alleine nicht erreicht werden können.

In Kurzform:

Management sorgt dafür, daß Ziele durch Mitarbeiter erreicht werden.

Der Managementprozeß ist in Abb. 1.2-1 dargestellt.

Da die fünf universellen Managementfunktionen – Planung, Organisation, Personalauswahl, Leitung, Kontrolle – in der Regel sequentiell durchgeführt werden, sind sie aufeinanderfolgend und als Zyklus angeordnet. Zuerst wird das Ziel oder der Zweck eines Vorhabens festgelegt. Dazu ist Planung erforderlich. Anschließend ist der Weg festzulegen, wie die Arbeit in verwaltbare Einheiten aufgeteilt werden kann: eine Aufgabe der Organisation. Danach ist das qualifizierte Personal zur Erledigung der Aufgaben auszuwählen. Leitung ist anschließend erforderlich, um die Mitarbeiter zur Arbeit zu motivieren, damit die gewünschten Ziele erreicht werden. Kontrolle dient dazu, die Ergebnisse mit der Planung zu vergleichen und die Erwartungen an die Mitarbeiter mit ihren Leistungen in Beziehung zu setzen. Außerdem ist eine revidierte Planung der Arbeit vorzunehmen, um Korrekturen durchzuführen. Der Zyklus startet wieder von vorne.



Legende: 1 Elemente
2 Aufgaben
3 Sequentielle Funktionen
4 Kontinuierliche Funktionen

Abb. 1.2-1: Der Managem Prozeß (abgeleitet vo /Mackenzie 6

Drei Funktionen – Analysieren von Problemen, Treffen von Entscheidungen und Kommunizieren – sind kontinuierliche Funktionen, da sie während des gesamten Managementprozesses auftauchen und nicht nur in einer speziellen Sequenz. Entscheidungen müssen beispielsweise während der Planung, der Organisation, der Leitung und der Kontrolle getroffen werden. Eine Problemanalyse ist während aller sequentiellen Funktionen erforderlich.

In der Praxis mischen sich die verschiedenen Funktionen und Aktivitäten. Die fundamentalen Managementaktivitäten sind in Tab. 1.2-1 zusammengestellt.

Tab. 1.2-1: Die Hauptktivitäten des Managements Anlehnung an /Thayer 90, S. 17/)

Planungsaktivitäten
- Ziele setzen
- Strategien und Taktiken entwickeln
- Termine festlegen
- Entscheidungen treffen
- Vorgehensweisen auswählen und Regeln festlegen
- Zukünftige Situationen vorhersehen
- Budgets vorbereiten

Organisationsaktivitäten
- Identifizieren und Gruppieren der zu erledigenden Aufgaben
- Auswahl und Etablierung organisatorischer Strukturen
- Festlegen von Verantwortungsbereichen und disziplinarischen Vollmachten
- Festlegen von Qualifikationsprofilen für Positionen

Personalaktivitäten
- Positionen besetzen
- Neues Personal einstellen und integrieren
- Aus- und Weiterbildung von Mitarbeitern
- Personalentwicklung planen
- Mitarbeiter beurteilen
- Mitarbeiter bezahlen
- Mitarbeiter versetzen oder entlassen

Leitungsaktivitäten
- Mitarbeiter führen und beaufsichtigen
- Kompetenzen delegieren
- Mitarbeiter motivieren
- Aktivitäten koordinieren
- Kommunikation unterstützen
- Konflikte lösen
- Innovationen einführen

Kontrollaktivitäten
- Prozeß- und Produktstandards entwickeln und festlegen
- Berichts- und Kontrollwesen etablieren
- Prozesse und Produkte vermessen
- Korrekturaktivitäten initiieren
- Loben und Tadeln

1.3 Produktivität

Ein zentrales Ziel des Software-Managements besteht darin, permanent die Produktivität der Software-Erstellung zu erhöhen. Produktivitätssteigerung kann verschiedenes bedeuten /Wallmüller 90, S. 57f./:

- Software-Produkte schneller, d.h. in kürzerer Kalenderzeit entwikkeln. Diese Art der Produktivitätssteigerung wird oft als Effizienzsteigerung bezeichnet.

- Software-Produkte so zu entwickeln, daß sie einen höheren *Return on Investment* liefern. Es soll weniger Geld ausgegeben werden, um Produkte mit gleichen Anforderungen zu entwickeln und zu pflegen.
- Software-Produkte mit besserer Qualität entwickeln.

Return on Investment = Verzinsung des eingesetzten Kapitals

Um Produktivitätssteigerungen feststellen zu können, muß die Produktivität gemessen werden. Dazu sind Produktivitätsmaße erforderlich.

Produktivität

Global läßt sich **Produktivität** folgendermaßen definieren.

$$\text{Produktivität} = \frac{\text{Leistung}}{\text{Aufwand}} \qquad 1$$

Es gibt unterschiedliche Ansätze, um Leistung und Aufwand zu definieren.

/Boehm 87, S. 44/ definiert Leistung durch die produzierten Ergebnisse im Entwicklungsprozeß:

$$\text{Produktivität} = \frac{\text{Produzierte Ergebnisse}}{\text{Eingesetzter Aufwand}} \qquad 2$$

/Sneed 87, S. 31/ definiert Leistung durch die Anzahl der Software-Elemente und Aufwand durch geleistete Mitarbeitertage:

$$\text{Produktivität} = \frac{\text{Anzahl Software-Elemente}}{\text{Geleistete Mitarbeitertage}} \qquad 3$$

Nach diesen Definitionen kann die Produktivität verbessert werden, wenn
- die Ergebnisse vermehrt,
- der Aufwand verringert oder
- die Ergebnisse vermehrt und der Aufwand verringert werden.

Die Definitionen 2 und 3 sind problematisch. Ein Problem besteht darin, geeignete Ergebnisse bzw. Elemente zu definieren, die quantifizierbar sind.

Software ist ein vielfältiges Produkt /Sneed 87, S. 31/. Es setzt sich zusammen aus Teilprodukten: Analysemodell, Architekturmodell, Programme. Zusätzlich gehören dazu: Benutzerhandbuch, Hilfesystem, Testdaten, *Review*-Ergebnisse usw. Daher stellt es eine grobe Vereinfachung dar, ein Software-Produkt auf ein Teilprodukt oder ein Element zu reduzieren. Dennoch geschieht dies heute in aller Regel. Das konventionelle Maß verwendet die Anzahl der Quellprogramm-Zeilen – *Lines of Code* (LOC). Ein Problem liegt darin, daß Quellprogrammzeilen nicht gleichwertig sind – weder innerhalb einer Programmiersprache noch zwischen Programmiersprachen. Empirische Untersuchungen haben ergeben, daß die Produktivität – gemessen in LOC/Aufwand – in Abhängigkeit vom Produkttyp, der Programmgröße und dem Prozentsatz der Programmänderungen signifikant variiert.

Trotz dieser Probleme verwendet beispielsweise die Firma HP das Maß NCSS *(non commented source statements)* als Teil ihres Produktivitätsmaßes mit folgender Definition /Grady, Caswell 87, S. 58/:

Kapitel 6.2

NCSS Anzahl Quellanweisungen einschließlich Compiler-Anweisungen, Datendeklarationen und ausführbaren Anweisungen, aber ohne Leerzeilen oder Zeilen, die vollständig aus Kommentaren bestehen.

Abb. 1.3-1 zeigt eine frühe Statistik von HP. Die Anzahl »Zeilen nichtkommentierter Quellanweisungen« (NCSS) werden durch die gesamte Projektzeit, gemessen in Ingenieurmonaten, geteilt. Jede Säule stellt die durchschnittliche Produktivität der Projekte, geschrieben in der angegebenen Sprache, dar. Die Säule »Wiederverwendet« bedeutet, daß mindestens 75 Prozent des Codes bereits existierte. »Verschiedene Sprachen« gibt an, daß keine Sprache mehr als 75 Prozent Codeanteil ausmacht. Es wird darauf hingewiesen, daß die gesammelten Daten nicht ausreichen, um statistisch signifikante Aussagen zu treffen. Auffällig ist der Produktivitätsunterschied zwischen Projekten, bei denen ein großer Anteil Software wiederverwendet wird gegenüber denjenigen, bei denen alles neu geschrieben wird.

Kapitel I 1.5 Ein anderes Maß für »Anzahl Software-Elemente« sind *Function Points*, die im Rahmen der *Function Point*-Methode für Aufwandsschätzungen ermittelt werden.

Im Gegensatz zur Definition und Messung von produzierten Ergebnissen bzw. Software-Elementen erscheint das Maß »Eingesetzter Aufwand« weniger problematisch.

Der Aufwand für die Software-Entwicklung setzt sich zusammen aus:

- Personalkosten,
- Kosten für Computerressourcen,
- Kosten für Hilfsmittel.

Den überragenden Kostenblock bilden dabei die Personalkosten, so daß es gerechtfertigt erscheint, wie in Definition 3, den Aufwand auf

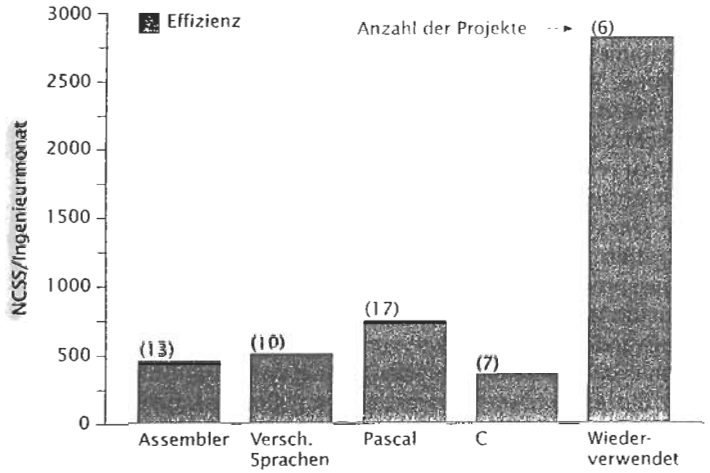


Abb. 1.3-1: Durchschnittliche NCSS/Ingenieurmonate pro Sprache /Grady, Caswell 87, S. 22/

die geleisteten Mitarbeitertage zu fokussieren. Aber auch diese Größe ist problematisch.

In /DeMarco, Lister 91, S. 73ff./ wird gezeigt, daß es sinnvoll ist, zwischen geistiger und körperlicher Anwesenheit zu unterscheiden. Was in Zeiterfassungsbögen eingetragen wird, ist die körperliche Anwesenheit. Für die Produktivität ausschlaggebend ist jedoch die geistige Anwesenheit. Als Metrik schlagen DeMarco und Lister /a.a.O. S. 77/ einen Umweltfaktor vor:

$$\text{Umweltfaktor} = \frac{\text{ungestörte Stunden}}{\text{Stunden körperlicher Anwesenheit}} \qquad 4$$

Erreicht der Umweltfaktor einen hohen Wert, dann erlaubt die Arbeitsumgebung den Mitarbeitern eine hohe Produktivität. 40 Prozent sind ein erreichbarer Wert. Aber selbst innerhalb einer Firma wurden Unterschiede zwischen 10 Prozent und 38 Prozent ermittelt.

Bei der Firma HP /Grady, Caswell 87, S. 251/ wird der Aufwand durch benötigte Ingenieurmonate quantifiziert, die folgendermaßen definiert sind: 40 – 50 Stunden pro Woche ohne Urlaub oder Krankheit.

Basili (zitiert nach /Grady, Caswell 87, S. 3/) ersetzt »produzierte Ergebnisse« durch »Produktwert«. Er betrachtet die Software-Produktion als einen Prozeß der Wertschöpfung. Das entstehende Produkt stellt einen Wert dar:

$$\text{Produktivität} = \frac{\text{Produktwert}}{\text{Kosten}} \qquad 5$$

Die Einflußfaktoren, die den Produktwert und die Kosten nach Basili bestimmen, zeigt Abb. 1.3-2. Entscheidend für den Produktwert ist der mögliche Nutzen für die Benutzer bzw. das Unternehmen. Als

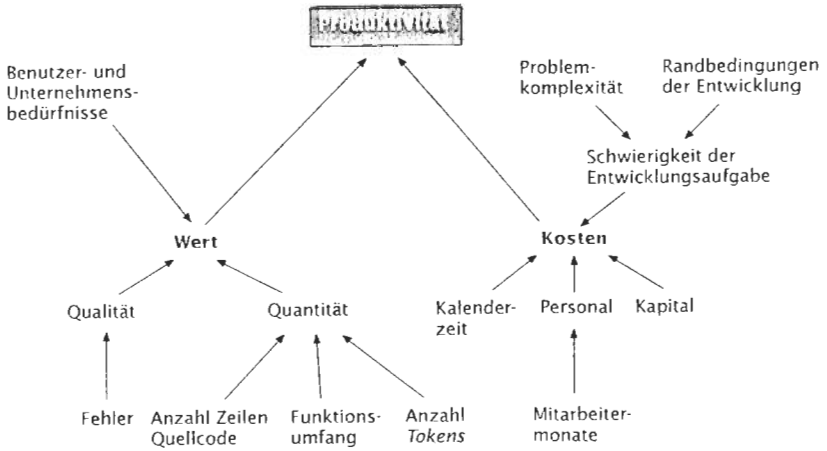


Abb. 1.3-2: Produktivität u ihre Einflußfakt ren nach Basili

Problem erweist sich aber auch hier, wie man den Produktwert ermittelt bzw. frühzeitig schätzen kann.

Als beste Produktivitätsmetrik hat sich nach einer neuen europäischen Untersuchung /Maxwell, Wassenhove, Dutta 96/ folgende bewährt:

$$\text{Produktivität} = \frac{\text{Anzahl Zeilen Quellcode [LOC]}}{\text{Aufwand in Mitarbeitermonaten}} \qquad 6$$

Trotz aller hier aufgezeigten Schwächen ist jeder Quantifizierungsversuch der Software-Produktivität besser als gar keiner. Nur wenn die Produktivität quantifizierbar und meßbar ist, ist es auch möglich, das Erreichen eines Produktivitätsziels festzustellen.

1.4 Einflußfaktoren der Produktivität

Damit ein Software-Manager die Produktivität verbessern kann, muß er wissen, welche Faktoren die Produktivität am meisten beeinflussen. Da es bisher noch kein umfassendes Modell dieser Einflußfaktoren gibt, werden im folgenden bekannte Untersuchungen zusammengestellt. Die Einflußfaktoren werden in folgende Gruppen eingeteilt (Abb. 1.4-1):

- Produkteinflüsse,
- Prozeßeinflüsse,
- Mitarbeitereinflüsse,
- Managementeinflüsse.

Produkteinflüsse

Komplexität

Die **Produktkomplexität** beeinflußt wesentlich die Produktivität. /Boehm 87, S. 47/ gibt ein Verhältnis von 1:2,36 an. Ein komplexes Produkt benötigt also den 2,36-fachen Aufwand eines einfachen Produkts. Das Messen der Produktkomplexität ist schwierig. In /Boehm 81, S. 390f/ werden sechs Komplexitätskategorien eingeführt, um die Komplexität von Programmcode mit Hilfe einer Tabelle zu beurteilen.

Kapitel 6.2 Metriken

Weitere Ansätze zur Vermessung von Code sind die Metriken von McCabe und Halstead. Es gibt zur Zeit aber keine befriedigende Metrik, die es erlaubt, die Komplexität eines Software-Produkts in eine Zahl zu fassen.

Die Produktkomplexität ist vom Management nur bedingt zu beeinflussen. Wesentlich erscheint, daß für die Entwicklung die Methoden eingesetzt werden, die es erlauben, die jeweilige Komplexitätsausprägung geeignet zu modellieren.

Kapitel I 2.14

Beispielsweise können Aktionen, die von komplexen Bedingungen abhängen, am besten durch Entscheidungstabellen beschrieben wer-

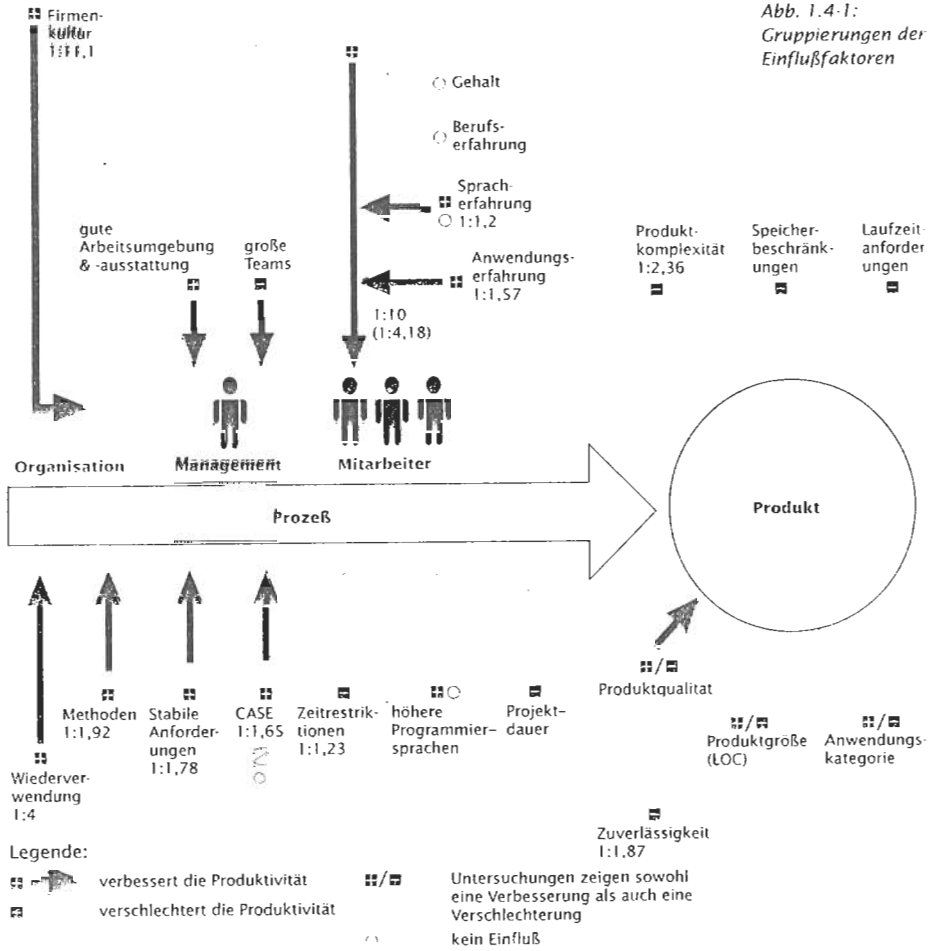


Abb. 1.4-1: Gruppierungen der Einflußfaktoren

den. Wird dagegen Pseudo-Code zur Beschreibung verwendet, dann wird die Beschreibung unübersichtlich.

Größe

Einen großen Einfluß auf die Entwicklungskosten hat die **Produktgröße,** gemessen in LOC /Boehm 87, S. 47/. In der Regel geht man davon aus, daß der Aufwand überproportional mit der Produktgröße zunimmt. /Maxwell, Wassenhove, Dutta 96/ haben jedoch festgestellt, daß die Produktivität mit zunehmendem Produktumfang steigt.

Qualität

Die geforderte **Produktqualität** hat ebenfalls einen Einfluß auf die Produktivität. Dieser Einfluß kann positiv oder negativ sein. Dieses Thema wird im nächsten Kapitel ausführlich behandelt. /Boehm 87, S. 47/ gibt ein Verhältnis von 1:1,87 in Bezug auf geforderte Zuverlässigkeit an. Ein zuverlässiges Programm benötigt den 1,87-fa-

chen Aufwand eines nicht so zuverlässigen Programms. /Maxwell, Wassenhove, Dutta 96/ bestätigen eine sinkende Produktivität bei höheren Zuverlässigkeitsanforderungen.

Laufzeit
Speicherbedarf

Zunehmende **Laufzeitanforderungen** und zunehmende **Speicherbeschränkungen** führen nach /Maxwell, Wassenhove, Dutta 96/ zu sinkender Produktivität.

Anwendungskategorie

Zum Einfluß der **Anwendungskategorie** auf die Produktivität liegen unterschiedliche Erkenntnisse vor.

Prozeßeinflüsse

Eine Software-Entwicklung wird wesentlich durch die verwendeten **Methoden** und die sie unterstützenden Werkzeuge beeinflußt.

CASE

Zur Produktivitätssteigerung durch den Einsatz von CASE-Umgebungen und **CASE-Werkzeugen** gibt es in der Literatur sehr unterschiedliche Aussagen. Leider werden oft nur pauschale Angaben publiziert, ohne das Umfeld genauer zu spezifizieren. In /Boehm 87, S. 47/ wird ein Produktivitätsverhältnis von 1:1,65 zwischen einem minimalen und einem maximalen CASE-Einsatz angegeben. Durch optimalen CASE-Einsatz kann also eine Produktivitätssteigerung um den Faktor 1,65 erreicht werden. 1981 lag der maximale Steigerungsfaktor noch bei 1,49 /Boehm 81, S. 642/.

Eine differenzierte Betrachtung erfordert zunächst die Berücksichtigung der Zeit. In der Einführungsphase von CASE und im ersten Projekt sinkt in der Regel zunächst die Produktivität. Außerdem erscheint es sinnvoll, zwischen der Entwicklungs- und der Wartungsproduktivität zu unterscheiden. Abb. 1.4-2 zeigt diese Differenzierung und darüber hinaus den Einfluß auf die Qualität.

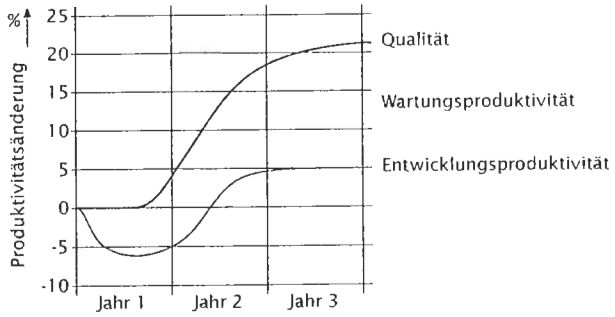


Abb. 1.4-2: Produktivitäts- und Qualitätsverlauf durch den Einsatz von CASE /Jones 92, S. 40/

Eine weitere Differenzierung muß den Umfang von CASE berücksichtigen. Es ist ein Unterschied, ob ein einzelnes CASE-Werkzeug oder eine umfangreiche CASE-Umgebung bestehend aus einer CASE-Plattform mit vielen Werkzeugen, eingeführt und eingesetzt wird.

Hauptkapitel IV 2

Abb. 1.4-3 zeigt das Ergebnis einer Produktivitätsschätzung unter Berücksichtigung des CASE-Umfangs.

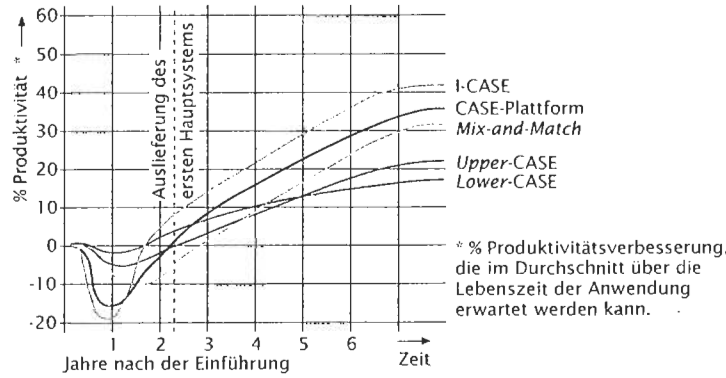


Abb. 1.4-3: Produktivitätsverlauf in Abhängigkeit vom CASE-Umfang /Gartner Group 90, S. 14/

Legende:

I-CASE:	*Integrated* CASE (Werkzeugketten, die die gesamte Entwicklung unterstützen)
CASE-Plattform:	Rahmensystem, das Basisdienstleistungen abdeckt und das Einbetten von Werkzeugen erlaubt
Mix-and-Match:	Einsatz verschiedener Werkzeuge in den einzelnen Phasen
Upper-CASE:	Werkzeuge für Planung, Definition und Entwurf
Lower-CASE:	Werkzeuge für Implementierung, Wartung und Pflege

Global kann festgestellt werden, daß komplexe CASE-Umgebungen zunächst zu einem größeren Produktivitätsverlust führen, nach der Einführungsphase aber größere Produktivitätssteigerungen ermöglichen. Das Phänomen der sinkenden Produktivität in der Einführungsphase von CASE-Produkten hängt wahrscheinlich damit zusammen, daß bei allen Produktivitätsangaben CASE- und Methodeneinführung zusammen betrachtet werden.

Der kritische Erfolgsfaktor ist nicht die CASE-Einführung, sondern die Methodeneinführung. Die Anwendung neuer Methoden erfordert von jedem Mitarbeiter eine Änderung seiner Denkgewohnheiten. Selbst nach einem guten Methodentraining wird der Mitarbeiter bei der Anwendung der neuen Methode auf seine erste »echte« Entwicklung Schwierigkeiten haben. Dies führt zu den angeführten Produktivitätseinbußen. Kritisch ist anzumerken, daß heute für das Methodentraining zu wenig Zeit und für die Werkzeugbedienung zuviel Zeit investiert wird.

Methoden

Für den Einsatz moderner **Software-Methoden** gibt /Boehm 87, S. 47/ ein Verhältnis von 1:1,92 an, d.h. durch moderne Methoden erzielt man einen Steigerungsfaktor von 1,92.

/Maxwell, Wassenhove, Dutta 96/ bestätigen, daß eine hohe Produktivität durch den intensiven Einsatz von Werkzeugen und modernen Methoden erreicht wird.

Stabile Anforderungen

Sind die Anforderungen an ein Produkt stabil, dann wirkt sich das auf die Produktivität im Verhältnis 1:1,78 aus /Boehm 87, S. 47/.

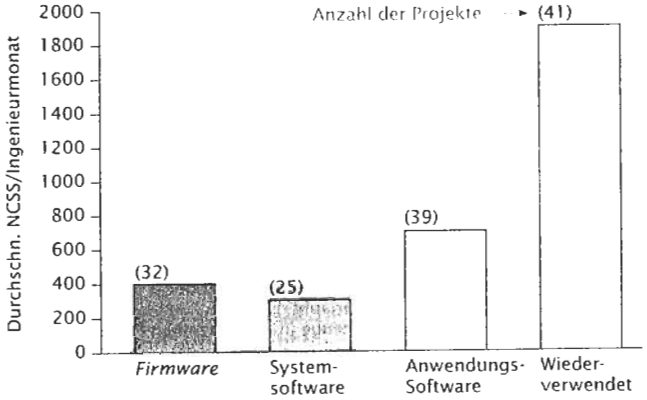
Wiederverwendun

Die Wiederverwendung von Software erlaubt große Produktivitäts- und Qualitätssteigerungen.

Bei HP wird unter wiederverwendeter Software folgendes verstanden: Software, die in ein Produkt eingebracht wird und vorher in einem anderen Produkt oder einem anderen Teil desselben Produkts einwandfrei arbeitete.

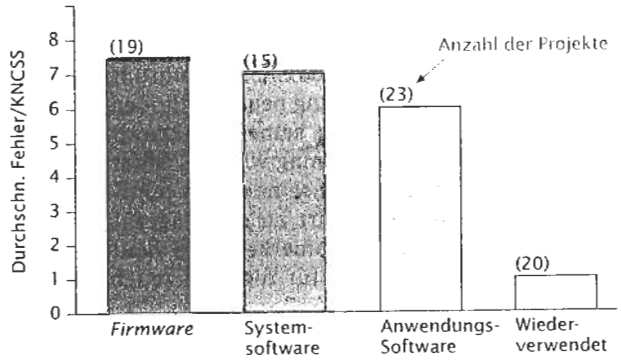
Die möglichen Produktivitätssteigerungen zeigt Abb. 1.4-4.

Abb. 1.4-4: HP-Produktivität in Abhängigkeit von Anwendungskategorien (137 Projekte insgesamt) /Grady 92, S. 43/



Die Säule »Wiederverwendet« enthält Projekte, bei denen der Anteil von wiederverwendetem Code mehr als 75 Prozent beträgt. Abb. 1.4-5 zeigt die entdeckte Fehlerquote vor der Produktfreigabe bezogen auf Anwendungskategorien.

Abb. 1.4-5: HP-Fehlerquote vor der Produktfreigabe in Abhängigkeit von Anwendungkategorien (77 Projekte insgesamt) Grady, Caswell 87, S. 112/



KNCSS = 1000 NCSS

Faustregel

Als Faustregel gibt /Grady 92, S. 14/ folgendes an:

- Projekte, die hauptsächlich aus wiederverwendeter Software bestehen, benötigen ungefähr 1/4 der Zeit und der Ressourcen, die eine Neuentwicklung benötigt.

Es gibt keinen Ersatz für Erfahrung. Besonders wichtig sind die Erfahrungen in der Definitionsphase. Wo immer möglich, sollten geprüfte Softwaremodelle und -komponenten wiederverwendet werden.

Alle anderen Teile, mit denen das Entwicklungsteam keine früheren Erfahrungen hat, sind am schwierigsten vorhersagbar und verursachen die größten Zeitprobleme.

Zeitrestriktionen

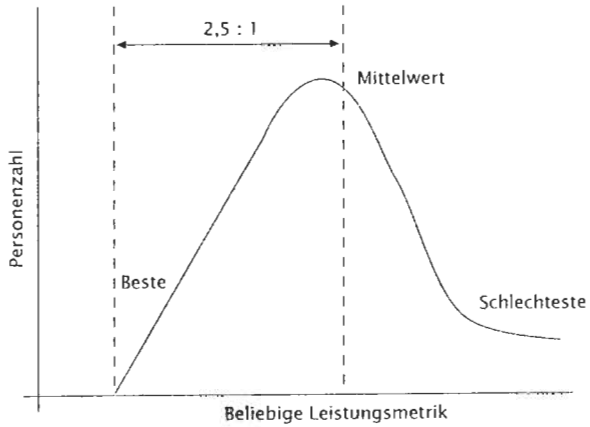
Zeitrestriktionen Abschnitt 5.5.10 Projektdauer

Zeitrestriktionen bzw. enge Termine wirken sich im Verhältnis 1:1,23 negativ auf die Produktivität aus. Je länger ein Projekt dauert, desto mehr sinkt die Produktivität /Maxwell, Wassenhove, Dutta 96/.

Mitarbeitereinflüsse

Verschiedene empirische Untersuchungen haben große Produktivitätsunterschiede zwischen Software-Ingenieuren nachgewiesen. Abb. 1.4-6 zeigt die Bandbreite der individuellen Leistungen.

Abb. 1.4-6: Produktivitätsunterschiede zwischen einzelnen Mitarbeitern /DeMarco, Lister 87, S. 52/



Aus diesen Daten leiten /DeMarco, Lister 87, S. 51f./ folgende drei Grundregeln ab:

Regeln

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

Diese Regeln gelten für fast jede beliebige Leistungsmetrik. Die bessere Mitarbeiterhälfte erledigt eine Arbeit in der Hälfte der Zeit, verglichen mit der schlechteren Mitarbeiterhälfte. Die fehleranfälligere Hälfte macht mehr als zwei Drittel der Fehler usw.

Abb. 1.4-7 bestätigt diese Regeln. Sie zeigt die Ergebnisse eines Programmierwettbewerbs. Aufgetragen ist die Verteilungsfunktion über die Zeit, die die Teilnehmer benötigten, um den ersten Meilenstein der Aufgabe (fehlerfreie Übersetzung, fertig zum Test) zu erreichen.

Die besten Leistungen lagen um den Faktor 2,1 über dem Durchschnitt. Die Hälfte über dem Mittelwert schlug die anderen im Verhältnis 1,9:1.

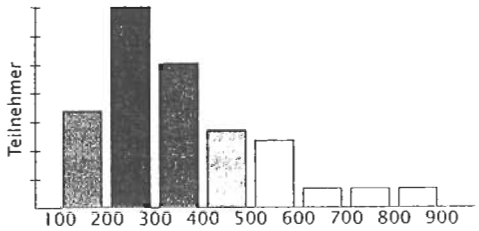


Abb. 1.4-7: Individuelle Leistungsunterschiede /DeMarco, Lister 87, S. 53/

In /Boehm 87, S. 47/ wird eine Produktivitätsspannbreite zwischen Mitarbeitern im Verhältnis 1:4,18 angegeben.

Es gibt unterschiedliche Meinungen darüber, welche Faktoren diese Produktivitätsunterschiede verursachen. In /DeMarco, Lister 87, S. 53f/ wird angegeben, welche Faktoren *wenig* oder *gar keine* Korrelation mit der Leistung zeigten:

- Programmiersprachen
  Es zeigte sich innerhalb der einzelnen Gruppen von höheren Programmiersprachen die gleiche Leistungsverteilung wie über alle Sprachen hinweg betrachtet.
- Berufserfahrung
  Zwischen den Jahren an Berufserfahrung und den erbrachten Leistungen gab es keine sichtbaren Zusammenhänge. Schlechter sind nur Mitarbeiter, die weniger als sechs Monate Erfahrung in der verwendeten Programmiersprache hatten.
- Anzahl der Fehler
  Die Programmierer, die keine Fehler in ihren Programmen hatten, benötigten, als Gruppe betrachtet, nicht mehr Zeit für ihre Ergebnisse als die anderen Programmierer.
- Gehalt
  Die Beziehung zwischen Gehältern und Leistung ist nur schwach ausgeprägt. Die Hälfte über dem Mittelwert verdient ungefähr 10 Prozent mehr als die Hälfte unter dem Mittelwert, war aber mindestens doppelt so gut.

Nach /Boehm 87, S. 47/ wirkt sich die Spracherfahrung und die Anwendungserfahrung auf die Produktivität aus:

- Ein Mitarbeiter mit Erfahrung in der verwendeten Programmiersprache ist im Verhältnis 1,2:1 produktiver als ein Mitarbeiter ohne die entsprechende Erfahrung.
- Ein Mitarbeiter mit Erfahrung auf dem Anwendungsgebiet ist im Verhältnis 1,57:1 produktiver als ein Mitarbeiter ohne entsprechende Anwendungserfahrungen.

Nach /Maxwell, Wassenhove, Dutta 96/ hat die Programmiersprachenerfahrung *keinen* signifikanten Einfluß auf die Produktivität.

Managementeinflüsse

Das Software-Management hat Einflüsse sowohl auf die Mitarbeiter als auch auf den Entwicklungsprozeß.

Einen oft unterschätzten Einfluß auf die Produktivität haben die **physische Arbeitsumgebung** und die **Arbeitsplatzausstattung.** /DeMarco, Lister 91/ haben in ihrem Buch die wesentlichen Faktoren dazu zusammengestellt.

Ein Mitarbeiter ist umso produktiver

- je ruhiger sein Arbeitsplatz ist,
- je weniger er gestört wird,
- je besser die Privatsphäre gewahrt ist,
- je größer der Arbeitsplatz ist.

Tab. 1.4-1 zeigt im Vergleich die Arbeitsplatzfaktoren des besten Viertels eines Programmierwettbewerbs und des schlechtesten Viertels.

Arbeitsplatzfaktoren	bestes Viertel der Teilnehmer	schlechtestes Viertel der Teilnehmer
1 Wieviel Arbeitsplatz steht Ihnen zur Verfügung?	7m²	4.1m²
2 Ist es annehmbar ruhig?	57% ja	29% ja
3 Ist Ihre Privatsphäre gewahrt?	62% ja	19% ja
4 Können Sie Ihr Telefon abstellen?	52% ja	10% ja
5 Können Sie Ihr Telefon umleiten?	76% ja	19% ja
6 Werden Sie oft von anderen Personen grundlos gestört?	38% ja	76% ja

Tab. 1.4-1: Der Arbeitsplatz der Besten und der Schlechtesten /DeMarco, Lister 91, S. 57/

Um produktive Ingenieurarbeit zu erledigen, wie Analyse, Entwurf, Programmierung, Schreiben von Handbüchern, muß man »in Fahrt« kommen. »In Fahrt sein« ist ein Zustand tiefer, fast meditativer Versunkenheit. Nur wenn man in Fahrt ist, geht die Arbeit wirklich voran. Man fühlt eine leichte Art von Euphorie und verliert das Zeitgefühl.

Bevor man diesen Zustand erreicht, benötigt man ca. 15 Minuten voller Konzentration. In diesem Zeitraum ist man besonders anfällig für Störungen und Unterbrechungen. In dieser Phase erledigt man noch keine Arbeit im eigentlichen Sinne. Ein Telefonanruf und die 15 Minuten Eintauchphase beginnen wieder von vorne.

Hieraus ergeben sich folgende Konsequenzen:

- Das Telefon eines Mitarbeiters muß für bestimmte Zeiträume abstellbar sein.
- Die Mitarbeiter sollten Einzelzimmer haben.

Untersuchungen bei IBM haben eine Produktivitätssteigerung von ungefähr 11 Prozent ergeben, bei TRW von 8 Prozent, wenn die Mitarbeiter Einzelzimmer besitzen /Boehm 87, S. 50/.

Zeitverteilung

Zwei häufige Mißverständnisse sind im Zusammenhang mit Einzelzimmern noch auszuräumen:

Einzelzimmer und Teamarbeit sind kein Widerspruch. Untersuchungen haben gezeigt, daß Mitarbeiter in 30 Prozent ihrer Zeit alleine arbeiten, in 50 Prozent der Zeit zu zweit und in 20 Prozent der Zeit in größeren Gruppen /DeMarco, Lister 91, S. 73/. Eine Untersuchung bei HP hat die Zeitverteilung zwischen verbalen Aktivitäten, ruhigen Aktivitäten und Nicht-Projekt-Aktivitäten ermittelt (Abb. 1.4-8).

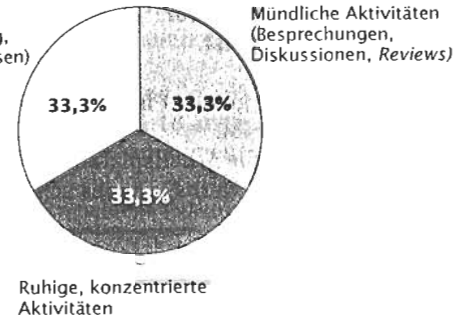


Abb. 1.4-8: Zeitaufteilung eines Software-Ingenieurs /Grady, Caswell 87, S. 37/

Für die rund 30 Prozent Einzelarbeit bzw. ruhige, konzentrierte Arbeit benötigen die Mitarbeiter Einzelzimmer, für die restliche Zeit Besprechungsräume.

Oft glaubt man in, Großraumbüros den Geräuschpegel durch Musik überspielen zu können. Die Wahrnehmung von Musik erfolgt in der rechten Gehirnhälfte, während die Abwicklung sequentieller Vorgänge in der linken Gehirnhälfte erfolgt. Kreative Erkenntnisse, geniale Ideen und Aha-Erlebnisse kommen aus der rechten Gehirnhälfte. Ist diese jedoch mit der Wahrnehmung von Musik beschäftigt, dann sind die Chancen für kreative Durchbrüche vertan /DeMarco, Lister 91, S. 91/.

Firmenkultur

Durch die Gestaltung der **Firmenkultur** beeinflußt das Management indirekt die Produktivität. /DeMarco, Lister 87, S. 55f./ haben bei Programmierwettbewerben zwischen Firmen festgestellt:

- Die beste Organisation, d.h. diejenige, die den besten Durchschnitt aller Teilnehmer zeigte, arbeitete 11,1 mal schneller als die schlechteste. Zusätzlich zu dem Arbeitstempo bestanden alle Teilnehmer der besten Organisationen auch den Abnahmetest mit ihren Produkten.

Bei den Programmierwettbewerben bildeten je zwei Programmierer aus einer Firma ein Team. Sie arbeiten aber nicht zusammen, sondern konkurrieren eher miteinander, aber auch gegen alle anderen Teams.

Die Leistungen der Teampartner lagen durchschnittlich nur um 21 Prozent auseinander. Dieses Ergebnis spricht dafür, daß das gleiche Umfeld und die gemeinsame Firmenkultur als Hintergrund ausschlaggebend sind.

Gute Software-Ingenieure »scharen« sich in bestimmten Firmen zusammen, die schlechten in anderen. Bei den schlechten Firmen stimmt offensichtlich etwas in deren Arbeitsumfeld oder in deren Firmenkultur nicht. Gute Software-Ingenieure werden von solchen Firmen nicht angezogen oder können auf Dauer nicht gehalten werden. Den wenigen guten Mitarbeitern, die dort noch arbeiten, macht es das Umfeld immer schwerer, gute Ergebnisse abzuliefern.

Die Firmenkultur beeinflußt auch wesentlich die Einführung von Innovationen. Je liberaler eine Firma ist, desto leichter lassen sich Innovationen einführen.

Kapitel 5.6

Mit wachsender **Teamgröße** sinkt die Produktivität /Maxwell, Wassenhove, Dutta 96/.

Kapitel 5.3 Teamgröße

1.5 Produktivität und Qualität

Produktivität und Qualität beeinflussen sich. Es gibt zwei konträre Meinungen über die Art der Abhängigkeit:

- Hohe Qualitätsanforderungen verringern die Produktivität.
- Hohe Qualitätsanforderungen verbessern die Produktivität.

Die häufig schlechte Qualität heutiger Software-Produkte läßt vermuten, daß die erste Meinung stimmt. Unbestritten ist, daß konstruktive und analytische Qualitätssicherungsmaßnahmen Kosten verursachen. Auf der anderen Seite führt eine bessere Produktqualität aber zur Einsparung von Wartungskosten (Abb. 1.5-1).

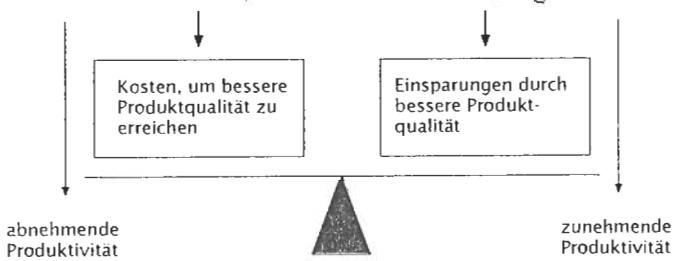


Abb. 1.5-1: Zusammenhang zwischen Produktivität und Qual

Damit die Produktivität zunimmt, müssen die durch höhere Qualität bedingten Einsparungen größer sein als die zusätzlichen Kosten.

Das Einsparungspotential ist groß. Auch heute entfallen oft 2/3 aller Lebenszyklus-Kosten auf die Wartung und Pflege eines Software-Produkts und nur 1/3 auf die eigentliche Entwicklung. Eine HP-Kundenumfrage ergab einen Wartungsanteil von durchschnittlich 75 Prozent am Gesamtaufwand /Grady 92, S. 17/.

Wie Abb. 1.3-2 zeigt, beeinflußt die Qualität auch den Produktwert. Probleme bei den Einsparungen ergeben sich in zweifacher Hinsicht:

- Wie quantifiziert man Software-Qualität?
- Wer bezahlt die Wartung?

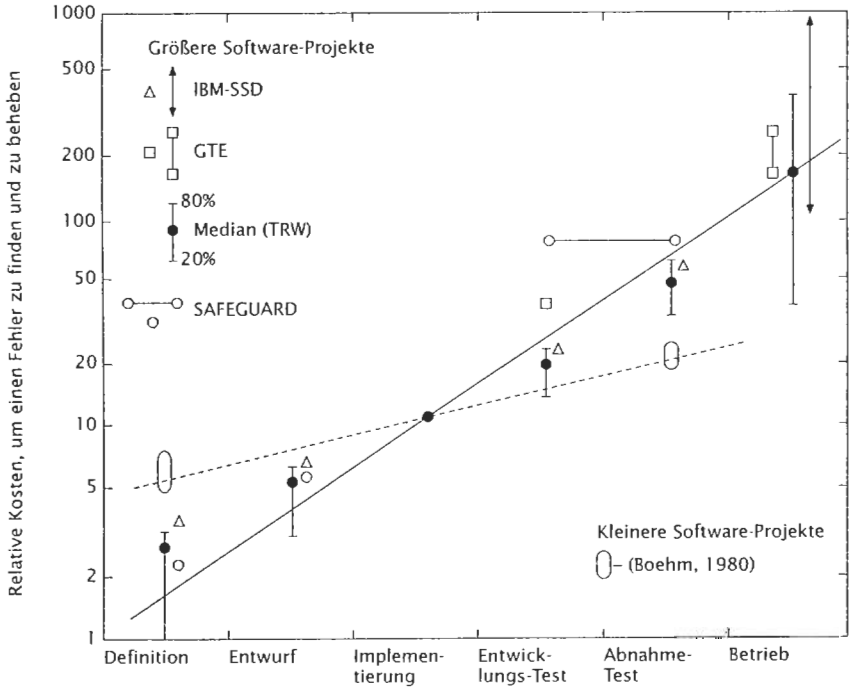
Die Software-Qualität kann man heute schlecht quantifizieren und messen. Als pragmatischer Ansatz werden die Wartungskosten ermittelt. Ist jedoch der Kunde bereit, hohe Wartungskosten zu bezahlen, dann hat der Software-Produzent kein Interesse daran, die Qualität zu verbessern. Oft verdient er mehr an der Wartung als am Verkauf des eigentlichen Produkts.

Ähnlich sieht die Situation aus, wenn innerhalb einer Firma die Budgets für Entwicklung und Wartung getrennt sind. Der Entwicklungsleiter ist dann nicht motiviert, gute Qualität zu produzieren, da dies Kosten verursacht, er aber an den Einsparungen nicht partizipiert.

Auf der Kostenseite haben empirische Untersuchungen gezeigt, daß eine frühzeitige Fehlerentdeckung und -behebung die kostengünstigste ist. /Boehm 81, S. 40/ hat mehrere Untersuchungen in einer Grafik zusammengefaßt (Abb. 1.5-2).

Abb. 1.5-2: Kostenverlauf der Fehlerbehebung

Die Grafik zeigt die relativen Kosten, um Fehler zu korrigieren oder Software-Änderungen vorzunehmen, als Funktion der Phase, in der die Korrekturen oder Änderungen vorgenommen werden. Kostet

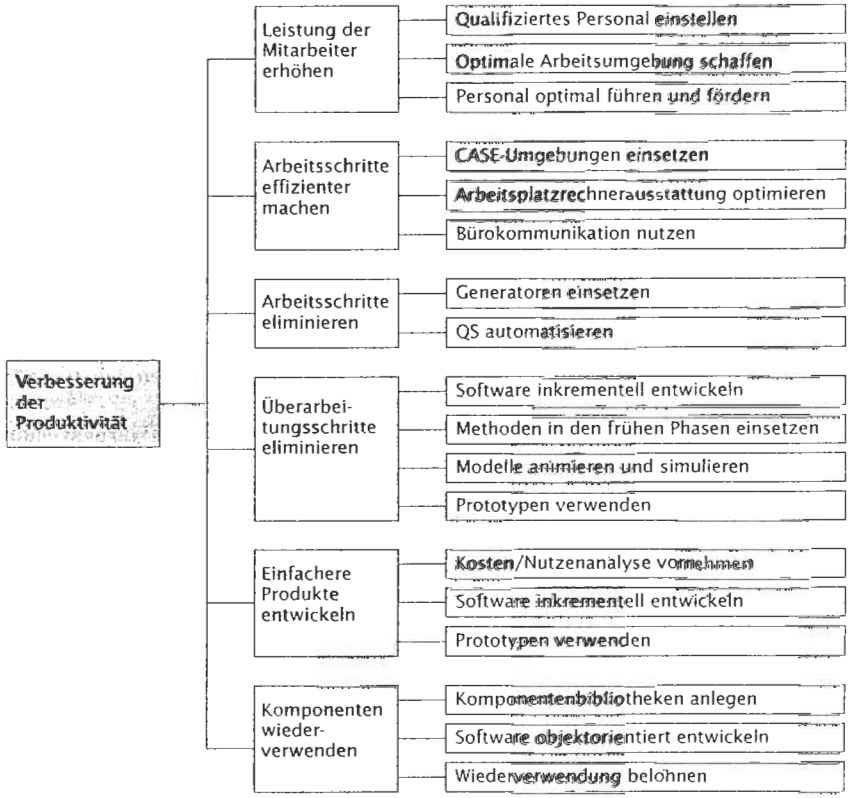


z.B. die Korrektur eines Anforderungsfehlers in der Definitionsphase 20,- DM, dann kostet die Behebung desselben Fehlers in der Betriebsphase 2.000,- DM, d.h. das Hundertfache. Jede Investition in die konstruktive und analytische Qualitätssicherung der frühen Entwicklungsphasen spart bis zum Hunderfachen an Kosten in der Wartungsphase.

1.6 Maßnahmen zur Produktivitätssteigerung

Die Möglichkeiten zur Produktivitätsverbesserung faßt /Boehm 87, S. 49/ in sechs Kategorien zusammen. In Anlehnung an seine Vorschläge zeigt Abb. 1.6-1 mögliche Maßnahmen zur Steigerung der Produktivität.

Abb. 1.6-1: Maßnahmen z Verbesserung Produktivität



Produktivität Quotient aus produzierten Ergebnissen und eingesetzten Ressourcen. Anstatt der produzierten Ergebnisse kann auch der Produktwert, d.h. der Wert, den das Produkt für den Benutzer hat, verwendet werden.

Die Software-Entwicklung unterscheidet sich signifikant von anderen Produktionsprozessen. Das Management einer Software-Entwicklung unterscheidet sich daher auch in vielen Dingen von dem Management anderer Ingenieurprojekte.

Die klassischen Managementaufgaben – Planung, Organisation, Personal, Leitung, Kontrolle – sind in angepaßter Form auch bei einer Software-Entwicklung auszuüben.

Ein Problem beim Software-Management ist die Definition der Produktivität. Die Faktoren, die die Produktivität beeinflussen, sind erst teilweise bekannt. Diese Einflüsse lassen sich den Kategorien Produkteinflüsse, Prozeßeinflüsse, Mitarbeitereinflüsse und Managementeinflüsse zuordnen.

Am stärksten wird die Produktivität von der Qualifikation der Mitarbeiter, der Firmenkultur und dem Grad der Wiederverwendung von Software-Komponenten beeinflußt.

Der Produktivitätsgewinn durch den Einsatz von CASE-Umgebungen wird oft überschätzt. CASE-Umgebungen bewirken vor allem eine Qualitätsverbesserung und eine Verbesserung der Wartungsproduktivität.

Unterschätzt werden dagegen die Einflüsse, die die physische Arbeitsumgebung, die Arbeitsausstattung und die Firmenkultur auf die Produktivität haben. Ein Einzelzimmer für jeden Mitarbeiter und ein abstellbares Telefon steigern die Produktivität.

Die gleichzeitige Steigerung der Produktivität und der Qualität wird oft als Widerspruch angesehen. Die Vermeidung von Fehlern – vor allem in den frühen Phasen – durch konstruktive und analytische Qualitätssicherungsmaßnahmen führt zu massiven Einsparungen in der Wartungsphase. Verbesserungen der Qualität führen daher in der Regel auch zu einer Verbesserung der Produktivität.

Maßnahmen zur Produktivitätssteigerung erfordern Verbesserungen in folgenden Kategorien: Leistung der Mitarbeiter erhöhen, Arbeitsschritte effizienter machen, Arbeitsschritte eleminieren, Überarbeitungsschritte eleminieren, einfachere Produkte entwickeln, Komponenten wiederverwenden.

/Boehm 81/
Boehm B.W., *Software-Engineering Economics*, Englewood Cliffs: Prentice Hall 1981, 767 Seiten
Es wird das Modell COCOMO zur Aufwandsschätzung detailliert beschrieben. Mitenthalten ist die Analyse von Einflußfaktoren auf die Produktivität.

/De Marco, Lister 87/
De Marco T., Lister T., *Peopleware*, New York: Dorset House Publishing Co. 1987, 188 Seiten
Die Autoren beschreiben die menschlichen Einflußfaktoren bei der Software-Entwicklung. Ein Muß für jeden Software-Manager!

/DeMarco, Lister 91/
DeMarco T., Lister T., *Wien wartet auf Dich! Der Faktor Mensch im DV-Management*, München: Carl Hanser Verlag 1991, 216 Seiten (Orginalausgabe /De Marco, Lister 87/)

/Grady 92/
Grady R.B., *Practical Software Metrics for Project Management and Process Improvement*, Englewood Cliffs: Prentice Hall 1992, 270 Seiten
Der Autor hat bei der Firma HP an der Einführung von Metriken mitgewirkt. Er berichtet über Erfahrungen. In dem Buch sind viele Metriken angegeben, die helfen, den Software-Prozeß zu verstehen. Viele Hinweise und Faustregeln für Software-Manager. Sehr zu empfehlen.

/Grady, Caswell 87/
Grady R.B., Caswell D.L., *Software Metrics: Establishing a Company-Wide Program*, Englewood Cliffs: Prentice Hall 1987, 288 Seiten
Ausführliche Darstellung, wie bei HP ein Metrik-Programm unternehmensweit eingeführt wurde.

/Sneed 87/
Sneed H.M., *Software-Management*, Köln: Verlagsgesellschaft Rudolf Müller GmbH 1987, 301 Seiten
Behandelt die Themen Grundlagen, Wirtschaftlichkeit, Produktivität, Qualität, Systemplanung, Systemdefinition, Projektkalkulation, Projektplanung & -steuerung sowie Systemverwaltung.

/Wallmüller 90/
Wallmüller E., *Software-Qualitätssicherung in der Praxis*, München: Carl Hanser Verlag 1990, 306 Seiten
Ausführliche Behandlung der konstruktiven und analytischen Qualitätssicherung.

Zitierte Literatur

/Boehm 87/
Boehm B.W., *Improving Software Productivity*, in: IEEE Computer, Sept. 1987, p. 43–57

/Gartner Group 90/
Gartner Group, *The Software Engineering Strategies Scenario*, Gartner Group Stanford CA USA 1990

/Jones 92/
Jones C., *CASE's missing elements*, in: IEEE Spectrum, June 1992, p. 38–41

/Mackenzie 69/
Mackenzie R.A., *The management process in 3-D*, in: Harward Business Review, Nov.-Dec 1969, Nachdruck in /Thayer 90/, S. 11–14.

/Maxwell, Wassenhove, Dutta 96/
Maxwell K. D., Wassenhove L. V., Dutta S., *Software Development Productivity of European Space, Military, and Industrial Applications*, in: IEEE Transactions on Software Engineering, Oct. 1996, pp. 706–718

/Thayer 90/
Thayer R.H., *Tutorial Software Engineering Project Management*, IEEE Computer Society Press, Los Alamitos CA., 1990

1 *Lernziel: Mögliche Software-Geschäftsstrategien und ihre Charakteristiken aufzählen können.*

Muß-Aufgabe
8 Minuten

Benennen Sie die drei wesentlichen Managementstrategien. Welche Auswirkungen haben diese auf den Software-Entwicklungsprozeß?

Muß-Aufgabe
25 Minuten

2 *Lernziele: Die fünf Managementfunktionen und die Hauptaktivitäten des Managements nennen können. Für gegebene Szenarien notwendige Managementaktivitäten identifizieren können.*
Ordnen Sie die folgenden Aufgaben den fünf Hauptmanagementfunktionen zu, und benennen Sie die notwendige Managementaktivität.

a Ein neuer Mitarbeiter soll eingestellt werden.
b Ihr Team hat mit großem Einsatzwillen einen wichtigen Termin bei einer Produktentwicklung halten können. Sie bedanken sich und loben den Einsatz.
c Für die Weiterentwicklung des von Ihnen betreuten Produktes gibt es zwei Alternativen. Sie wählen eine aus.
d Sie beginnen ein neues Projekt.
e Ihr Team ist an einem kritischen Punkt in einer Software-Entwicklung angekommen. Sie laden alle Mitarbeiter zum Essen ein und diskutieren das weitere Vorgehen.
f Sie legen fest, wie weit Ihr Produkt bis zum nächsten Quartalsbeginn entwickelt sein muß.
g Sie schlagen eine Gehaltsverbesserung für einen Mitarbeiter vor.
h Sie führen Wochenprotokolle ein, in denen der Produktfortschritt festgehalten wird.
i Sie beschließen den Einsatz eines neuen CASE-Systems für die nächste Software-Entwicklung.
k Sie gewähren ihren Mitarbeitern eine Fortbildung.
l Für ein neues Projekt bestimmen Sie einen Projektleiter und einen Qualitätssicherungsbeauftragten.

Muß-Aufgabe
10 Minuten

3 *Lernziel: Angeben können, welche Einflußfaktoren wie stark die Produktivität beeinflussen.*
In Ihrem Projekt ändern sich die Randbedingungen. Geben Sie anhand der aufgeführten Vergleichszahlen und empirischen Werte an, welche der folgenden Änderungen die Produktivität am meisten und welche sie am wenigsten beeinflussen:

a Das Produkt, das Sie entwickeln, ist nicht mehr komplex, sondern es ist als einfach einzustufen.
b Sie haben CASE-Werkzeuge über einen längeren Zeitraum eingeführt.
c Ihr Team besteht nicht aus durchschnittlichen, sondern aus den besten Mitarbeitern der Firma.
d Das Großraumbüro wurde abgeschafft, und alle Ihre Mitarbeiter haben Einzelzimmer.

Muß-Aufgabe
5 Minuten

4 *Lernziel: Die Unterschiede zwischen einer Software-Entwicklung und Produktentwicklungen in anderen Ingenieurbereichen erklären können.*
Welchen Unterschied gibt es bei der Software-Produktion im Gegensatz zu einer beliebigen Hardware-Produktion bezüglich der Kontrollierbarkeit des Projektfortschritts?

Muß-Aufgabe
5 Minuten

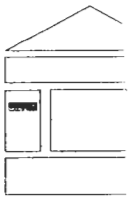
5 *Lernziel: Definitionen für die Software-Produktivität und ihre Problematik aufzeigen können.*
Eine höhere Produktivität kann u.a. durch:

- höhere Qualität,
- verringerten Aufwand,
- und vermehrte Ergebnisse,

erreicht werden. Sind alle Ziele gleichzeitig erreichbar?

Weitere Aufgaben befinden sich auf der beiliegenden CD-ROM.

2 Planung



wissen

- Angeben können, welche Anforderungen Meilensteine erfüllen müssen.
- Zweck und Darstellungsform von Gantt-Diagrammen angeben können.
- Die verschiedenen Möglichkeiten der Netzplanstrukturierung aufzählen können.
- Den Zusammenhang zwischen Prozeß-Architektur, Prozeß-Modell und Projektplan erklären können.
- Das methodische Vorgehen beim Aufstellen eines Projektplans erläutern können.
- Die verschiedenen Abhängigkeitsarten zwischen Vorgängen aufzeigen können.
- Netzpläne und Gantt-Diagramme mit ihren Eigenschaften erklären und vergleichen können.

verstehen

- Prozesse und Prozeß-Modelle in der angegebenen ETXM-Notation spezifizieren können.
- Einen MPM-Netzplan aufstellen, eine Vorwärts- und Rückwärtsrechnung durchführen sowie kritische Pfade ermitteln können.
- Termin- und kapazitätstreue Bedarfsoptimierungen von Ressourcen vornehmen können.
- Einfache Kostenplanungen durchführen können.
- Das verwendete Projektplanungssystem einsetzen können.
- Für kleinere Projekte eine vollständige Projektplanung computerunterstützt vornehmen können.

anwenden

2.1 Einführung

Planung

Planung ist die Vorbereitung zukünftigen Handelns. Sie legt vorausschauend fest, auf welchen Wegen, mit welchen Schritten, in welcher zeitlichen und sachlogischen Abfolge, unter welchen Rahmenbedingungen und mit welchen Kosten und Terminen ein Ziel erreicht werden soll:

»Planung ist Entscheiden im voraus, was zu tun ist, wie es zu tun ist, wann es zu tun ist und wer es zu tun hat« (in Anlehnung an /Koontz, O'Donnell 72/).

Planung ist keine einmalige Angelegenheit, sondern sie muß sich dynamisch und flexibel anpassen, wenn sich die Umgebung oder die Entwicklung ändert.

Jede erfolgreiche Software-Entwicklung beginnt mit einem guten Plan. Zukünftige Unsicherheiten und Änderungen, sowohl innerhalb der Entwicklungsumgebung als auch von externer Quelle, erfordern eine sorgfältige Planung, um die Risiken zu reduzieren.

Bei der Planung von Software-Entwicklungen sind drei Abstraktionsebenen zu unterscheiden:

rozeß-Architektur

- Es muß überlegt werden, wie der Ablauf von Software-Entwicklungen spezifiziert werden soll, welche Standard-Prozeßelemente es gibt und wie ihr Zusammenwirken beschrieben werden soll. Solche Überlegungen führen zu einer **Prozeß-Architektur.**

Prozeß-Modell

- Es muß für eine Firma einmal das generelle Vorgehen beim Entwikkeln eines Software-Produkts festgelegt werden.

Das Ergebnis einer solchen Planung können ein **Prozeß-Modell** – auch Vorgehensmodell genannt – oder mehrere Prozeß-Modelle sein, die einen Rahmen oder ein Muster für das Vorgehen vorgeben.

Kapitel 3.3

Solche Prozeß-Modelle werden im allgemeinen von einer Software-Prozeß-Gruppe im Rahmen einer Prozeß-Architektur erstellt und in bestimmten Intervallen überprüft und überarbeitet.

Projektplan

- Für jede konkrete Software-Entwicklung wird ein **Projektplan** erstellt. Er wird vom Projektleiter aufgestellt und orientiert sich an einem oder mehreren Prozeß-Modellen (Inkarnationen eines Prozeß-Modells).

2.2 Aufbau von Prozeß-Architekturen und Prozeß-Modellen

rozeß-Architektur

Eine **Prozeß-Architektur** legt den allgemeinen Rahmen für die Spezifikation von Software-Entwicklungsprozessen fest.

Prozeß

Sie besteht aus einer Standardmenge von fundamentalen Prozeßschritten. Regeln bestimmen, wie Prozesse beschrieben und in Beziehung gesetzt werden. Ein **Prozeß** beschreibt Aktivitäten, Methoden und Verfahren, die zur Entwicklung und Überprüfung von Software benötigt werden.

Das Grundelement einer Prozeß-Architektur ist das Prozeß-Einheitselement, wie es Abb. 2.2-1 zeigt. Die hier verwendete Notation zur Beschreibung von Prozessen nennt man ETXM-Spezifikation *(Entry, Task, Exit, Measurement)* /Humphrey 89, S. 257ff./.

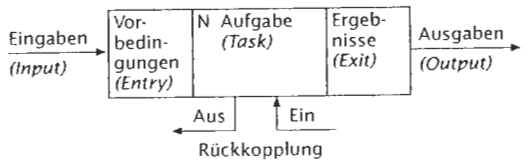


Abb. 2.2-1: Spezifikation des Prozeß-Einheitselements

Spezifikationen:

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

Prozeß-Modelle

Durch das geeignete Zusammenschalten von Standard-Prozeßelementen entstehen **Prozeß-Modelle**. Ein Prozeß-Modell stellt eine spezifische Ausprägung einer Software-Prozeß-Architektur dar. Es legt das methodische Vorgehen für die Entwicklung eines Software-Produkts fest. In Abhängigkeit von den zu entwickelnden Produktklassen kann es auch mehrere Prozeß-Modelle geben. Prozeß-Modelle können auf unterschiedlichen Abstraktionsebenen definiert werden.

Ein Prozeß-Modell kann als ein Metaplan angesehen werden, aus dem später konkrete Projektpläne abgeleitet werden.

Beispiel

Die Spezifikation des Definitionsprozesses zeigt Abb. 2.2-2.

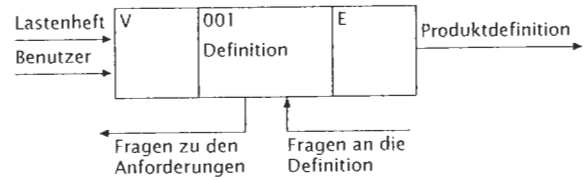
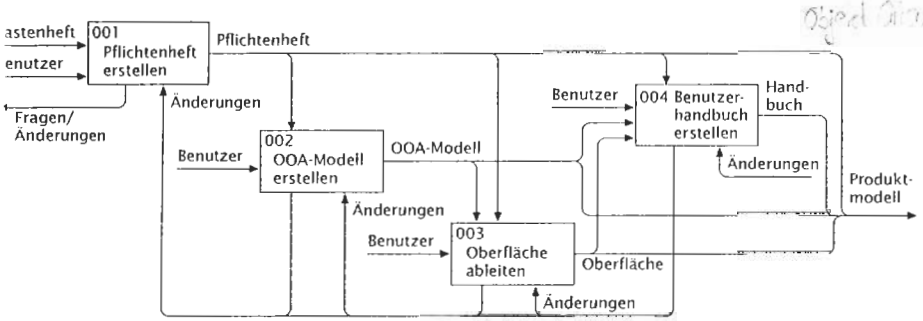


Abb. 2.2-2: Spezifikation des Definitionsprozesses

Spezifikationen:

Vorbedingungen:	Abgenommenes Lastenheft.
Ergebnisse:	Abgenommene Produktdefinition.
Rückkopplung:	
Ein:	Fragen an die Definition.
Aus:	Fragen zu den Anforderungen.
Aufgabe:	Auf der Grundlage des vorgegebenen Lastenheftes verfeinern die Systemanalytiker die Anforderungen und erstellen eine Produktdefinition.
Maße:	Maße der Aufgabe (Zeit, Termin), des Produkts (Umfang der Produktdefinition) und der Rückkopplung (Anzahl der Fragen).



Vorgang	001	002	003	004
Vorbedingungen	Abgenommenes Lastenheft	Pflichtenheft	Pflichtenheft OOA-Modell	Pflichtenheft, OOA-Modell, Oberfläche
Ergebnisse	Pflichtenheft	OOA-Modell	Oberfläche	Benutzerhandbuch
Rückkopplung in	Änderungen vom OOA-Modell, der Oberfläche und vom Handbuch	Änderungen durch Oberfläche, Handbuch, Benutzer	Änderungen durch Benutzerhandbuch	Änderungen durch Benutzer
Rückkopplung aus	Fragen an den Benutzer, Änderungen des Lastenhefts	Änderungen am Pflichtenheft	Änderungen am OOA-Modell, am Pflichtenheft	Änderungen am OOA-Modell, Oberfläche, Pflichtenheft
Aufgabe	Unter Verwendung des standardisierten Gliederungsschemas Pflichtenheft erstellen	OOA-Modell mit dem OO-Werkzeug erstellen	Aus dem OOA-Modell anhand bekannter Heuristiken eine Oberfläche ableiten, ein GUI-Werkzeug einsetzen	Nach didaktisch-methodischen Gesichtspunkten ein Handbuch erstellen
Maße	Umfang des Pflichtenheftes, Ausprägung der Qualitätsziele	Anzahl der Klassen, Anzahl der Verbindungen zwischen den Klassen, Anzahl Attribute und Operationen	Anzahl der Fenster, Interaktionselemente, Menüs	Umfang

Abb. 2.2-3: Verfeinerung des Definitionsprozesses

Abb. 2.2-3 zeigt eine detaillierte Betrachtung des Definitionsprozesses.

Prozeß-Modelle beschreiben die Ablauforganisation einer Software-Entwicklung. Welche verschiedenen Prozeß-Modelle man in der Software-Technik unterscheidet, wird im Kapitel Organisation behandelt.

Kapitel 3.3

Um die Qualität des Entwicklungsprozesses sicherzustellen, ist es sinnvoll und notwendig, jede Prozeßaktivität um eine Inspektionsaktivität zu ergänzen (Abb. 2.2-4).

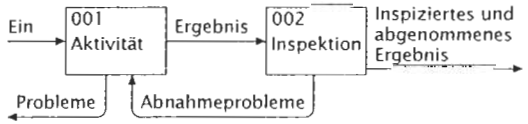


Abb. 2.2-4: Ergänzung jeder Prozeßaktivität um eine Inspektionsaktivität

Die abschließende Inspektionsaktion betrachtet die Inspektionsergebnisse und entscheidet, ob das Produkt an die nächste Entwicklungsphase weitergegeben, dem Prozeß »Aktivität« zur Fehlerbehebung zurückgegeben oder zur Vorgängerphase zurückgegeben wird.

2.3 Aufbau von Projektplänen

Projektplan

Ein **Projektplan** verfeinert, konkretisiert und ergänzt ein ausgewähltes Prozeß-Modell.

Jeder Prozeß in einem Prozeß-Modell wird zunächst projekt- und fachspezifisch verfeinert.

Vorgang

Dazu werden die in jedem Prozeß zu erledigenden Aufgaben in Vorgänge untergliedert. Ein **Vorgang** ist dabei eine in sich abgeschlossene identifizierbare Aktivität, die innerhalb einer angemessenen Zeitdauer durchgeführt werden kann. Für jeden Vorgang sind festzulegen:

- Name des Vorgangs (wenn neu).
- Erforderliche Zeitdauer zur Erledigung des Vorgangs.
- Zuordnung von Personal und Betriebsmitteln, die die Arbeit durchführen.
- Kosten und Einnahmen, die mit dem Vorgang zusammenhängen.

Um die Zeitdauer pro Vorgang angeben zu können, ist vorher eine Aufwandsschätzung des Gesamtprojekts durchzuführen. Der Aufwand ist dann auf die Vorgänge zu verteilen. Anschließend ist zu überlegen, mit wieviel Personal und mit wieviel Betriebsmitteln die Arbeit erledigt werden soll. Daraus ergeben sich sowohl die Zeitdauer des Vorgangs als auch die Zuordnung des Personals und der Betriebsmittel. Um zu einer Kostenkalkulation zu gelangen, müssen Kosten und Erlöse den Vorgängen zugeordnet werden.

Phase

Mehrere Vorgänge, die einen globalen Arbeitsabschnitt darstellen, werden oft zu einer **Phase** zusammengefaßt.

Meilensteine

Um eine Projektüberwachung zu ermöglichen, müssen Meilensteine festgelegt werden. **Meilensteine** kennzeichnen den Beginn und das Ende eines Projekts, den Abschluß jeder Phase und meist auch den Abschluß einer Gruppe von Vorgängen innerhalb einer Phase. Da ein Meilenstein eine Markierung und keine Aktivität ist, beansprucht er keine Zeit im Projektplan.

Meilensteine geben dem Projektleiter klare und eindeutige Anhaltspunkte für die Bewertung des Projektfortschritts. Sie motivieren das Projektteam, diese Teilziele in der festgelegten Zeit zu erreichen, und zeigen, daß ein bestimmter Teil der Arbeit abgeschlossen wurde.

Anforderungen Meilensteine

Um zu einer wirksamen Kontrolle des Entwicklungsfortschrittes zu gelangen, müssen Meilensteine folgende drei Anforderungen erfüllen:

1 Überprüfbarkeit
Mit dem Erreichen eines Meilensteins ist ein Teilprodukt fertiggestellt, d.h. jeweils ein Vorgang oder eine Reihe von Vorgängen resultiert in einem fertiggestellten oder weiterverarbeitbaren Teilprodukt. Da Aussagen wie »Das Programm ist zu 90 Prozent fertig codiert und zu 50 Prozent getestet« oder »Noch 8 Tage bis zur Fertigstellung« nicht überprüfbar sind, können darauf keine Meilensteine aufgebaut werden.

Beispiele

Beispiele für überprüfbare Meilensteine sind:

- Es sind zehn Testfälle aus der Spezifikation des Moduls A abgeleitet.
- Es sind drei Klassen einschließlich Attributen und Operationen entsprechend der OOA-Methode X definiert.
- Das Datumsprüfprogramm ist implementiert und besitzt eine Zweigüberdeckung von 90 Prozent.

2 Kurzfristigkeit
Vorgänge, die mit einem Meilenstein abschließen, müssen in ein bis vier Wochen erledigt werden können. Nur dadurch können Verzögerungen rechtzeitig erkannt werden. Diese Anforderung führt dazu, daß viele Meilensteine zu definieren sind.

3 Gleichverteilung
Meilensteine müssen kontinuierlich und gleichverteilt aufeinander folgen, damit jederzeit definierte Aussagen über den Entwicklungsstatus möglich sind. Ein Zweijahresprojekt in 24 Meilensteine zu untergliedern, wovon sechs im ersten Jahr und 18 im zweiten Jahr liegen, bietet kein geeignetes Planungs- und Kontrollraster. Eine Gleichverteilung dagegen bedeutet z.B., daß jeden Monat ein Meilenstein vorhanden ist.

Netzplan

Sowohl zwischen Vorgängen als auch zwischen Meilensteinen (Ereignissen) bestehen fachliche, terminliche und personelle Abhängigkeiten. Daher ordnet man sie grafisch in einem **Netzplan** an, um die Abhängigkeiten sichtbar zu machen.

Es werden im wesentlichen drei verschiedene Netzplanarten unterschieden, die in Abb. 2.3-1 vergleichend gegenübergestellt sind.

Aus dem Netzplan können verschiedene Auswertungen abgeleitet werden.

- Ein **vorgangsbezogenes** bzw. **aufgabenbezogenes Balkendiagramm** zeigt auf der Vertikalen die einzelnen Vorgänge und auf dem zugehörigen Balken die ausführenden Personen bzw. Stellen.
- Ein **personalbezogenes Balkendiagramm** zeigt alle Mitarbeiter auf der Vertikalen, so daß man sofort sieht, welche Vorgänge ein Mitarbeiter durchführt.

Gantt-Diagramme

Solche Balkendiagramme bezeichnet man auch als **Gantt-Diagramme.**

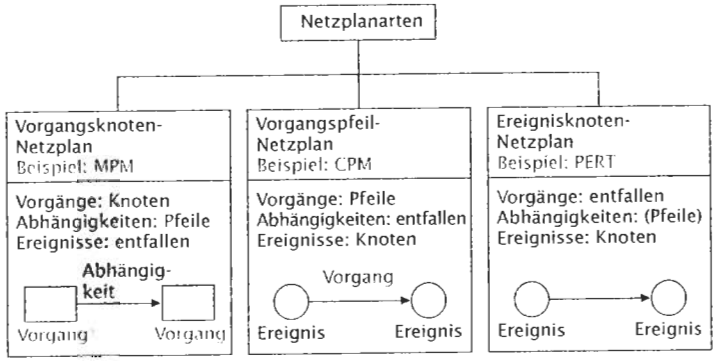


Abb. 2.3-1: Netzplanarten im Vergleich

Bei kleinen, übersichtlichen Projekten verzichtet man oft auf Netzpläne und stellt nur Balkendiagramme auf.

Zusätzlich zu diesen Darstellungsarten werden je nach Situation und Bedarf weitere Grafiken und Tabellen benutzt, um eine Projektplanung unter bestimmten Blickwinkeln zu betrachten.

Werkzeuge

Für die Projektplanung werden Planungssysteme verwendet, die es gestatten, die verschiedenen Grafiken und Tabellen zu erstellen und zu analysieren. Die Beispiele in den folgenden Abschnitten wurden mit dem Planungssystem *Microsoft Project* erstellt /MS Project 94/.

2.4 Zeitplanung mit MPM-Netzplänen

MPM-Netzplan

Die jüngste und in Europa am meisten verbreitete Netzplanart ist der Vorgangsknoten-Netzplan. MPM *(Metra Potential Method)* ist der bekannteste Vertreter des Vorgangsknoten-Netzplans.

Vorgang

Bei MPM-Netzplänen werden die Vorgänge als Rechtecke dargestellt. Die Verbindungspfeile symbolisieren die Abhängigkeiten zwischen den Vorgängen.

Meilenstein

Explizite Ereignisse werden nicht modelliert. Meilenstein-Ereignisse werden daher als eigene Vorgänge mit einer Null-Dauer dargestellt (in der Grafik durch einen zusätzlichen grauen Rahmen verdeutlicht).

Zeiten

Zu jedem Vorgang müssen Attribute festgelegt werden:

Die **Vorgangsdauer** ist die Arbeitszeit, die ein Vorgang insgesamt erfordert; z.B. benötigt der Vorgang »Pflichtenheft erstellen« 20 Tage.

Die **Arbeitsdauer** ist die Zeit, die eine Ressource für einen Vorgang aufwendet. Bearbeiten zwei Mitarbeiter parallel den Vorgang »Pflichtenheft erstellen«, dann beträgt die Arbeitsdauer pro Mitarbeiter 10 Tage. Arbeiten zwei Mitarbeiter 50 Prozent ihrer Zeit parallel an der Aufgabe »Pflichtenheft erstellen«, dann beträgt die Arbeitsdauer 20 Tage. Vorgangsdauer und Arbeitsdauer sind identisch, wenn

nicht mehrere Personen oder ein Teilzeit-Mitarbeiter an dem Vorgang arbeiten. Die längste Dauer bestimmt den Zeitplan. Eine Dauer kann in Minuten, Stunden, Tagen, Wochen oder Monaten gemessen werden. Die verwendete Zeiteinheit ist die Projektzeiteinheit.

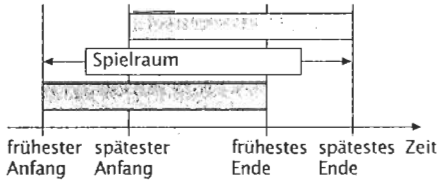
Der **Gesamtzeitraum** eines Vorgangs ist die Kalenderzeit, die für den Vorgang benötigt wird, einschließlich der arbeitsfreien Zeit.

Termine

Für jeden Vorgang und für jeden Meilenstein gibt es vier verschiedene Termintypen:

Geplante Termine legen fest, wann ein Vorgang beginnen und enden muß. Jeder Vorgang muß innerhalb einer bestimmten Zeitspanne ausgeführt werden (Abb. 2.4-1).

Abb. 2.4-1: Geplante und späte Termine



Ein Vorgang kann zwischen dem frühesten und dem spätesten Anfang beginnen. Dementsprechend endet der Vorgang zwischen dem frühesten und dem spätesten Ende. Der Spielraum gibt an, inwieweit der Vorgang zwischen dem frühesten und spätesten Anfang bzw. Ende verschoben werden kann.

Tatsächliche Termine zeigen den errechneten oder tatsächlichen Start- oder Endtermin eines Vorgangs.

Späte Termine geben den spätesten Zeitpunkt an, an dem ein Vorgang beginnen darf, ohne das Projektende zu verzögern.

Ein Vorgang kann auf bestimmte Termine festgelegt werden. Ein fester Anfangs- oder ein fester Endtermin beschränken den Zeitplan auf genau diese Termine.

Vorgegebene Termine sind Termine, die ursprünglich für Vorgänge geplant waren. Sie werden als Standard für den Vergleich mit neu geplanten Vorgängen oder ähnlichen Vorgängen in anderen Projekten verwendet.

Abhängigkeiten, Vorgangsbeziehungen

Abhängigkeiten bzw. Vorgangsbeziehungen legen die Reihenfolge von Vorgängen fest. Der Vorgang, von dem der aktuelle Vorgang in der sachlogischen Abfolge abhängig ist, ist sein Vorgänger. Der abhängige Vorgang heißt Nachfolger (Abb. 2.4-2).

Abb. 2.4-2: Abhängigkeitslinien

Es lassen sich vier Vorgangsbeziehungen unterscheiden:

- Normalfolge: **Ende-Anfang** (EA),
- Anfangsfolge: **Anfang-Anfang** (AA),
- Endfolge: **Ende-Ende** (EE) und
- Sprungfolge: **Anfang-Ende** (AE).

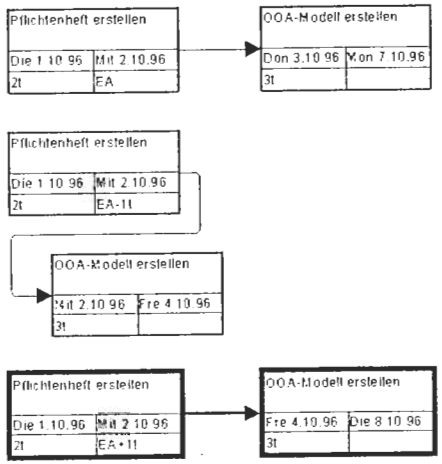
Zusätzlich können zusammengehörende Vorgänge **überlappt** oder **verzögert** werden, d.h. es kann ein positiver oder negativer **Zeitabstand**, auch Wartezeit genannt, angegeben werden.

Normalfolge: EA

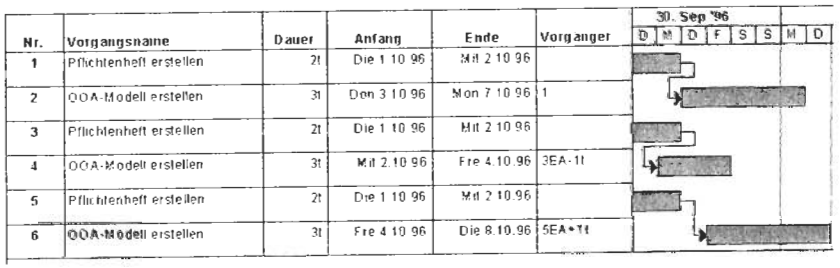
Bei der Beziehungsart Ende-Anfang kann ein Vorgang anfangen, sobald sein Vorgänger endet. Soll ein Vorgang anfangen, bevor sein Vorgänger beendet ist, dann wird ein negativer Zeitabstand angegeben. Durch einen positiven Zeitabstand kann der Anfang des Nachfolgers verzögert werden. Ein Zeitabstand kann in Zeiteinheiten oder als Prozentsatz der Dauer des Vorgängervorgangs ausgedrückt werden (Abb. 2.4-3).

Abb. 2.4-3: Beispiele für verschiedene EA-Beziehungen und ihre Darstellung

a Netzplandarstellung



b Gantt-Diagrammdarstellung

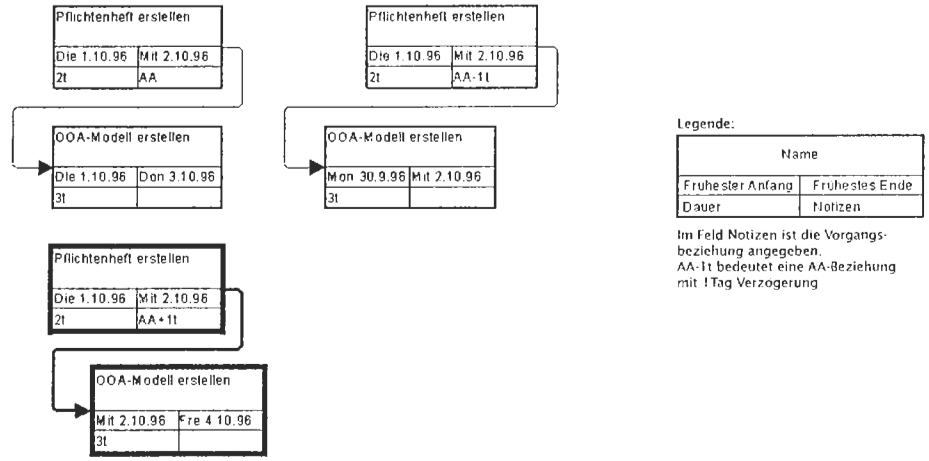


Nr.	Vorgangsname	Dauer	Anfang	Ende	Vorganger
1	Pflichtenheft erstellen	2t	Die 1.10.96	Mit 2.10.96	
2	OOA-Modell erstellen	3t	Don 3.10.96	Mon 7.10.96	1
3	Pflichtenheft erstellen	2t	Die 1.10.96	Mit 2.10.96	
4	OOA-Modell erstellen	3t	Mit 2.10.96	Fre 4.10.96	3EA-1t
5	Pflichtenheft erstellen	2t	Die 1.10.96	Mit 2.10.96	
6	OOA-Modell erstellen	3t	Fre 4.10.96	Die 8.10.96	5EA+1t

Anfangsfolge: AA

Bei einer Anfangsfolge fängt ein Vorgang an, sobald sein Vorgänger anfängt, d.h. die Vorgänge beginnen gleichzeitig. Zeitabstände können ebenfalls angegeben werden (Abb. 2.4-4).

a Netzplandarstellung



b Gantt-Diagrammdarstellung

Nr.	Vorgangsname	Dauer	Anfang	Ende	Vorganger
1	Pflichtenheft erstellen	2t	Die 1.10.96	Mit 2.10.96	
2	OOA-Modell erstellen	3t	Die 1.10.96	Don 3.10.96	1AA
3	Pflichtenheft erstellen	2t	Die 1.10.96	Mit 2.10.96	
4	OOA-Modell erstellen	3t	Mon 30.9.96	Mit 2.10.96	3AA-1t
5	Pflichtenheft erstellen	2t	Die 1.10.96	Mit 2.10.96	
6	OOA-Modell erstellen	3t	Mit 2.10.96	Fre 4.10.96	5AA+1t

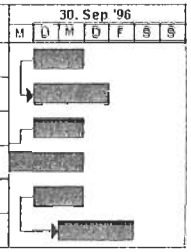
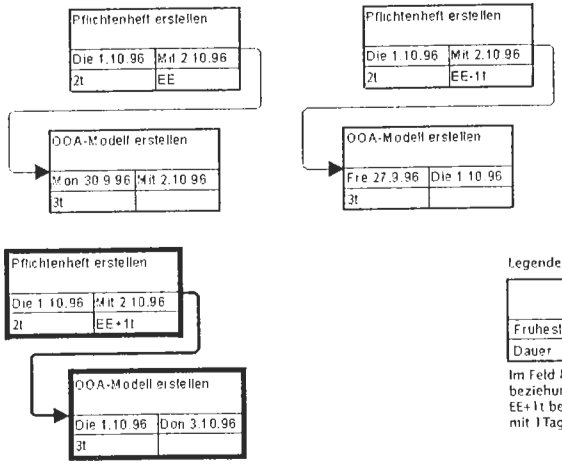


Abb. 2.4-4: Beispiele für verschiedene AA-Beziehungen und ihre Darstellung

Endfolge: EE

Bei der Beziehungsart Ende-Ende endet ein Vorgang, sobald sein Vorgänger endet. Überlappungen und Verzögerungen können angegeben werden (Abb. 2.4-5).

a Netzplandarstellung



b Gantt-Diagrammdarstellung

Nr.	Vorgangsname	Dauer	Anfang	Ende	Vorgan
1	Pflichtenheft erstellen	2t	Die 1.10.96	Mit 2.10.96	
2	OOA-Modell erstellen	3t	Mon 30.9.96	Mit 2.10.96	1EE
3	Pflichtenheft erstellen	2t	Die 1.10.96	Mit 2.10.96	
4	OOA-Modell erstellen	3t	Fre 27.9.96	Die 1.10.96	3EE-1t
5	Pflichtenheft erstellen	2t	Die 1.10.96	Mit 2.10.96	
6	OOA-Modell erstellen	3t	Die 1.10.96	Don 3.10.96	5EE+1t

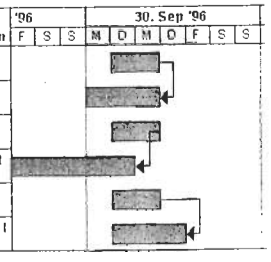


Abb. 2.4-5: Beispiele für verschiedene I Beziehungen u ihre Darstellu

Sprungfolge: AE

Bei einer Sprungfolge kann ein Vorgang enden, sobald sein Vorgänger anfängt (Abb. 2.4-6).

Abb. 2.4-6: Beispiele für verschiedene AE-Beziehungen und ihre Darstellung

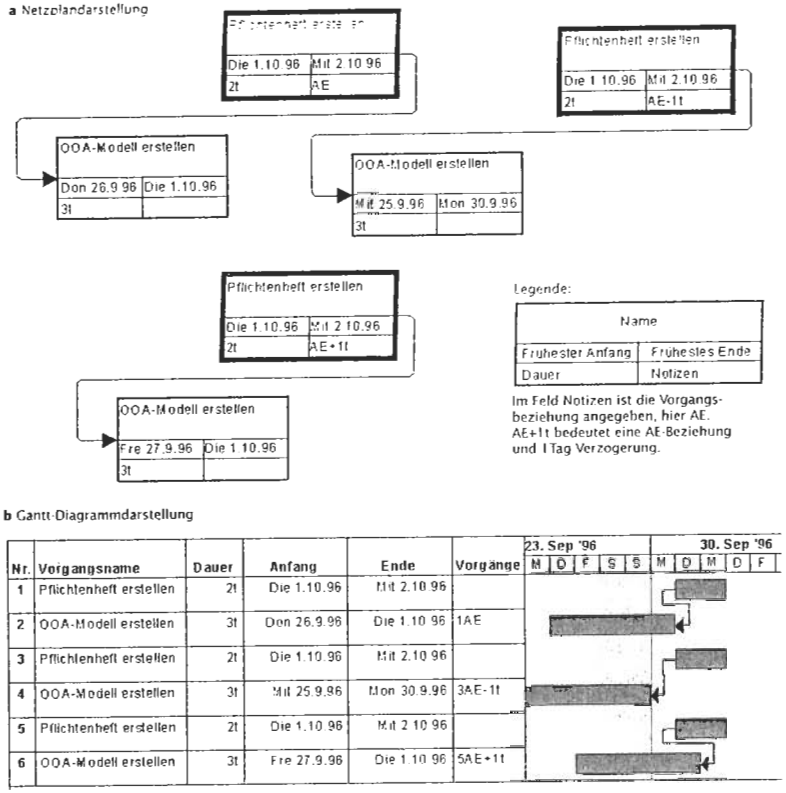


Abb. 2.4-7 zeigt nochmals die Beispiele im Zusammenhang.

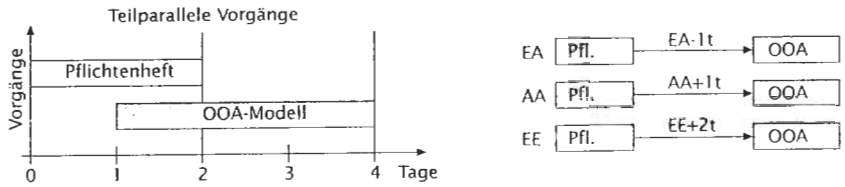


Abb. 2.4-7: Beziehungsarten im Vergleich dargestellt an einem Beispiel

Pufferzeit

Eine **Pufferzeit** ist die Differenz zwischen dem frühesten und spätesten Anfangstermin eines Vorgangs. Die Vorgangsdauer kann also um die Pufferzeit überschritten werden, ohne den Projektablauf

zu verzögern. Je größer die Pufferzeit ist, desto geringer ist die Wahrscheinlichkeit, daß sich ein Vorgang verzögert. Wird der späteste Endtermin nicht überschritten, dann verzögert sich der nachfolgende Vorgang nicht.

Es lassen sich zwei Pufferzeiten unterscheiden:

- Die **freie Pufferzeit** gibt die Zeitspanne an, um die sich ein Vorgang verzögern kann, ohne einen anderen Vorgang zu verzögern.
- Die **gesamte Pufferzeit** gibt die Zeitspanne an, um die ein Vorgang verzögert werden kann, ohne den Endtermin des Projekts zu beeinflussen.

Einschränkunge

Pufferzeiten entstehen dann, wenn es für den Anfang oder das Ende von Vorgängen Vorgangseinschränkungen gibt.

Folgende Einschränkungen werden oft vorgenommen:

- So früh wie möglich.
  Der Vorgang fängt so früh wie möglich an unter Berücksichtigung der sonstigen Einschränkungen und Beziehungsarten.
- So spät wie möglich.
  Der Vorgang fängt so spät wie möglich an.
- Ende nicht früher als.
  Der Vorgang endet am oder nach dem gegebenen Termin.
- Anfang nicht früher als.
  Der Vorgang fängt am oder nach dem gegebenen Termin an.
- Ende nicht später als.
  Der Vorgang endet am oder vor dem gegebenen Termin.
- Anfang nicht später als.
  Der Vorgang fängt am oder vor dem gegebenen Termin an.
- Muß enden am.
  Der Vorgang endet an einem bestimmten Termin.
- Muß anfangen am.
  Der Vorgang beginnt an einem bestimmten Termin.

Die beiden ersten Einschränkungen werden am häufigsten verwendet.

Wenn eine Einschränkung mit einer Vorgangsbeziehung im Konflikt steht, dann erhält bei vielen Planungssystemen die Einschränkung den Vorrang.

Kritischer Vorga

Besitzt ein Vorgang keine Pufferzeit, dann handelt es sich um einen **kritischen Vorgang.**

Kritischer Pfad

Bilden mehrere kritische Vorgänge eine Folge, dann liegt ein **kritischer Pfad** vor. Er zeigt, welche Vorgänge sorgfältig überwacht werden müssen, damit das Projekt termingerecht abgeschlossen werden kann.

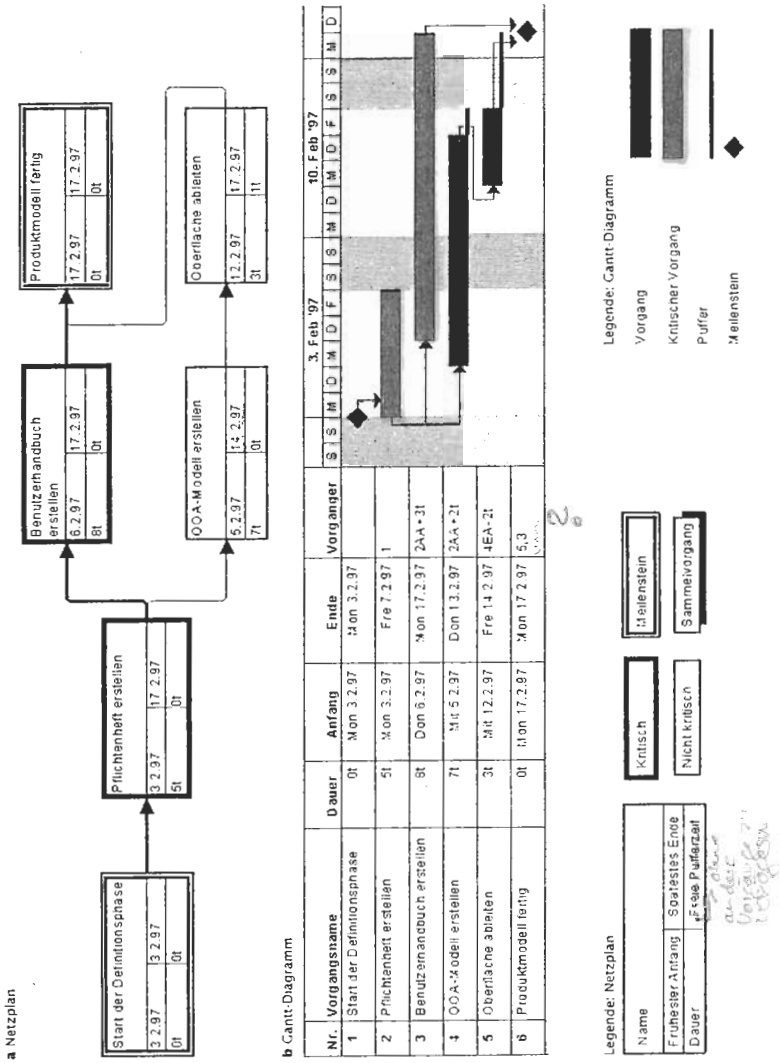
Wird ein kritischer Vorgang auf dem kritischen Pfad nicht an seinem spätesten Endtermin abgeschlossen, dann verzögert sich der Beginn aller Nachfolger. Ist die verlorene Zeit nicht durch schnellere Erledigung eines Nachfolgers aufzuholen, dann verzögert sich das Projektende um die Zeitspanne, um die der kritische Vorgang sein spätestes Ende überschritten hat.

Kritische Vorgänge und kritische Pfade werden im Netzplan hervorgehoben, z.B. durch eine farbige Darstellung.

Beispiel

Abb. 2.4-8 zeigt ein vereinfachtes Beispiel für die Erstellung eines Produktmodells.
Am Anfang und Ende befindet sich jeweils ein Meilenstein. Als frühester Starttermin wurde der 3.2.97 vorgegeben. Pro Vorgang wurde die geschätzte Dauer in Tagen angegeben. Zwischen den Vorgängen

Abb. 2.4-8: Netzplan und vorgangsbezogenes Gantt-Diagramm mit kritischem Pfad



Nr.	Vorgangsname	Dauer	Anfang	Ende	Vorgänger
1	Start der Definitionsphase	0t	Mon 3.2.97	Mon 3.2.97	
2	Pflichtenheft erstellen	5t	Mon 3.2.97	Fre 7.2.97	1
3	Benutzerhandbuch erstellen	8t	Don 6.2.97	Mon 17.2.97	2AA+3t
4	OOA-Modell erstellen	7t	Mit 5.2.97	Don 13.2.97	2AA+2t
5	Oberfläche ableiten	3t	Mit 12.2.97	Fre 14.2.97	4EA-2t
6	Produktmodell fertig	0t	Mon 17.2.97	Mon 17.2.97	5,3

a Netzplan

b Gantt-Diagramm

gibt es Anfang-Anfang- und Ende-Anfang-Abhängigkeiten. Als Beschränkung wurde für alle Vorgänge »so früh wie möglich« gewählt. Für die Vorgänge »OOA-Modell erstellen« und »Oberfläche ableiten« ergibt sich eine gesamte Pufferzeit von jeweils einem Tag, alle anderen Vorgänge sind kritische Vorgänge. Der Vorgang »Oberfläche ableiten« hat eine freie Pufferzeit von einem Tag.

Aus einem Netzplan kann automatisch ein vorgangsbezogenes Gantt-Diagramm abgeleitet werden, das auch Vorgangsplan genannt wird. Umgekehrt kann aus dem Gantt-Diagramm ein Netzplan hergeleitet werden.

Termindurchrechnung

Die Termindurchrechnung eines Netzplans führt zu einer zeitlichen Anordnung der Vorgänge unter Berücksichtigung der gegenseitigen Abhängigkeiten.

Vorwärtsrechn

Zum Bestimmen der frühesten Termine dient die **Vorwärtsrechnung**. Es wird vom Anfangszeitpunkt des Startvorgangs ausgegangen. Durch Addition mit dessen Dauer erhält man das früheste Ende für diesen Vorgang. Dieses bestimmt gleichzeitig – unter Berücksichtigung entsprechender Zeitabstände – die frühesten Anfangszeitpunkte für die Nachfolger-Vorgänge des Startvorgangs. Addiert man dazu die jeweiligen Vorgangsdauern, dann ergeben sich die zugehörigen frühesten Endzeitpunkte dieser Nachfolger-Vorgänge. Die analoge Durchrechnung der weiteren Vorgänge führt zu einem frühesten Endzeitpunkt für den Zielvorgang.

Rückwärtsrechnung

In einem zweiten Rechnungsgang, der **Rückwärtsrechnung,** werden die spätesten Zeitpunkte bzw. Termine bestimmt. Es ist von dem spätesten Endzeitpunkt des Zielvorgangs auszugehen. Dieser kann entweder als fester Termin oder durch die Projektdauer vorgegeben sein. Ist beides nicht der Fall, dann wird der durch die Vorwärtsrechnung ermittelte früheste Endzeitpunkt gewählt und mit dem spätesten Endzeitpunkt gleichgesetzt. Durch Subtraktion der Dauer des Zielvorgangs vom Endzeitpunkt ergibt sich der späteste Anfangszeitpunkt dieses Vorgangs. Dieser bestimmt gleichzeitig – unter Berücksichtigung eventueller Zeitabstände – die spätesten Endzeitpunkte seiner Vorgänger. Die Rechnung wird analog weitergeführt, bis der späteste Anfangszeitpunkt des Startvorgangs bestimmt ist.

Nach Abschluß der Vorwärts- und Rückwärtsrechnung liegen für jeden Vorgang folgende Termine fest:
- Frühester Anfang,
- Spätester Anfang,
- Frühestes Ende,
- Spätestes Ende.

Ein Netzplan ist zeitkonsistent, wenn keine negativen Puffer auftreten. Negative Puffer entstehen allein durch die Vorgabe von festen Terminen. Fehlen feste Termine, dann ergibt sich immer ein zeitkonsistenter Netzplan.

Netzplan-strukturierung

Ein Gesamtnetzplan für ein umfangreiches Projekt kann schnell unübersichtlich werden. Daher gibt es mehrere Möglichkeiten, Netzpläne zu strukturieren.

Bei der Netzplanunterteilung werden nach bestimmten Gliederungskriterien aus dem Gesamtnetzplan Teilnetzpläne gebildet. Bei einer organisationsorientierten Unterteilung erhält jede Organisationseinheit nur die von ihr zu bearbeitenden Vorgänge ausgewiesen. Eine projektorientierte Gliederung ist notwendig, wenn in einem Entwicklungsbereich mehrere Projekte durchgeführt werden. Beide Unterteilungsarten sind notwendig, um eine Multiprojektplanung vorzunehmen.

Bei der Netzplanverdichtung wird eine hierarchische Netzplanstruktur aufgebaut. Nach oben hin erfolgt eine Verdichtung der Netzplaninformationen. Bei einer Vorgangsreduktion werden abgeschlossene Vorgänge zu Sammelvorgängen zusammengefaßt, wenn die Detailliertheit abgeschlossener Vorgänge nicht mehr von Bedeutung ist. Zukünftige Vorgänge sind noch nicht detailliert.

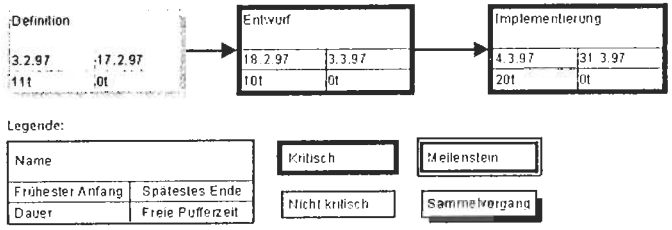
Ein Meilenstein-Netzplan liegt vor, wenn er nur die Meilenstein-»Vorgänge« enthält. Diese Information ist für das Management oft ausreichend. Ein Meilenstein-Netzplan kann auch hierarchisch aufgebaut sein.

Für ähnlich verlaufende Entwicklungsabschnitte können Standardnetzpläne, auch Projektvorlagen genannt, erstellt werden, die dann jeweils projektspezifisch adaptiert werden. Die Zusammenfassung mehrerer solcher Pläne ergibt einen Rahmennetzplan.

Beispiel

Der Netzplan der Abb. 2.4-8 wurde verdichtet und in einen übergeordneten Netzplan integriert (Abb. 2.4-9). Der Vorgang »Definition« ist ein Sammelvorgang, gekennzeichnet durch ein schattiertes Rechteck. Die Vorgangsinformationen werden aus dem detaillierten Netzplan in den Sammelvorgang übernommen.

Abb. 2.4-9: Netzplanverdichtung



2.5 Einsatzmittelplanung

Zur Durchführung der Vorgänge werden Einsatzmittel benötigt. Aufgabe der Einsatzmittelplanung ist es, den Bedarf an Einsatzmitteln vorherzusagen und durch Aufzeigen von Engpässen und Leerläufen eine Einsatzoptimierung zu erreichen.

Einsatzmittel sind

- Personal,
- Betriebsmittel (Maschinen, Materialien) und
- Geldmittel.

Personal- und Betriebsmittel werden oft unter dem Oberbegriff **Ressourcen** zusammengefaßt. Ressourcen

Bei einer Multiprojektplanung müssen die Einsatzmittel auf mehrere Projekte im Zeitablauf optimal aufgeteilt werden. Projektspezifische Über- und Unterdeckungen von Einsatzmitteln können so ausgeglichen werden.

Eine Projektdurchführung kann daran scheitern, daß die notwendigen Einsatzmittel nicht zeitgerecht vorhanden sind. Vorgänge, die zeitlich nicht kritisch sind, können durch Fehlen eines bestimmten Einsatzmittels kapazitätskritisch werden. Neben einem zeitkritischen Pfad kann es also auch einen kapazitätskritischen Pfad geben.

Aufgabe der Einsatzmittelplanung ist es daher, die Einsatzmittel auslastungsoptimal auf die einzelnen Vorgänge und Projekte zu verteilen.

Einsatzplanung des Personals

Bei der Personalplanung sind folgende Gesichtspunkte zu berücksichtigen: Personaleinsatz

- Qualifikation des Personals,
- verfügbare Personalkapazität,
- zeitliche Verfügbarkeit,
- örtliche Verfügbarkeit,
- organisatorische Zuordnung.

Bei der Einsatzplanung ist insbesondere die Qualifikation und die zeitliche Verfügbarkeit zu berücksichtigen. Aber auch die Teamzugehörigkeit *(»Never Change A Winning Team«)* und die Identifikation mit der zu erledigenden Aufgabe spielen eine wichtige Rolle. Bei der Personalplanung sollten auch ungewöhnliche Wege gegangen werden, z.B. »Lassen Sie eine behinderte Person zusätzlich in einem neuen Team mitarbeiten, und die Erfolgschancen für gute Zusammenarbeit sind größer« /De Marco, Lister 91, S. 181/. Kapitel 5.3

Ziel der Personaleinsatzplanung ist ein optimaler Personaleinsatz über die gesamte Projektlaufzeit hinweg. Es sollen möglichst keine Überlastungen und keine zu geringen Auslastungen einzelner Mitarbeitergruppen auftreten.

Ausgangsbasis für die Optimierung sind die Terminanforderungen.

termintreu

Eine **termintreue Einsatzplanung** liegt vor, wenn die Termine vom Auftraggeber festliegen und ermittelt werden muß, welche Personalkapazität in welcher zeitlichen Belegung erforderlich ist.

kapazitätstreu

Eine **kapazitätstreue Einsatzplanung** liegt vor, wenn das zur Verfügung stehende Personal auf der Auftragnehmerseite feststeht und ermittelt werden muß, welches der früheste Fertigstellungstermin bei optimalem Personaleinsatz ist.

Die Personaleinsatzplanung erfolgt in den Schritten:

1 Ermitteln des Personalvorrats,
2 Errechnen des Personalbedarfs,
3 Vergleich von Bedarf und Vorrat,
4 Optimierung der Auslastung.

1 Personalvorrat ermitteln

Ist es notwendig, die Qualifikation zu berücksichtigen, dann teilt man das zur Verfügung stehende Personal in Gruppen gleicher Qualifikation ein. Anschließend ordnet man diese Gruppen unter Berücksichtigung der geographischen und organisatorischen Randbedingungen den einzelnen Vorgängen zu. Abb. 2.5-1 zeigt eine mögliche Zuordnungsmatrix.

Abb. 2.5-1: Personalzuordnung nach Qualifikationen

Personalvorrat	5	1	2	Eingeplantes Personal
Qualifikation / Projektvorgänge	System-analytiker	Software-ergonom	Handbuch-autoren	
Pflichtenheft erstellen	3			3
OOA-Modell erstellen	3			3
Oberfläche ableiten		1		1
Benutzerhandbuch erstellen			1	1

Brutto-Zeitvorrat

Bei einer zeitgerechten Vorratsbetrachtung wird festgestellt, welche Personalkapazität je Zeiteinheit überhaupt verfügbar ist. Die zusätzliche Berücksichtigung der Qualifikation schränkt den Planungsspielraum weiter ein.

Der Brutto-Zeitvorrat je Zeiteinheit ergibt sich durch Berücksichtigung von

- Neueinstellungen,
- Kündigungen,
- Verrentungen,
- Versetzungen,
- Teilzeitarbeit und
- Arbeitszeitverkürzungen.

Netto-Zeitvorrat

Der Netto-Zeitvorrat ergibt sich aus dem Brutto-Zeitvorrat abzüglich der Fehl- und Ausfallzeiten wie Krankheit und Urlaub.

Oft werden die Fehl- und Ausfallzeiten als pauschaler Wert von der theoretischen Gesamtarbeitszeit abgezogen. Bei dieser Abzugsrechnung können entweder die Arbeitsmonate pro Jahr (Brutto-Rechnung) oder die Arbeitsstunden pro Monat (Netto-Rechnung) reduziert werden.

Setzt man im Durchschnitt sechs Wochen Urlaub sowie zwei Wochen Fehl- und Ausfallzeiten an, dann bleiben im Durchschnitt zehn Monate Arbeitszeit im Jahr übrig.

Brutto-Rechnung

Die durchschnittliche produktive Jahresarbeitszeit berechnet man wie folgt (Annahmen: 37-Stundenwoche, 1 Woche = 5 Arbeitstage, 1 Monat = 20,8 Arbeitstage):

37 Stunden/Woche = 7,4 Stunden/Tag
= 1 MT_{brutto}
154 Stunden/Monat = 1 MM_{brutto}

MT = Mitarbeitertag
MM = Mitarbeitermonat
MJ = Mitarbeiterjahr

Bei 10 MM_{brutto} im Jahr erhält man eine Brutto-Jahresarbeitszeit von

1.539 Stunden/Jahr = 1 MJ_{brutto}

Netto-Rechnung

Bei der Netto-Rechnung bleibt der Bezug auf 12 Monate im Jahr erhalten. Die Abzüge reduzieren hierbei die durchschnittlichen Arbeitsstunden pro Monat bzw. Tag.
Also erhält man bei 12 MM pro Jahr

1.539 Stunden/Jahr = 128 Stunden/Monat
= 1 MM_{netto}

und entsprechend

6,1 Stunden/Tag = 1 MT_{netto}

Der Zusammenhang zwischen der Brutto- und Netto-Rechnung ergibt sich durch den Produktivanteil:

$$\text{Produktivanteil} = \frac{\text{Netto-Stundenanzahl je Zeiteinheit}}{\text{Brutto-Stundenanzahl je Zeiteinheit}} \times 100\ [\%]$$

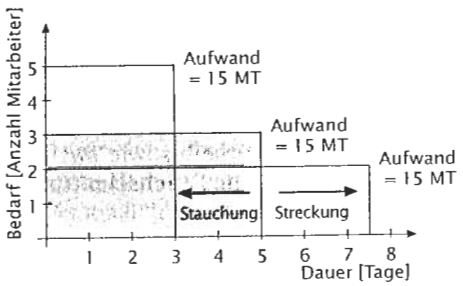
Für das Beispiel ergibt sich ein Produktivanteil von 83%.

2 Bedarfsberechnung

Die für einen bestimmten Vorgang benötigte Personalkapazität steht in einem direkten Verhältnis zu der Dauer, die für diesen Vorgang eingeplant wird (Abb. 2.5-2).

Rein rechnerisch ist z.B. ein Arbeitsvolumen von 15 MT von drei Mitarbeitern in fünf Tagen oder von zwei Mitarbeitern in 7,5 Tagen

Abb. 2.5-2: Äquivalenz des Bedarfs



(Streckung) oder von fünf Mitarbeitern in drei Tagen (Stauchung) zu bewältigen.

Abschnitt 3.2.4

Eine Streckung oder Stauchung kann nicht beliebig groß gemacht werden, da es vom Aufwand her eine optimale Personalstärke für ein Team gibt.

Der Personalbedarf für einen Vorgang ergibt sich also auf der Basis des geschätzten Gesamtaufwands aus der geplanten Dauer wie folgt:

$$\text{Bedarf} = \frac{\text{Aufwand}}{\text{Dauer}}$$

mit:
Bedarf in Anzahl Mitarbeiter (MA)
Aufwand in Mitarbeiter-Tagen (MT)
oder Mitarbeiter-Monaten (MM)
Dauer in Tagen (T) oder Monaten (M)

Unter Berücksichtigung der Brutto- und Netto-Berechnungen ergeben sich folgende Formeln:
Bedarf in MA (nach Brutto-Rechnung) =

$$= \frac{\text{Aufwand in Brutto-MM}}{\text{Dauer in M}} \times \frac{1}{\text{Produktivanteil}}$$

Bedarf in MA (nach Netto-Rechnung) =

$$= \frac{\text{Aufwand in Netto-MM}}{\text{Dauer in M}}$$

Beispiel

Für die Erstellung eines ersten Produktmodells wird ein Gesamtaufwand von 47 Mitarbeiter-Tagen à 8 Stunden geschätzt (376 Mitarbeiter-Stunden). Für die Erstellung des Produktmodells stehen 11 Arbeitstage zur Verfügung (1 Monat = 20,8 Tage).

Brutto-Rechnung:

$$\text{Aufwand in Brutto-MT} = \frac{376 \text{ M Stunden}}{153 \text{ Stunden/Monat}} = 2{,}4 \text{ Brutto-MM}$$

$$\text{Bedarf an MA} = \frac{2{,}4 \text{ Brutto-MM}}{0{,}53 \text{ M}} \times \frac{1}{0{,}83} = 5{,}5 \text{ MA}$$

Netto-Rechnung:

$$\text{Aufwand in Netto-MT} = \frac{376 \text{ M Stunden}}{127 \text{ Stunden/Monat}} = 2{,}9 \text{ Netto-MM}$$

$$\text{Bedarf an MA} = \frac{2{,}9 \text{ Netto-MM}}{0{,}53 \text{ M}} = 5{,}5 \text{ MA}$$

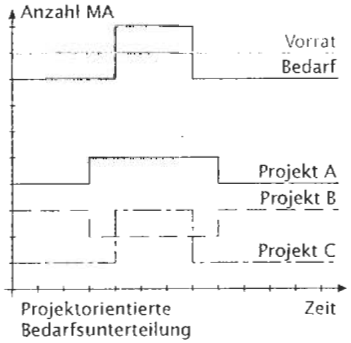
Der ermittelte Bedarf kann dem Vorrat nach projektorientierten, qualifikationsorientierten und organisationsorientierten Gesichtspunkten gegenübergestellt werden (Abb. 2.5-3).

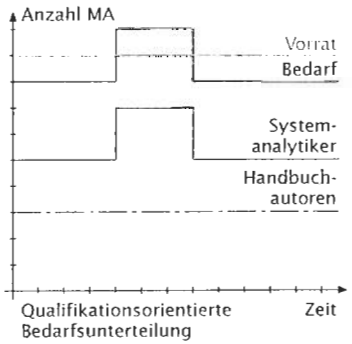
3 Vergleich Bedarf – Vorrat

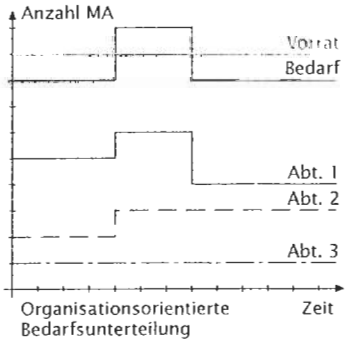
Ziel der Personaleinsatzplanung ist die Optimierung der ermittelten Personalauslastung. Dabei wird versucht, nichtkritische Vorgänge aus Überlastbereichen in Bereiche mit geringer Auslastung zu verlegen. Um dies zu tun, werden Kalender benötigt.

4 Optimierung

Abb. 2.5-3: Kapazitätsauslastungsübersicht nach unterschiedlichen Kriterien







Kalender

Kalender legen die verfügbare Arbeitszeit (Stunden, Wochentage, Termine und Jahre) für einen Vorgang oder eine Ressource fest. Jedes Projekt verfügt über einen primären Kalender, den sogenannten Projektkalender, der im ganzen Projekt verwendet wird.

Für Ressourcen oder Vorgänge sollten eigene Kalender definiert werden, wenn sich die Zeiten vom Projektkalender unterscheiden. Ein Ressourcenkalender kann z.B. die Urlaubszeiten der Mitarbeiter enthalten.

Bedarfsoptimierung

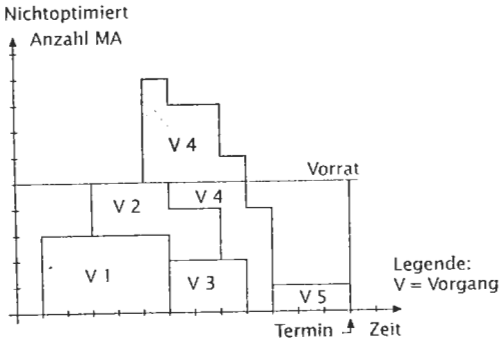
Bei der **termintreuen Bedarfsoptimierung** versucht man, die einzelnen Vorgänge innerhalb ihrer jeweiligen Zeitpuffer so zu verschieben, daß eine möglichst gleichmäßige Auslastung erreicht wird.

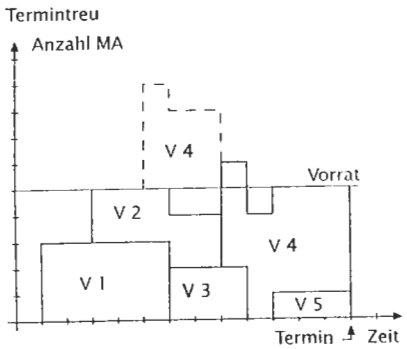
Nivellieren

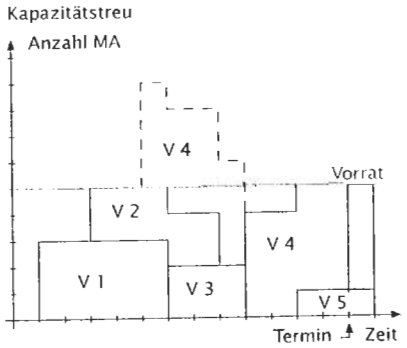
Bei der **kapazitätstreuen Bedarfsoptimierung** erfolgt ein weiteres Nivellieren, z.B. auf die Anzahl der verfügbaren Mitarbeiter. Meist werden dabei die Termine so verändert, daß sich auch der Endtermin verschiebt. Ziel bei der kapazitätstreuen Bedarfsoptimierung ist eine Terminfestlegung, so daß zu keiner Zeit der Bedarf den Vorrat an Ressourcen übersteigt. Gleichzeitig wird eine möglichst kurze Projektdauer angestrebt.

Abb. 2.5-4: Termin- und kapazitätstreue Auslastungsoptimierung

Abb. 2.5-4 zeigt ein Beispiel für die verschiedenen Auslastungsoptimierungen.







Die verfügbaren Personalkapazitäten sind in einer Ressourcenliste aufgeführt (Abb. 2.5-5).

Beispiel

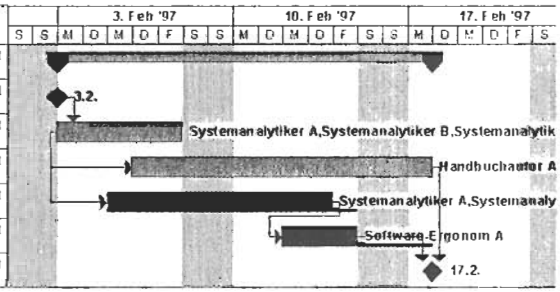
Nr.	Ressourcenname	Kurzel	Gruppe	Max. Einh.	Basiskalender
1	Handbuchautor A	HA	Handbuchautc	1	Standard
2	Handbuchautor B	HB	Handbuchautc	1	Standard
3	Software-Ergonom A	SEA	Ergonom	1	Standard
4	Systemanalytiker A	SA	Analytiker	1	Standard
5	Systemanalytiker B	SB	Analytiker	1	Standard
6	Systemanalytiker C	SC	Analytiker	1	Standard
7	Systemanalytiker D	SD	Analytiker	1	Standard
8	Systemanalytiker E	SE	Analytiker	1	Standard

Abb. 2.5-5: Auszug aus einer Ressourcenliste

Diese Ressourcen wurden den Vorgängen aus dem Netzplan der Abb. 2.4-8 zugeordnet (siehe Abb. 2.5-6).

Abb. 2.5-6: Auszug aus einer Zuordnungstabell

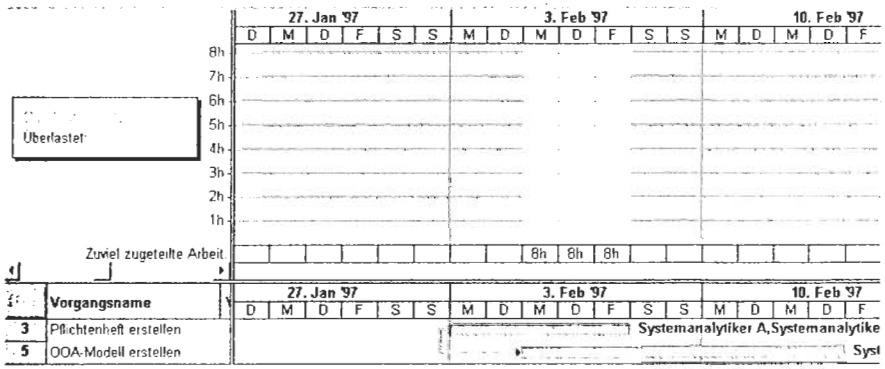
Nr.	Vorgangsname	Dauer
1	Definition	11t
2	Start der Definitionsphase	0t
3	Pflichtenheft erstellen	5t
4	Benutzerhandbuch erstellen	8t
5	OOA-Modell erstellen	7t
6	Oberfläche ableiten	3t
7	Produktmodell fertig	0t



Für das Pflichtenheft wurde ein Aufwand von 15 MT, für das OOA-Modell von 21 MT, für die Oberfläche 3 MT und für das Benutzerhandbuch von 8 MT geschätzt.

Eine Analyse der Ressourcenauslastung zeigt, daß die Systemanalytiker A, B und C vom 3.2 bis zum 9.2. mit 100 Prozent Überlast arbeiten (Abb. 2.5-7). Der Grund liegt darin, daß die Pflichtenheft- und OOA-Erstellung mit zwei Tagen Verzug parallel durchgeführt werden.

Abb. 2.5-7: Ressourcenhistogramm



Um zu einer kapazitätstreuen Auslastung zu gelangen, gibt es zwei Möglichkeiten. Bei der ersten Möglichkeit (Abb. 2.5-8) wird der Start des Vorgangs »OOA-Modell erstellen« um fünf Tage nach hinten verschoben. Dadurch ändern sich die kritischen Vorgänge und der kritische Pfad. Außerdem verschiebt sich der Endtermin um drei Tage nach hinten. Die Systemanalytiker A, B und C sind jetzt zu jeweils 100 Prozent ausgelastet.
Die zweite Möglichkeit besteht darin, für die Überlappungszeit der Vorgänge drei weitere Systemanalytiker einzusetzen, oder ein bis zwei weitere Systemanalytiker, bei gleichzeitig verringerter Überlast der bisherigen Systemanalytiker, hinzuzuziehen.

Einsatzplanung der Betriebsmittel

Entsprechend ihrer »Beständigkeit« unterscheidet man »nicht verzehrbare« Betriebsmittel wie Software-Entwicklungsarbeitsplätze, Transportmittel, Räumlichkeiten und Lagerflächen und »verzehrbare« Betriebsmittel, die verbraucht werden und nach der Nutzung nicht mehr zur Verfügung stehen, wie Datenträger und Büromaterial. Es hängt vom jeweiligen Entwicklungstyp ab, welche Betriebsmittel in einem Projekt relevant sind.

Allerdings sollte eine Betriebsmitteleinsatzplanung nur dann vorgenommen werden, wenn Engpässe bei relevanten Betriebsmitteln eintreten können. Sonst sollte man auf eine entsprechende Einsatzplanung verzichten, da sie eine zusätzliche Arbeitsbelastung mit sich bringt.

Die Methoden für die Betriebsmitteleinsatzplanung entsprechen weitgehend denen für die Personaleinsatzplanung. Man unterscheidet bei der Betriebsmitteleinsatzplanung jedoch drei Vorgehensweisen.

vorrats-eingeschränkt

Bei der **vorratseingeschränkten Einsatzplanung** ist ein festgelegter, nicht vergrößerbarer Vorrat eines Betriebsmittels vorhanden. Dieser Vorrat muß in einer zeitlichen Folge auf mehrere Nutzer möglichst fair aufgeteilt werden. Eine Aufteilung erfolgt am besten über abwechselnde Nutzungsschichten. Die Verteilung erfolgt durch einen Schichtplan. Beispielsweise werden begrenzt verfügbare Spezialrechner auf eine Früh- und eine Spätschicht aufgeteilt.

bedarfsbezogen

Bei einer **bedarfsbezogenen Einsatzplanung** wird zunächst von einem unbegrenzten Vorrat ausgegangen. Zuerst wird festgestellt, welcher Bedarf zu welchen Zeiten besteht. Auf Wunsch kann dann eine termintreue oder kapazitätstreue Durchrechnung der Bedarfsmengen erfolgen, wobei die kapazitätstreue Durchrechnung von dem Einhalten einer obersten Vorratsmenge ausgeht.

frei

Bei der **freien Einsatzplanung** besteht keine Gefahr von Engpässen. Daher trägt jeder Nutzer die gewünschte Belegung in einem Belegungsplan ein.

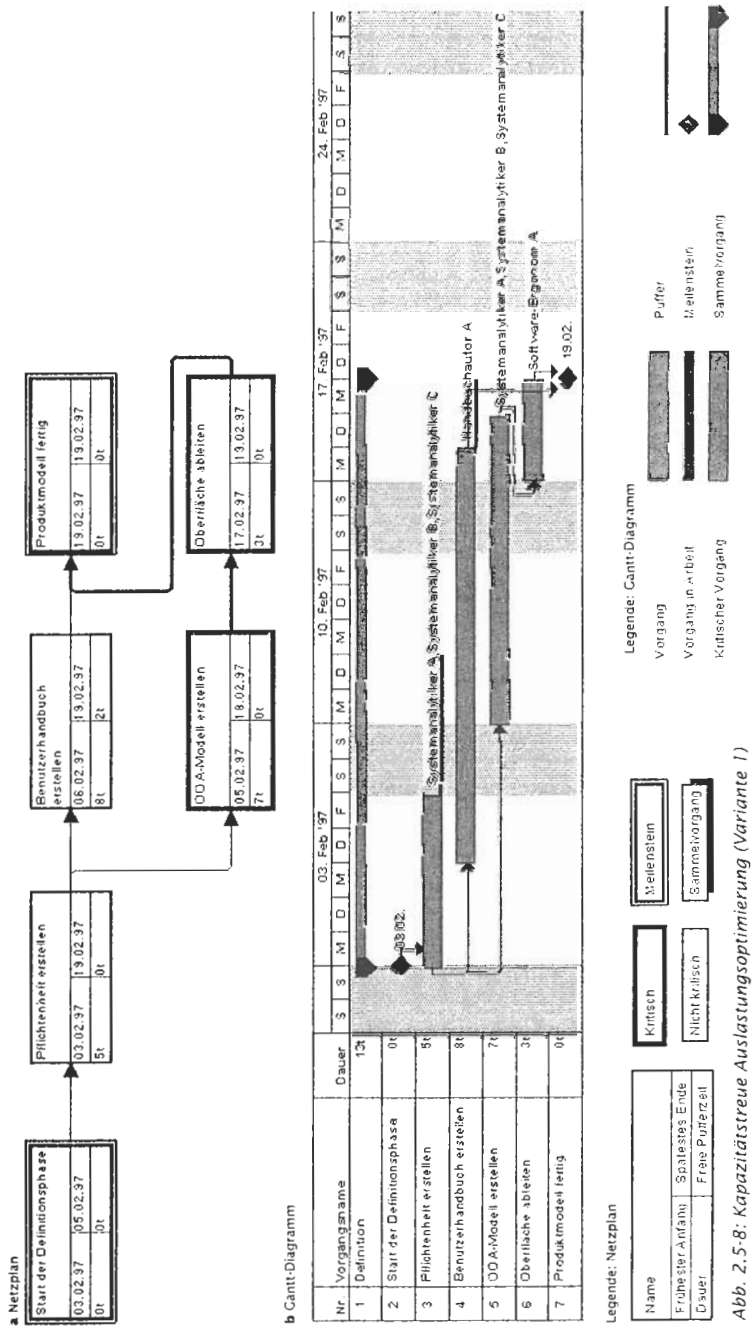


Abb. 2.5-8: Kapazitätstreue Auslastungsoptimierung (Variante 1)

Ressourcenplan

Während der Vorgangsplan ein vorgangsbezogenes Gantt-Diagramm ist, stellt der Ressourcenplan eine ressourcenbezogene Tabelle dar. In der Regel wird ein Ressourcenplan dazu benutzt, die Zuordnung der Vorgänge zu den Mitarbeitern aufzuzeigen.

Beispiel

Abb. 2.5-9 zeigt den mitarbeiterbezogenen Ressourcenplan (Ausschnitt) für den Netzplan nach der kapazitätstreuen Auslastungsoptimierung (siehe Abb. 2.5-8).

Abb. 2.5-9: Personenbezogener Ressourcenplan

	1.2.	2.2.	3.2.	4.2.	5.2.	6.2.	7.2.
Handbuchautor A						8h	8h
Benutzerhandbuch erstellen						8h	8h
Handbuchautor B							
Software-Ergonom A							
Oberfläche ableiten							
Systemanalytiker A			8h	8h	8h	8h	8h
OOA-Modell erstellen							
Pflichtenheft erstellen			8h	8h	8h	8h	8h
Systemanalytiker B			8h	8h	8h	8h	8h
OOA-Modell erstellen							
Pflichtenheft erstellen			8h	8h	8h	8h	8h
Systemanalytiker C			8h	8h	8h	8h	8h
OOA-Modell erstellen							
Pflichtenheft erstellen			8h	8h	8h	8h	8h
Systemanalytiker D							
Systemanalytiker E							

Bedarfsaufsummierung

Liegt ein konsistenter Netzplan vor, dann können die einzelnen Bedarfsmengen der Vorgänge im Rahmen einer Einsatzmittelberechnung zeitgerecht addiert werden.

Bezogen auf die Vorgangsdauer fällt die Menge der Einsatzmittel entweder an

- zu Beginn des Vorgangs (z.B. Materialien, Geräte) oder
- am Ende des Vorgangs (z.B. Rechnungen) oder
- verteilt über die Vorgangsdauer (z.B. Personal).

Außerdem kann noch zwischen dem frühesten und spätesten Beginn bzw. zwischen frühestem und spätestem Ende des Bedarfsanfalls unterschieden werden. Damit ergeben sich sechs Einplanungsmöglichkeiten des Bedarfs eines Einsatzmittels.

Einsatzplanung bei Multiprojekten

Teilen sich mehrere Projekte ein bestimmtes Einsatzmittel (z.B. den Software-Ergonomen A) oder einen beschränkten Vorrat eines bestimmten Einsatzmittels (z.B. fünf OOA-Werkzeuglizenzen), dann ist eine Multiprojekt- bzw. Mehrprojektplanung nötig.

Die Einsatzplanungen der Projekte sind nicht mehr unabhängig voneinander möglich. Es ist eine Planabstimmung der vorhandenen Ressourcen mit Prioritätsvergabe erforderlich.

Beim abgestimmten Einplanen können unterschiedliche Aspekte wichtig sein, z.B.:

- Bestimmte Mitarbeiter sollen zeitparallel in mehreren Projekten mitarbeiten.
- Eine feste Mitarbeiteranzahl steht als Summe für mehrere Projekte zur Verfügung und soll fachgerecht aufgeteilt werden.
- Ein vorgegebenes Budget soll auf die einzelnen Projekte aufgeteilt werden.
- Eine beschränkte Menge eines bestimmten Betriebsmittels soll fair auf mehrere Projekte aufgeteilt werden.

2.6 Kostenplanung

Der letzte wesentliche Abschnitt einer Projektplanung ist die Kostenplanung. Sie stützt sich auf Daten aus der technischen und der kaufmännischen Planung. Die technische Planung hat primär das technische Entwicklungsziel im Auge, während die kaufmännische Planung die Gesamtheit eines Entwicklungsbereiches betrachtet.

Mit einer projektumfassenden Vorkalkulation sollen möglichst in einer Vollkostenrechnung alle direkten und indirekten Kosten für ein Projekt erfaßt werden.

In der Regel werden in die Stundenverrechnungssätze bereits ein Teil oder alle Gemeinkosten eingearbeitet.

Gemeinkosten

Gemeinkosten sind indirekte Kosten, die nicht direkt einem Projekt zugeordnet werden können, z.B. Mietkosten für das Bürogebäude, Löhne und Gehälter für das Verwaltungspersonal usw. Diese Kosten werden entsprechend einem Gemeinkostenschlüssel auf die einzelnen Projekte oder Stundensätze der Projektmitarbeiter umgelegt. Auch künftige Kostensteigerungen durch Inflation und Gehaltserhöhungen sind in der Kostenplanung zu berücksichtigen.

Die Kostenplanung ergibt sich aus der Aufwandsplanung auf der fachlichen Ebene der Projekte. Aus ihr entstehen die für das jeweilige Projekt erforderlichen Kosten, d.h. die geforderten Geldmittel.

In den meisten Projekten sind mit Vorgängen und Ressourcen Kosten und Erlöse verbunden.

Fixe Kosten sind einmalige, mit einem Vorgang zusammenhängende Kosten, z.B. »OOA-Werkzeug kaufen«.

Fixe Erlöse sind einmalige, mit einem Vorgang zusammenhängende Erlöse, z.B. »10% der Vertragssumme bei Fertigstellung der 1. Version des Pflichtenheftes«.

Geplante Kosten und Erlöse fallen am frühesten Start des entsprechenden Vorgangs an.

Ressourcenkosten

Ressourcenkosten sind laufende, mit einer Ressource zusammenhängende Kosten, z.B. der Stundensatz eines Mitarbeiters. Diese Kosten summieren sich über den Zeitraum, den die Ressource für die Arbeit an einem Vorgang aufbringt.

Vorgangskosten und -erlöse sind die Summe aller festen Kosten und festen Erlöse plus die Ressourcenkosten für jeden Vorgang im Projekt.

Beispiel Abb. 2.6-1 zeigt den Stunden- und Überstundensatz pro Ressource (eingerechnet sind die Gemeinkosten).

Nr.	Ressourcenname	Gruppe	Max. Einh.	Höchstwert	Standardsatz	Überstd.-satz
1	Handbuchautor A	Handbuchautoren	1	1	100.00 DM/h	120.00 DM/h
2	Handbuchautor B	Handbuchautoren	1	0	80.00 DM/h	96.00 DM/h
3	Software-Ergonom A	Ergonomen	1	1	150.00 DM/h	180.00 DM/h
4	Systemanalytiker A	Analytiker	1	1	200.00 DM/h	240.00 DM/h
5	Systemanalytiker B	Analytiker	1	1	180.00 DM/h	216.00 DM/h
6	Systemanalytiker C	Analytiker	1	1	180.00 DM/h	216.00 DM/h
7	Systemanalytiker D	Analytiker	1	0	200.00 DM/h	220.00 DM/h
8	Systemanalytiker E	Analytiker	1	0	180.00 DM/h	216.00 DM/h

Abb. 2.6-1: Ressourcentabelle mit Ressourcenkosten

In einer vorgangsbezogenen Kostentabelle (Abb. 2.6-2) können den Vorgängen die Vorgangskosten zugeordnet werden. Es wird angenommen, daß der Vorgang »OOA-Modell« 3.000,- DM fixe Kosten für den Kauf eines OOA-Werkzeuges verursacht. Außerdem werden für die erste Version des Pflichtenheftes 10.000,- DM fixe Erlöse zu Beginn des »OOA-Modells« geplant. Der Vorgang »Oberfläche« erfordert 5.000,- DM fixe Kosten für die Beschaffung eines UIMS *(User Interface Management System)*. Die Gesamtkosten sind die Summe aller fixen Kosten und die Ressourcenkosten.

Abb. 2.6-2: Vorgangsbezogene Kostentabelle

Nr.	Vorgangsname	Feste Kosten	Gesamtkosten
1	**Definition**	**0.00 DM**	**66,760.00 DM**
2	Start der Definitionsph	0.00 DM	0.00 DM
3	Pflichtenheft erstellen	(10,000.00 DM)	12,400.00 DM
4	Benutzerhandbuch	0.00 DM	6,400.00 DM
5	OOA-Modell erstellen	8,000.00 DM	39,360.00 DM
6	Oberfläche ableiten	5,000.00 DM	8,600.00 DM
7	Produktmodell fertig	0.00 DM	0.00 DM

Legende: () Erlöse

cash-flow

Einige Projekt-Planungssysteme erlauben die Ausgabe einer zeitbezogenen Kostentabelle. Sie zeigt den *cash-flow* eines Projektes. *Cash-flow* ist der Kassenzufluß, d.h. der Überschuß der einem Unternehmen nach Abzug aller Kosten verbleibt. Er dient zur Beurteilung der finanziellen Situation eines Unternehmens.

Unter Budgetierung versteht man die zweckgebundene Zuweisung von Etats oder Ressourcen für einen definierten Zeitraum. Budgets (Kostenrahmen) entstehen im Rahmen der Wirtschaftsplanung eines Unternehmens. Sie sind das Resultat der Aufteilung der Mittel des Wirtschaftsplans auf die Teilbereiche des Unternehmens. Das Budget

besteht im allgemeinen aus vorgegebenen Finanzmitteln oder Ressourcen-Etatzahlen für das laufende oder das nächste Geschäftsjahr.

Während die Projektkosten *bottom-up* ermittelt werden, werden die Budgets *top-down* von der Geschäftsleitung festgelegt.

Der Abgleich zwischen beantragten Projektkosten und bereitgestellten Budgets ist Aufgabe des Managements auf den verschiedenen Ebenen.

2.7 Methodik der Projektplanung

Prozeßmodell auswählen

Ausgangspunkt für eine Projektplanung ist die Auswahl eines Prozeß-Modells, an dem sich die Projektdurchführung orientieren soll. Das Prozeß-Modell gibt Vorgaben für Projektphasen, Vorgänge und Abhängigkeiten zwischen Vorgängen.

Projektplan ableiten

Ein Projektplan muß wesentlich detaillierter und konkreter als ein Prozeß-Modell sein. Insbesondere können in Abhängigkeit vom zu entwickelnden Software-Produkt zusätzliche Vorgänge und Meilensteine definiert werden.

Meilensteine festlegen

Beispiel Anhang I A, Abschnitt I 2.18 f

Im Prozeß-Modell ist der Vorgang »OOA-Modell erstellen« aufgeführt. Bei der Projektplanung für das Projekt »Seminarorganisation« werden die Themenbereiche »Kundenverwaltung« und »Seminarverwaltung« identifiziert, die als unterschiedliche, parallel durchführbare Vorgänge geplant werden.

Aufwandsschätzung durchführen

Da für jeden Vorgang eine Vorgangsdauer festgelegt werden muß, ist eine Aufwandsschätzung des Gesamtprojektes erforderlich. Dafür eignen sich Schätzverfahren, die insbesondere von den Anforderungen an das Produkt ausgehen. Eine geeignete Methode dafür ist die *Function Point*-Methode. Zusätzlich sollten in einer empirischen Datenbasis noch Daten über vergangene Projekte vorliegen sowie Informationen, wie sich der Aufwand normalerweise auf einzelne Phasen und Vorgänge eines Prozeßmodells aufteilt.

Kapitel I 1.5

Bedarfsüberlegungen

Ausgehend vom Aufwand pro Vorgang sind erste Überlegungen anzustellen, mit welcher Personalkapazität der Aufwand bewältigt werden soll.

Dauer = Aufwand/Bedarf

Ausgehend von diesen Überlegungen werden die Vorgangsdauern für das Projekt ermittelt.

Netzplan durchrechnen

Danach kann eine erste Netzplandurchrechnung erfolgen. Die kritischen Vorgänge und Pfade werden ermittelt.

Terminbeschleunigung

Es ist zu prüfen, ob eine Terminbeschleunigung durch Planen paralleler Vorgänge oder überlagernder Vorgänge möglich ist. Stehen zwei Vorgänge in einem Projekt nicht miteinander in Beziehung, dann läßt sich Zeit sparen, indem die Arbeit an ihnen gleichzeitig durchgeführt wird.

Außerdem läßt sich Zeit sparen, wenn zwei Vorgänge auf dem kritischen Pfad überlagert werden. Dies ist dann möglich, wenn ein Vorgang vom Start (nicht notwendigerweise vom Ende) des vorhergehenden Vorgangs abhängt. Für solche Vorgänge kann eine Anfangs/Anfangs-Verzögerung angegeben werden.

Da feste, vorgegebene Termine berechnete Termine ersetzen, sollten so wenig Termine wie möglich fest vorgegeben werden. Nur dann kann der bestmögliche Zeitplan berechnet werden.

Risikominimierung

Um das Risiko der Projektplanung zu reduzieren, sollten Zeitreserven für unerwartete Zwischenfälle eingeplant werden. Wenn aufgrund von Erfahrungen bekannt ist, welche Vorgänge unter Umständen mehr Zeit erfordern, dann sollten einige »Polster« eingeplant werden. Bei den risikoreichen Vorgängen eines Projektes sollten Pufferzeiten vorgesehen werden.

›rgangsbezogenes Gantt-Diagramm ausgeben

Die zeitliche Anordnung der Vorgänge kann durch ein vorgangsbezogenes Gantt-Diagramm dargestellt werden.

Ressourcen schät-:en und zuordnen

Nachdem eine erste Terminplanung vorliegt, müssen für jeden Vorgang die benötigten Ressourcen (Personal und Betriebsmittel) geschätzt und zugeordnet werden.

Zunächst ist der Personal- und Betriebsmittelvorrat zu ermitteln. Der Bedarf ist wie folgt zu ermitteln:

Bedarf = Aufwand/Dauer

Vorrat und Bedarf sind insbesondere bezüglich Qualifikation und Zeitverfügbarkeit zu vergleichen.

Separate Kalender anlegen

Wenn die Arbeitszeiten von Mitarbeitern von der üblichen Arbeitszeit für Vorgänge im Projekt abweichen, dann sind für diese Mitarbeiter separate Kalender anzulegen. Ist z.B. ein Teilzeit-Mitarbeiter einem Vollzeitvorgang zugeordnet, dann erfordert die Arbeit an dem Vorgang mehr Zeit.

Ressourcenaus-ıstung überprüfen

Sind die Ressourcen den Vorgängen zugeordnet, dann ist die Arbeitsauslastung für jede Ressource im Projekt zu überprüfen.

Bedarfsoptimierung vornehmen

Sind einige Ressourcen überlastet, d.h. zu vielen Vorgängen zugeordnet, dann muß versucht werden, eine termintreue oder kapazitätstreue Bedarfsoptimierung vorzunehmen.

Kosten zuordnen

Um eine Kostenplanung vornehmen zu können, sind den Ressourcen Kosten zuzuordnen. Bei den einzelnen Vorgängen können außerdem fixe Kosten und fixe Erlöse angegeben werden. Mit diesen Angaben können die Projektkosten dann kumuliert werden.

Anfang-Anfang-Beziehung (AA) Abhängigkeitsart zwischen zwei →Vorgängen (Anfangsfolge). Der Nachfolger beginnt, wenn der Vorgänger beginnt. Bei Angabe eines Zeitabstands entsprechend zeitlich versetzt.

Anfang-Ende-Beziehung (AE) Abhängigkeitsart zwischen zwei →Vorgängen (Sprungfolge). Ein Vorgang kann enden, sobald sein Vorgänger anfängt.

Arbeitsdauer Zeit, die eine →Ressource für die Arbeit an einem →Vorgang aufbringt.

Ende-Anfang-Beziehung (EA) Normale Abhängigkeitsbeziehung zwischen zwei →Vorgängen (Normalfolge). Das Ende des Vorgängers bestimmt den Beginn des Nachfolgers. Zwischen dem Ende des Vorgängers und dem Beginn des abhängigen Nachfolgers kann ein positiver oder negativer Zeitabstand eingefügt werden.

Ende-Ende-Beziehung (EE) Abhängigkeitsart zwischen zwei →Vorgängen (Endfolge). Der Nachfolger kann vor oder nach dem Ende seines Vorgängers beendet sein. Ein positiver oder ein negativer Zeitabstand kann angegeben werden.

Gantt-Diagramm Balkendiagramm, das die Zuordnung von →Vorgängen zu Personen oder die Zuordnung von Personen zu Vorgängen zeigt. Vorgänge bzw. Personen werden entlang einer Zeitachse dargestellt.

Gemeinkosten Indirekte Kosten, die nicht direkt einem Projekt zugeordnet werden können und daher auf alle Projekte umgelegt werden.

Geplante Termine Frühester Anfangstermin und frühester Endtermin für jeden Vorgang. Ausgehend von dem frühesten Anfang des ersten Vorgangs werden die geplanten Termine in einer → Vorwärtsrechnung berechnet.

Gesamtzeitraum Gesamte Kalenderzeit, d.h. Dauer (→Arbeitsdauer, →Vorgangsdauer) plus arbeitsfreie Zeit, zwischen Beginn und Ende der Arbeit an einem Vorgang.

Kapazitätstreue Bedarfsoptimierung Festlegung der Termine, so daß zu keiner Zeit der Bedarf den Vorrat an → Ressourcen übersteigt. Meistens verschiebt sich dadurch auch der Endtermin (→Termintreue Bedarfsoptimierung).

Kritischer Pfad Folge →kritischer Vorgänge, die den kürzesten Zeitraum für die Projektdurchführung darstellt.

Kritischer Vorgang →Vorgang, dessen Verzögerung die spätesten Endtermine abhängiger Vorgänge verschieben würde, oder ein Vorgang, dessen →Pufferzeit kleiner oder gleich einer vorgegebenen Toleranz ist (z.B. Null).

Meilenstein Markiert einen Zeitpunkt, zu dem ein Arbeitsergebnis fertiggestellt sein soll. Meilensteine sollen überprüfbar, kurzfristig und gleichverteilt sein. Oft wird der Projektbeginn ebenfalls durch einen Meilenstein markiert.

MPM-Netzplan Vorgangsknoten-Netzplan (→Netzplan), bei dem Rechtecke die →Vorgänge und Verbindungspfeile die Abhängigkeiten zwischen den Vorgängen symbolisieren.

Netzplan Stellt alle →Vorgänge und → Meilensteine grafisch dar und zeigt die Reihenfolge, in der sie bearbeitet werden müssen (→MPM-Netzplan).

Phase Gruppe zusammenhängender → Vorgänge, die einen globalen Arbeitsabschnitt darstellen. In der Software-Technik ein festgelegter globaler Arbeitsabschnitt innerhalb einer Software-Entwicklung.

Planung Systematisches, zukunftsbezogenes Durchdenken und Festlegen von Zielen sowie der Wege und Mittel zur Erreichung der Ziele.

Projektplan Festlegung von →Vorgängen und Meilensteinen einschließlich ihrer zeitlichen Zusammenhänge (→ Netzplan), Zuordnung von →Ressourcen und →Kosten sowie termintreue oder kapazitätstreue Durchrechnung der Planung.

Prozeß-Architektur Rahmen für die Spezifikation und Interaktion von Prozessen.

Prozeß-Modell Allgemeiner Entwicklungsplan, der das generelle Vorgehen beim Entwickeln eines Software-Produkts festlegt. Die Spezifikation erfolgt im Rahmen einer →Prozeß-Architektur.

Prozeß Menge von Methoden, Verfahren und Werkzeugen, die zur Entwicklung eines Software-Produktes benötigt werden.

Pufferzeit Zeit, um die sich ein → Vorgang verzögern kann, ohne den Endtermin des Projekts zu beeinflussen. Bei geplanten Terminen ist die Pufferzeit die Differenz zwischen dem spätesten und dem frühesten Anfang eines Vorgangs.

Ressource Alle für die Durchführung eines →Vorgangs erforderlichen Einsatzmittel wie Personal oder Betriebsmittel (Maschinen, Materialien).

Rückwärtsrechnung Berechnung der →späten Termine, beginnend beim letzten →Vorgang und endend beim ersten (→Vorwärtsrechnung).

Späte Termine Spätester Anfangstermin und spätester Endtermin für jeden Vorgang. Hat ein Projekt einen festen Endtermin, dann werden die späten Termine durch eine Rückwärtsrechnung berechnet.

Tatsächliche Termine Tatsächlicher Anfangstermin und tatsächliches Ende für jeden Vorgang.

Termintreue Bedarfsoptimierung Verschiebung einzelner →Vorgänge innerhalb ihrer Zeitpuffer, so daß eine möglichst gleichmäßige Auslastung der →Ressourcen erreicht wird (→kapazitätstreue Bedarfsoptimierung).

Vorgang Identifizierbare Aktivität, die innerhalb einer angemessenen Zeitdauer durchgeführt werden kann. Mehrere →Vorgänge werden zu Phasen zusammengefaßt.

Vorgangsdauer Arbeitszeit, die für die Fertigstellung eines →Vorgangs erforderlich ist.

Vorwärtsrechnung Berechnung der frühen Termine, beginnend beim ersten → Vorgang und endend beim letzten (→ Rückwärtsrechnung).

Das Software-Management legt im Rahmen seiner Kompetenzen und unter Berücksichtigung von einzuhaltenden Vorgaben Ziele sowie Wege und Mittel zur Erreichung der Ziele fest. Diese Aufgabe bezeichnet man als Planung. Die grundsätzlichen Vorgehensweisen werden in Form von Prozeß-Modellen beschrieben, wobei eine Prozeß-Architektur den allgemeinen Beschreibungsrahmen von Prozessen und ihren Interdependenzen angibt. Aus Prozeß-Modellen werden konkrete Projektpläne abgeleitet, meistens in Form von Netzplänen. Weit verbreitet sind MPM-Netzpläne. Sie zeigen den zeitlichen Zusammenhang von Vorgängen und Meilensteinen. Mehrere Vorgänge können zu Phasen zusammengefaßt werden.

Zu jedem Vorgang kann die Arbeitsdauer, die Vorgangsdauer und der Gesamtzeitraum angegeben werden. Geplante Termine werden durch eine Vorwärtsrechnung, späte Termine durch eine Rückwärtsrechnung berechnet. Tatsächliche Termine geben den tatsächlichen Anfangs- und Endetermin eines Vorgangs an.

Zwischen zwei Vorgängen kann eine Ende-Anfang-Beziehung (EA), eine Anfang-Anfang-Beziehung (AA), eine Ende-Ende-Beziehung (EE) oder eine Anfang-Ende-Beziehung (AE) bestehen.

Eine Termindurchrechnung des Netzplans ermittelt kritische Vorgänge und kritische Pfade. Kritische Vorgänge besitzen keine Pufferzeit. Aus einem Netzplan kann ein vorgangsbezogenes Gantt-Diagramm abgeleitet werden.

Um Personal und Betriebsmittel planen zu können, werden den Vorgängen Ressourcen zugeordnet. Anschließend kann eine termin- oder kapazitätstreue Bedarfsoptimierung vorgenommen werden. Der zeitliche Personaleinsatz kann durch ein personenbezogenes Gantt-Diagramm dargestellt werden. Die Kostenplanung erfordert die Zuordnung von Kosten zu den Ressourcen. In den Ressourcenkosten werden meist auch die Gemeinkosten mit berücksichtigt.

Abb. 2.7-1 zeigt zusammenfassend nochmals die wichtigsten Planungsaktivitäten.

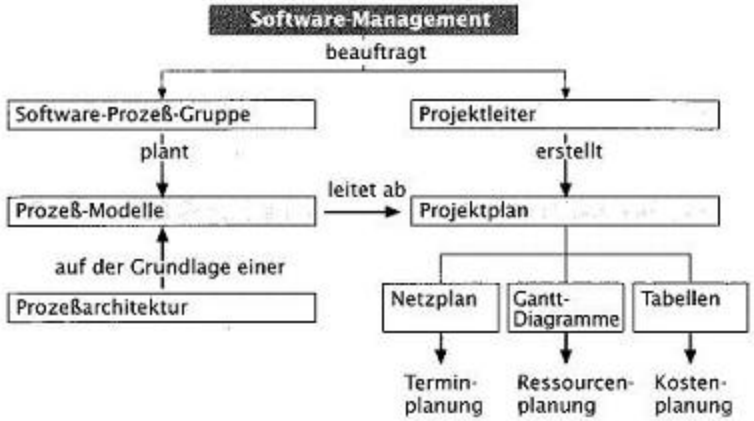


Abb. 2.7-1: Planungsaktivitäten im Überblick

/Burghardt 88/
Burghardt M., *Projektmanagement*, Berlin: Siemens AG, 1988, 511 Seiten
Leitfaden für die Planung, Überwachung und Steuerung von Entwicklungsprojekten. Es wird sowohl auf das Projektmanagement elektrotechnischer Produkte als auch von Software-Produkten eingegangen. Ausführliches, übersichtliches Buch, das viele gute Grafiken enthält.

/Humphrey 89/
Humphrey W.S., *Managing the Software Process*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1989, 494 Seiten
Standardwerk für das Software-Management. Beschreibt die fünf Reifestufen einer Software-Entwicklung und der dazugehörigen Techniken. Dieses Buch war der Ausgangspunkt für das *Capability Maturity Model* (CMM) des SEI *(Software Engineering Institute)*.

Zitierte Literatur

/Koontz, O'Donnell 72/
Koontz H., O'Donnell C., *Principles of Management: An Analysis of Mangerial Functions*, 5th ed., New York: McGraw-Hill Book Company, 1972

/MS Project 94/
Microsoft Project, Benutzerhandbuch, Microsoft Corporation, 1994

/DeMarco, Lister 91/
DeMarco T., Lister T., *Wien wartet auf Dich! Der Faktor Mensch im DV-Management*, München: Carl Hanser Verlag, 1991

1 *Lernziel: Angeben können, welche Anforderungen Meilensteine erfüllen müssen.* — Muß-Aufgabe, *10 Minuten*
- **a** Erklären Sie den Unterschied zwischen Vorgängen und Meilensteinen.
- **b** Wozu dienen Meilensteine und welche Anforderungen müssen sie erfüllen.

2 *Lernziel: Zweck und Darstellungsform von Gantt-Diagrammen angeben können.* — Muß-Aufgabe, *5 Minuten*
Welche Arten von Gantt-Diagrammen unterscheidet man und welche Auswertungen ermöglichen sie?

3 *Lernziel: Die verschiedenen Möglichkeiten der Netzplanstrukturierung aufzählen können.* — Muß-Aufgabe, *10 Minuten*
- **a** Welche Netzplanarten gibt es und durch welche Eigenschaften sind diese unterscheidbar?
- **b** Welche Netzplanart wird in dem Produkt *Microsoft Project* eingesetzt?

Muß-Aufgabe *10 Minuten*

4 *Lernziel: Den Zusammenhang zwischen Prozeß-Architektur, Prozeß-Modell und Projektplan erklären können.*

a Wie sind Prozeß-Architektur, Prozeß-Modell und Projektplan definiert?
b Skizzieren Sie die Zusammenhänge zwischen den drei Begriffen.

Muß-Aufgabe *30 Minuten*

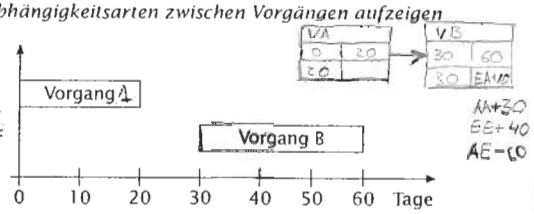
5 *Lernziel: Prozesse und Prozeß-Modelle in der angegebenen ETXM-Notation spezifizieren können.*

Beschreiben Sie den Implementierungsprozeß mit den Prozessen »Codierung«, »Testfallerstellung anhand von Äquivalenzklassen«, »Funktionstest«, »Testfallerstellung anhand nicht durchlaufender Zweige« und »Strukturtest«. Überlegen Sie, welche Prozesse parallelisierbar sind!
Geben Sie eine grafische Darstellung sowie eine tabellarische Darstellung analog der Abb. 2.2-3 an.

Muß-Aufgabe *5 Minuten*

6 *Lernziel: Die verschiedenen Abhängigkeitsarten zwischen Vorgängen aufzeigen können.*

Beschreiben Sie folgende Vorgangskonstellation als Normalfolge, Anfangsfolge, Endfolge und Sprungfolge in einem MPM-Netzplan:



Muß-Aufgabe *60 Minuten*

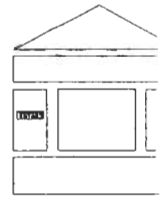
7 *Lernziele: Einen MPM-Netzplan aufstellen, eine Vorwärts- und Rückwärtsrechnung durchführen sowie kritische Pfade ermitteln können. Termin- und kapazitätstreue Bedarfsoptimierungen von Ressourcen vornehmen können. Einfache Kostenplanungen durchführen können.*

Gegeben seien folgende Vorgänge:
/1/ Vorgang 1, Aufwand 3 MT, fester Anfang am 21.10.96
/2/ Vorgang 2, Aufwand 20 MT
/3/ Vorgang 3, Aufwand 15 MT
/4/ Vorgang 4, Aufwand 5 MT, festes Ende am 13.12.96
Zwischen den Vorgängen existieren folgende Abhängigkeiten:
/2/ kann sofort nach Ende von /1/ beginnen
/3/ kann erst 5 Tage nach Ende von /1/ beginnen
/4/ kann erst beginnen, wenn /3/ beendet ist
/4/ kann frühestens 5 Tage vor dem Ende von /2/ beginnen

a Gehen Sie zunächst davon aus, daß jedem Vorgang ein anderer Mitarbeiter zugeordnet ist. Berechnen Sie zu jedem Vorgang die frühen und die späten Termine und geben Sie die Pufferzeiten an. Hat der resultierende Netzplan einen kritischen Pfad?
b Gehen Sie nun davon aus, daß das gesamte Projekt von einem einzigen Mitarbeiter durchgeführt wird. Kann man in diesem Fall eine termin- und kapazitätstreue Bedarfsoptimierung vornehmen? Wenn nicht, welche Möglichkeiten hat man, um das Projekt dennoch durchzuführen?
c Der Mitarbeiter hat einen Überstundensatz von 1000 DM pro Werktag und 1500 DM an Sonn- und Feiertagen. Eine Verzögerung des Endtermins kostet 1000 DM Konventionalstrafe pro Tag.
Welches ist für den Auftragnehmer die kostengünstigste Lösung?

Weitere Aufgaben befinden sich auf der beiliegenden CD-ROM.

3 Organisation



wissen

- Die fünf grundlegenden Koordinationsmechanismen aufzählen können.
- Die fünf Teile einer Organisation einschließlich ihrer Aufgaben nennen können.
- Aufbau- und Ablauforganisation unterscheiden können.
- Gruppierungsalternativen nennen können.
- Kriterien, die die Größe einer Einheit bestimmen, angeben können.

verstehen

- Erklären können, wie die Aufgabenspezialisierung, die Verhaltensformalisierung sowie die Ausbildung und Indoktrination die Gestaltung von Positionen beeinflussen.
- Die Dimensionen der Aufgabenspezialisierung kennen und erläutern können.
- Gruppierungskriterien nennen und beschreiben können.
- Gruppierung nach Märkten und nach Funktionen unterscheiden können.
- Die Kontaktinstrumente Projektleiter und Matrixstrukturen darstellen können.
- Die situativen Faktoren aufzählen und beschreiben können.
- Einsatzgebiete, Charakteristika sowie Vor- und Nachteile von Projektstrukturen und Profibürokratien darstellen können.

anwenden

- Die Methodik zur Organisationsgestaltung auf Fallbeispiele anwenden können.
- Die Auswirkung des Kommunikationsaufwands auf die Produktivität und Zeitdauer berechnen können.
- Ausgehend von vorgegebenen Charakteristika Vorschläge für Organisationsstrukturen erstellen können.

3.1 Einführung

Ziel des Software-Managements ist es, Software-Produkte entwickeln zu lassen. Da aus Termingründen ein Software-Produkt durch mehrere Mitarbeiter erstellt werden muß, sind zwei grundlegende Aufgaben zu erledigen:

Ablauforganisation

1 Der Arbeitsablauf zur Erstellung der Software-Produkte ist festzulegen bzw. aus verschiedenen Modellen auszuwählen, d.h. die **Ablauforganisation** bzw. das **Prozeß-Modell** ist zu definieren. Aus dem Arbeitsablauf ergeben sich die zu erledigenden Einzelaufgaben und ihre Interdependenzen.

Prozeßmodell

Aufbauorganisation

2 Einzelaufgaben müssen im Rahmen der gesamten Entwicklung koordiniert werden. Dazu ist es notwendig, eine geeignete **Aufbauorganisation** festzulegen.

Das Software-Management muß also zwei organisatorische Aufgaben erledigen:

- Für einen mittelfristigen Zeitraum ist eine geeignete Aufbauorganisation zu identifizieren und zu etablieren. Verbunden mit dieser Struktur sind organisatorische Positionen, Verantwortungsbereiche und disziplinarische Vollmachten sowie Qualifikationsprofile für die einzelnen Positionen. Die Aufbauorganisation ist in regelmäßigen Abständen darauf hin zu überprüfen, ob sie für das betreffende Unternehmen noch adäquat ist.
- Eine Aufbauorganisation kann temporäre Elemente enthalten. Diese müssen z.B. beim Start einer neuen Produktentwicklung kurzfristig entschieden und festgelegt werden. Ebenso muß für jede Software-Entwicklung ein jeweils geeignetes Prozeß-Modell ausgewählt werden.

Die folgenden zwei Abschnitte behandeln diese Aufgaben. Im nächsten Abschnitt werden die Grundlagen der Organisationsgestaltung dargestellt. Anschließend werden die für die Software-Entwicklung wichtigsten Prozeß-Modelle mit ihren Vor- und Nachteilen beschrieben.

3.2 Grundlagen der Organisationsgestaltung

Literaturhinweis: /Mintzberg 92/

/Mintzberg 92/ hat in seinem Buch fünf Koordinationsmechanismen, neun Gestaltungsparameter und vier situative Faktoren isoliert, die die Gestaltung einer Organisation beeinflussen. In Abhängigkeit von der Ausprägung dieser Einflußfaktoren ergeben sich fünf Organisationskonfigurationen, zwischen denen es zusätzlich noch Mischformen gibt (Abb. 3.2-1).

Im folgenden werden die Erkenntnisse von Mintzberg in komprimierter Form skizziert und auf die Software-Entwicklung übertragen. Es werden im wesentlichen die Einflußfaktoren und die Organisationskonfigurationen behandelt, die für die Software-Entwicklung entscheidend sind. Einige Einflußfaktoren werden auch erst in dem folgenden Kapitel erläutert.

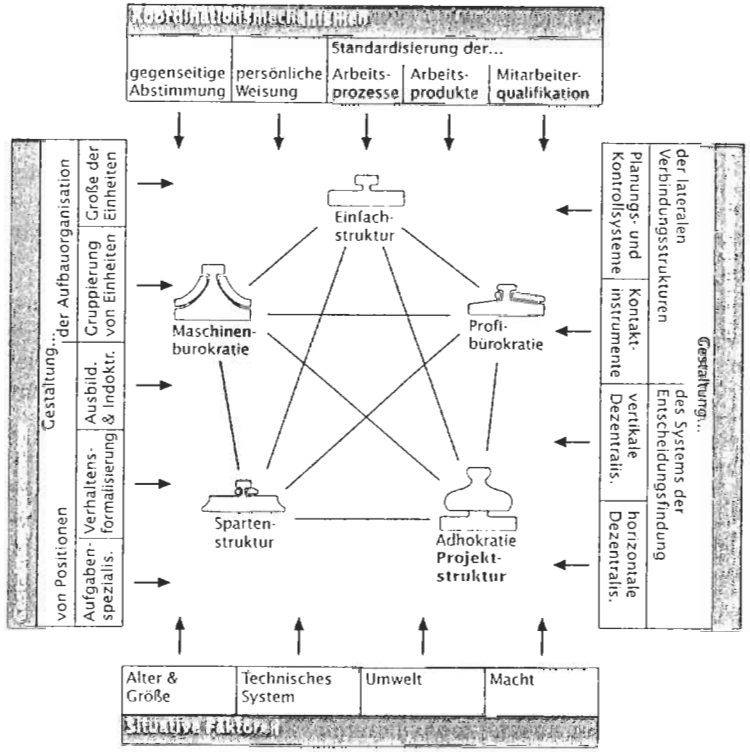


Abb. 3.2-1: Einflußfaktoren der Organisationsgestaltung in Anlehnung an /Mintzberg 92/

Hinweis: Die verwendete Symbolik ist in Abb. 3.2-5 erklärt.

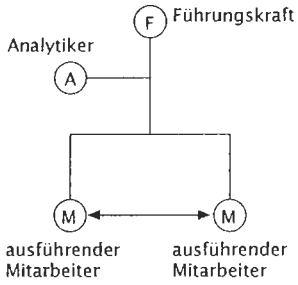
3.2.1 Koordinationsmechanismen

Arbeitsabläufe lassen sich auf fünf verschiedene Arten koordinieren:

- durch gegenseitige Abstimmung,
- durch persönliche Weisung,
- durch Standardisierung der Arbeitsprozesse,
- durch Standardisierung der Arbeitsprodukte und
- durch Standardisierung der bei den Mitarbeitern vorauszusetzenden Qualifikationen.

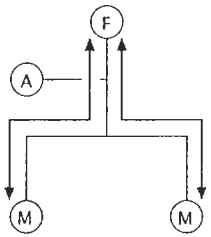
Diese Koordinationsmechanismen umfassen gleichzeitig auch Kontroll- und Kommunikationsaspekte.

Abb. 3.2-2: Gegenseitige Abstimmung /Mintzberg 92, S. 20/



Bei der gegenseitigen Abstimmung erfolgt die Koordinierung der Arbeitsabläufe durch informelle Kommunikation (Abb. 3.2-2). Die Arbeitskontrolle liegt bei den Mitarbeitern selbst.

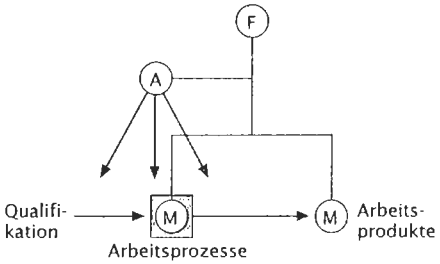
Abb. 3.2-3: Persönliche Weisung /Mintzberg 92, S. 20/



Bei der persönlichen Weisung erteilt ein Mitarbeiter Anweisungen an andere Mitarbeiter und kontrolliert ihre Arbeit (Abb. 3.2-3).

Arbeit läßt sich auch durch Standardisierung koordinieren (Abb. 3.2-4). Arbeitsprozesse, Arbeitsprodukte und Mitarbeiterqualifikationen lassen sich so gestalten, daß zuvor vereinbarte Standards erfüllt werden.

Abb. 3.2-4: Standardisierung /Mintzberg 92, S. 20/



Arbeitsprozeß

Ein Arbeitsprozeß ist standardisiert, wenn die einzelnen Arbeitsgänge festgelegt oder vorausgeplant sind. Eine solche Standardisierung kann z.B. durch einen Planungsanalytiker vorgegeben werden.

Beispiel Hauptkapitel III 5

Für jedes Modul ist ein Funktionstest durchzuführen. Dazu sind aus den Eingabeparametern entsprechend der Äquivalenzklassenanalyse Testfälle abzuleiten. Zu jedem Testfall ist ein Sollergebnis zu ermitteln usw. Anschließend ist ein Zweigüberdeckungstest vorzunehmen.

Arbeitsprodukt

Ein Arbeitsprodukt ist standardisiert, wenn die Ergebnisse der Arbeit, z.B. der Funktionsumfang oder die Qualität, festgelegt sind.

Beispiel Kapitel III 1.2

Von einem Software-Produkt erwartet man, daß die im Pflichtenheft festgelegten Qualitätsziele eingehalten werden, z.B. gute Funktionalität, normale Zuverlässigkeit und sehr gute Benutzbarkeit.

QS-Standards

Qualifikation

Qualifikationen und Kenntnisse sind standardisiert, wenn die Ausbildung, die für die Ausführung bestimmter Arbeiten vorausgesetzt wird, festgelegt ist.

In der Regel hat der Mitarbeiter einer Software-Abteilung eine Ausbildung absolviert. Die Ausbildungsinstitutionen haben Einfluß auf die Arbeitsweise der künftigen Mitarbeiter.

Beispiel

Mitarbeiter, die objektorientiertes Programmieren in ihrer Ausbildung erlernt haben, schreiben objektorientierte Programme. Ein Mitarbeiter der Qualitätssicherung weiß daher, welche Art Programme er zu erwarten hat.

Durch die Standardisierung der Qualifikationen wird also auf indirektem Weg eine Kontrolle und Koordination des Arbeitsablaufs erreicht. In den meisten Organisationen werden diese fünf Koordinierungsmechanismen kombiniert eingesetzt. Ein gewisses Maß an persönlicher Weisung und gegenseitiger Abstimmung ist immer erforderlich.

3.2.2 Die fünf Teile einer Organisation

Um Organisationsabläufe zu kanalisieren und die Wechselbeziehungen der verschiedenen Teile untereinander festzulegen, werden Organisationen strukturiert. Prinzipiell läßt sich eine Organisation in fünf Teile gliedern (Abb. 3.2-5).

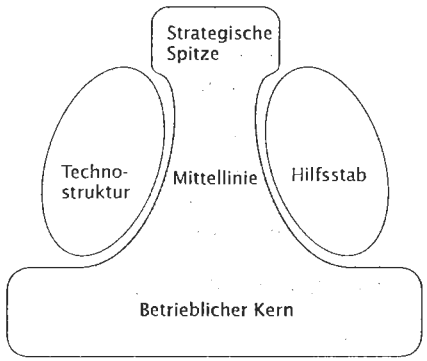


Abb. 3.2-5: Die fünf Teile ei Organisation /Mintzberg 92, S. 28/

betrieblicher K

Im **betrieblichen Kern** einer Organisation befinden sich die ausführenden Mitarbeiter, deren Arbeit direkt mit der Erstellung von Produkten und der Bereitstellung von Dienstleistungen verbunden ist.

strategische Spitze

Die **strategische Spitze** ist dafür verantwortlich, daß die Organisation ihren Auftrag effektiv erfüllt. Außerdem sind die Wünsche derjenigen, die Kontrolle oder sonstigen Einfluß auf die Organisation ausüben (Eigentümer, Gewerkschaften u.a.), zu beachten. Sie hat drei Aufgabenbereiche. Durch persönliche Weisung werden Ressourcen verteilt, Aufträge vergeben, Entscheidungen genehmigt, Konflikte gelöst, Strukturpläne erstellt, Personal eingestellt, Arbeitsleistungen kontrolliert sowie Mitarbeiter motiviert und belohnt. Diesen Aufgabenbereich teilt sich die strategische Spitze mit der Mittellinie.

Der zweite Aufgabenbereich umfaßt die Vertretung der Organisation »nach außen«, d.h. die Aufrechterhaltung ihrer Beziehungen zur Umwelt.

Die strategische Planung der Organisation ist der dritte Aufgabenbereich.

Die Arbeit der strategischen Spitze ist durch einen geringen Wiederholungs- und Standardisierungsanteil, durch große Entscheidungsfreiheiten und relativ lange Entscheidungszyklen charakterisiert. Die Koordinierung erfolgt durch gegenseitige Abstimmung.

Top-Management

Die Führungskräfte der strategischen Spitze werden oft als Top-Management bezeichnet.

Mittellinie

Über eine formale Autoritätskette von Führungskräften (Managern) der **Mittellinie** ist die strategische Spitze mit dem betrieblichen Kern verbunden. Die meisten dieser Ketten verlaufen linear von oben nach unten, d.h. die Auftragserteilung läuft direkt von oben nach unten. Die Führungskräfte der Mittellinie erledigen eine Reihe von Aufgaben im Rahmen der persönlichen Weisung oberhalb und unterhalb der eigenen Position. Abwärts fließen Entscheidungen über die Ressourcen für die eigenen Organisationseinheiten, über die auszuarbeitenden Vorschriften und Pläne sowie über Arbeiten, für deren Durchführung die Einheiten zuständig sind. Rückmeldungen aus den eigenen Einheiten werden oft in komprimierter Form nach oben weitergeleitet, ebenso Veränderungsvorschläge sowie Entscheidungen, die einer Genehmigung bedürfen.

In komplexen Organisationen versucht man, durch Standardisierung die Arbeitsabläufe zu koordinieren. Die Verantwortung für einen Teil dieser Standardisierung liegt bei einer Gruppe von »Analytikern«.

Technostruktur

Sie bilden die **Technostruktur** außerhalb der Hierarchie der Linienführungskräfte.

Analytiker erledigen keine betrieblichen Arbeiten, sondern gestalten, planen und verändern den betrieblichen Arbeitsablauf. Oft bilden sie auch die betroffenen Mitarbeiter aus. Es gibt Analytiker, die für die Innovation sorgen, d.h. für die ständige Anpassung der Organisation an Veränderungen in der Umwelt.

In der Software-Technik bezeichnet man diese Analytiker oft als Methodenberater.

Methodenber

Planungsanalytiker befassen sich mit der Planung und Kontrolle, d.h. der Stabilisierung und Standardisierung der Arbeitsvorgänge. Es lassen sich drei Arten von Planungsanalytikern unterscheiden:

- Arbeitsstudienanalytiker standardisieren Arbeitsprozesse. In der Software-Technik überträgt man diese Aufgabe meist einer Software-Prozeßgruppe.

Prozeßgrupp

- Planungs- und Kontrollanalytiker standardisieren Arbeitsprodukte, z.B. Ingenieure in der Qualitätskontrolle.
- Personalanalytiker einschließlich der Ausbilder und der für die Einstellung von Mitarbeitern zuständigen Führungskräfte standardisieren die Qualifikationen der Mitarbeiter.

Hilfsstab

Der **Hilfsstab** unterstützt mit seinen Diensten die Organisation außerhalb des betrieblichen Arbeitsablaufs. Dazu zählen Einheiten wie Poststelle, Telefonzentrale, Kantine, Rechtsabteilung, Öffentlichkeitsarbeit.

mittleres Management

Als »mittleres Management« oder »mittlere Ebene« werden oft die Führungskräfte bezeichnet, die weder zur strategischen Spitze noch zum betrieblichen Kern gehören.

Linie

Der Begriff »Linie« beschreibt die Hierarchie der Führungskräfte von der strategischen Spitze bis zum betrieblichen Kern. Linienpositionen sind in der Regel mit formalen Entscheidungsbefugnissen ausgestattet.

Stab

Der Begriff »Stab« bezieht sich auf die Technostruktur und den Hilfsstab. Im Gegensatz zu Linienpositionen sind Stabspositionen im allgemeinen nicht mit formalen Entscheidungsbefugnissen versehen. Es handelt sich um Leitungshilfsstellen mit Unterstützungscharakter.

3.2.3 Gestaltung von Positionen

Es lassen sich vier Gruppen von Gestaltungsparametern unterscheiden, die die Arbeitsteilung und Aufgabenkoordinierung beeinflussen:

- Gestaltung von Positionen,
- Gestaltung der Rahmenstruktur,
- Gestaltung der lateralen Verbindungsstrukturen,
- Gestaltung des Systems der Entscheidungsfindung.

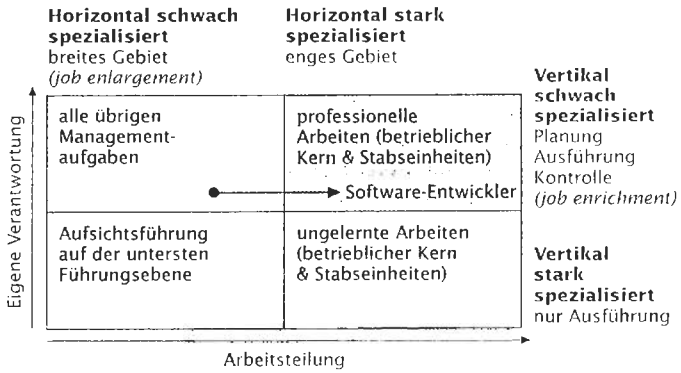
Auf alle vier Gruppen wird im folgenden kurz eingegangen, allerdings schwerpunktmäßig auf diejenigen, die für das Software-Management von besonderer Bedeutung sind.

In diesem Abschnitt wird auf die Gestaltung von Positionen eingegangen. Die Positionsgestaltung hängt von der Aufgabenspezialisierung, der Verhaltensformalisierung bei der Ausführung der Arbeiten sowie der dazu erforderlichen Ausbildung und Indoktrination ab.

Aufgabenspezialisierung

Arbeitsbereiche lassen sich horizontal und vertikal spezialisieren (Abb. 3.2-6).

Abb. 3.2-6: Dimensionen der Aufgabenspezialisierung



Aufgabenspezialisierung

Eine Position mit einem breiten Aufgabengebiet, d.h. einer Vielzahl von Tätigkeiten, ist horizontal gering spezialisiert, während ein enges Aufgabengebiet horizontal stark spezialisiert ist. Eine **Aufgabenspezialisierung** in horizontaler Richtung ist die vorrangige Form der Arbeitsteilung.

Ein Mitarbeiter, der eine Arbeit ausführt, aber nicht plant und kontrolliert, führt eine vertikal spezialisierte Tätigkeit durch. Ist ein Mitarbeiter dagegen für alle Aspekte seiner Arbeit selbst verantwortlich (Planung, Ausführung, Kontrolle), dann ist seine Arbeit vertikal erweitert bzw. gering spezialisiert.

Eine horizontale Aufgabenspezialisierung hat repetitive Arbeitsgänge zur Folge und erleichert damit die Standardisierung. Arbeitsprodukte lassen sich gleichförmiger und effizienter produzieren. Für bestimmte Aufgaben können jeweils geeignete Mitarbeiter eingesetzt werden.

Aufgabenspezialisierung führt zu Kommunikations- und Koordinationsproblemen. Eine hohe horizontale Aufgabenspezialisierung kann auch zu Auslastungsproblemen führen. Die Größe des Betriebes ist dabei ein entscheidender Faktor. Ein hoher Arbeitsanfall begünstigt eine horizontale Spezialisierung.

Arbeitsaufgaben können zu umfassend, aber auch zu eng angelegt sein. Die Beurteilung hängt von der jeweiligen Aufgabe und vom Grad der bisherigen Spezialisierung ab.

professionell

Komplexe Aufgaben, die horizontal, aber nicht vertikal spezialisiert sind, werden als **professionell** bezeichnet. Die Aufgaben der Analytiker in der Technostruktur sind professionelle Aufgaben. Managementaufgaben weisen in der Regel die geringste Aufgabenspezialisierung in der gesamten Organisation auf.

Software-T
Hauptkapi

In der Software-Technik ist die Arbeit oft horizontal und vertikal erweitert. Notwendig ist aber eine verstärkte horizontale Spezialisierung, um die Aufgaben professionell zu erledigen. Der dazu notwendige Weg ist in Abb. 3.2-6 durch einen Pfeil angedeutet.

Formalisierung von Verhaltensweisen

Die Entscheidungsfreiheit der Mitarbeiter kann durch Verhaltensvorschriften eingeschränkt werden.

bürokratis
organisch

Je stärker das Verhalten vorbestimmt oder voraussagbar und somit standardisiert ist, desto **bürokratischer** ist eine Struktur. Im Gegensatz dazu ist eine Struktur **organisch**, wenn es in ihr keine Standardisierung gibt. Je stabiler und repetitiver die Arbeit ist, desto stärker ist sie »programmiert« und desto bürokratischer ist der betreffende Teil der Organisation.

Organisationen mit starker Neigung zu bürokratischen oder organischen Strukturen errichten zur Durchführung spezieller Aufgaben oft unabhängige Arbeitskonstellationen mit entgegengesetzter Struktur. Eine bürokratische Firma bildet beispielsweise ein Team für Neuentwicklungen, das administrativ, finanziell, räumlich und manchmal sogar juristisch von der übrigen Organisation getrennt ist, um innovativ sein zu können.

Ausbildung und Indoktrination

Durch Ausbildung werden Qualifikationen und Kenntnisse für eine Position vermittelt. Durch Indoktrination werden organisatorische Normen erworben. Beide Verfahren bewirken, daß die Mitarbeiter standardisierte Verhaltensmuster »verinnerlichen«.

profession

Eine Arbeit, die komplex und nicht rationalisiert, aber zum Teil formal erfaßt und spezifiziert ist, wird als **professionell** bezeichnet. In einigen Organisationen erfordert ein großer Teil der Arbeit im betrieblichen Kern komplexe Qualifikationen und detaillierte Kenntnisse, z.B. in der Software-Entwicklung. Auch in der Technostruktur wird überwiegend professionell gearbeitet.

Ausbildung ist daher bei allen professionellen Arbeitsbereichen der Gestaltungsparameter, der eine Koordination durch Standardisierung von Qualifikationen ermöglicht.

Professionelle Mitarbeiter müssen lange Ausbildungszeiten durchlaufen, bevor sie ihre Positionen überhaupt einnehmen können. Eine solche Ausbildung erfolgt oft an einer Hochschule außerhalb der Organisation. Damit geht die Verantwortung von der Technostruktur auf Ausbildungsinstitutionen und Berufsverbände über. Durch Indoktrination erreicht eine Organisation die formale Sozialisierung ihrer Mitarbeiter zum eigenen Nutzen, z.B. durch innerbetriebliche Ausbildungsprogramme.

Verhaltensformalisierung und Ausbildung sind austauschbar. Bei professionellen Arbeiten sind die Arbeitsaufgaben komplex. Sie las-

sen sich nicht oder nur schlecht vertikal spezialisieren oder von der Technostruktur der Organisation formalisieren.

Experten — Die Mitarbeiter sind Experten auf wohldefinierten Gebieten und damit horizontal spezialisiert. Die Koordination erfolgt vielfach durch die Standardisierung von Qualifikationen in umfassenden Ausbildungsprogrammen.

Software-Technik — Die Software-Technik-Ausbildung ist insgesamt noch zu wenig standardisiert.

Software-Organisation — Für eine Software-Organisation läßt sich folgendes feststellen:

- Im betrieblichen Kern und in der Technostruktur sind professionelle Arbeiten durchzuführen (horizontal spezialisiert, vertikal erweitert). Den einzelnen Positionen sind daher spezialisierte Arbeitsaufgaben zuzuordnen, die in eigener Verantwortung durchzuführen sind.
- Für jede Position sind die notwendigen Qualifikationen und Kenntnisse zu definieren.
  Die Koordination erfolgt zum Teil durch die Standardisierung von Qualifikationen.
- Sind in einer vorhandenen Organisation die Aufgaben nicht ausreichend horizontal spezialisiert, dann ist eine horizontale Spezialisierung vorzunehmen.

Beispiel — Für ein Software-Haus werden in der technischen Software-Entwicklung folgende Positionen festgelegt:
Systemanalytiker, Konstrukteur (Entwurf und Implementierung) (betrieblicher Kern), Methodenberater, Prozeßplaner (Technostruktur), Software-Ergonom (Hilfsstab).

3.2.4 Gestaltung der Aufbauorganisation

Sind Positionen für eine Organisation beschrieben, dann muß überlegt werden, wie die Positionen zu verschiedenen Einheiten zusammengefaßt werden sollen und wie groß diese Einheiten sein sollen.

Organigramm — Ein **Organigramm** stellt das Ergebnis des Gruppierungsvorgangs als Organisationshierarchie dar. Es vermittelt also ein Bild von der Arbeitsteilung und veranschaulicht den Koordinationsmechanismus der persönlichen Weisung (Abb. 3.2-7).

Methodik — Methodisch bietet sich ein kombiniertes *top-down* und *bottom-up*-Verfahren zur Organisationsgestaltung an:

1 Sämtliche Aufgaben werden zusammengestellt, die unter Beachtung der übergeordneten organisatorischen Voraussetzungen anfallen, z.B. Ziele und Strategien der Organisation.
  Die spezifischen Aufgaben werden aus den generellen Erfordernissen abgeleitet.
2 Die einzelnen Aufgaben werden je nach Grad der gewünschten Spezialisierung unterschiedlichen Positionen zugeordnet. Es wird festgelegt, wieweit diese formalisiert werden sollen und welche Ausbildung und Indoktrination erforderlich ist.
3 Es wird festgelegt, wie viele Positionen in den Einheiten auf unterster Ebene zusammengefaßt werden sollen. Dann wird entschieden, welche und wie viele Einheiten jeweils in den nächsthöheren Einheiten gruppiert werden, bis die vollständige Hierarchie vorliegt (Aufbauorganisation).
4 Die Aufbauorganisation wird ausgebaut, und es werden Entscheidungsbefugnisse zugeordnet.

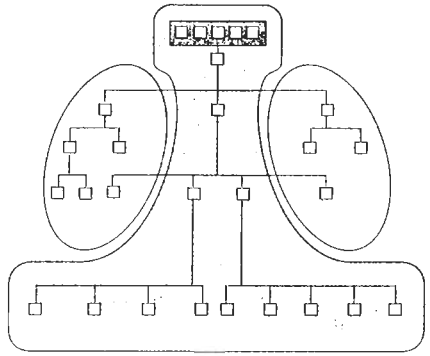


Abb. 3.2-7: Aufbau eines Organigramm

Ein Software-Haus entwickelt Standardsoftware für ausgewählte Bran — Beispiel Fa
chen.

1 Zusammenstellung sämtlicher Aufgaben
- **a** Beobachtung und Analyse des Marktes
- **b** Entwicklung der Branchensoftware
- **c** Vertrieb der Software
- **d** Führung des Unternehmens

2 Zuordnung zu Positionen
- **a** Marketingspezialist: Beobachtung und Analyse des Marktes (betrieblicher Kern)
- **b** Anwendungsspezialist, Systemanalytiker, Konstrukteur: Entwicklung der Branchensoftware (betrieblicher Kern)
- **c** Vertriebsbeauftragter: Vertrieb der Software (betrieblicher Kern)
- **d** Manager: Führung der Firma (Mittellinie und strategische Spitze)
- **e** Methodenberater, Prozeßplaner, Qualitätssicherer (Technostruktur)
- **f** Software-Ergonom, Buchhaltung (Hilfsstab)

Gruppierung in Einheiten
Durch die Gruppierung von Positionen und Einheiten werden Arbeitsbereiche in einer Organisation koordiniert.

Auswirkungen von Gruppierungen

Gruppierungen haben folgende Auswirkungen:

1 Für alle Gruppenmitglieder wird ein gemeinsames Weisungssystem geschaffen.
Jede Einheit wird durch eine Führungskraft geleitet, die für alle Aktivitäten der Einheit verantwortlich ist. Dadurch wird die persönliche Weisung in der Organisationsstruktur verankert.

2 Innerhalb einer Gruppe werden Ressourcen in der Regel gemeinsam genutzt (Budget, Einrichtungen, Geräte).

3 Innerhalb einer Gruppe entwickeln sich gemeinsame Leistungsmaßstäbe. Dies fördert die Bereitschaft der Mitglieder, ihre Aktivitäten zu koordinieren. Außerdem bildet dies die Grundlage für die Standardisierung der Arbeitsprodukte.

4 Gruppierungen fördern die gegenseitige Abstimmung. Die Nutzung gemeinsamer Einrichtungen führt zu konkreten Begegnungen. Dadurch werden informelle Kontakte gefördert und damit auch die Koordination von Aktivitäten durch gegenseitige Abstimmung.

Gruppierungen fördern also eine starke Koordination innerhalb *einer* Einheit. Zwischen *verschiedenen* Einheiten kann es aber zu Koordinationsproblemen kommen.

Gruppierungsalternativen

Folgende Gruppierungsalternativen gibt es:

Qualifikationen

1 Gruppierung nach Kenntnissen und Qualifikationen
Beispielsweise werden alle Diplom-Informatiker in einer Einheit zusammengefaßt.

Arbeitsprozeß, Funktion

2 Gruppierung nach Arbeitsprozeß und -funktion
Einheiten lassen sich nach dem jeweiligen Arbeitsprozeß bzw. nach den Aktivitäten der Mitarbeiter einteilen. Oft ist das technische System ausschlaggebend für die Gruppierung nach Arbeitsprozessen. Arbeitsbereiche können auch nach ihren grundlegenden Funktionen in der Organisation gruppiert werden. Am häufigsten erfolgt eine Gruppierung nach Geschäftsfunktionen: Marketing, Entwicklung, Vertrieb, Buchhaltung, Forschung. Dabei sind einige Gruppen Linieneinheiten und andere Stabseinheiten. Die Gruppierung in Linien- und Stabseinheiten ist ein weiteres Beispiel für Gruppierungen nach Arbeitsfunktionen.

Zeit

3 Gruppierung nach Zeit
Gruppen können auch danach eingeteilt werden, wann eine Arbeit verrichtet wird, z.B. in verschiedenen Schichten.

Produkt

4 Gruppierung nach Arbeitsprodukten
Die Einheiten werden nach den jeweils erstellten Produkten oder Dienstleistungen gebildet.

Beispiel

Das Software-Haus gliedert seine Einheiten nach den drei Branchen, für die es Software entwickelt.

Kunden

5 Gruppierung nach Kunden

Beispiel

Ein Software-Haus, das Software für zwei Automobilkonzerne erstellt, kann seine Einheiten an diesen Kunden ausrichten.

Orte

6 Gruppierung nach Orten

Diese Gruppierungsalternativen lassen sich zu zwei Gruppen zusammenfassen:

- Gruppierung nach Märkten bzw. Objekten (Produkt, Kunde, Ort)
- Gruppierung nach Funktionen bzw. Verrichtungen (Qualifikationen, Arbeitsprozeß, Funktion).

Die erste Gruppierung orientiert sich an den organisatorischen Sachzielen, die zweite an den organisatorischen Maßnahmen.

Gruppierungskriterien

Es gibt vier Kriterien, die helfen, eine geeignete Gruppierung auszuwählen:

Arbeitsablau

1 Interdependenzen im Hinblick auf die Arbeitsabläufe
Betriebliche Aufgaben sollten so gruppiert werden, daß sie den Interdependenzen des natürlichen Arbeitsablaufs entsprechen. Dadurch werden Arbeitsbereiche geschaffen, die arbeitspsychologisch als vollständige Aufgabe anzusehen sind.

Beispiel

Geht man davon aus, daß sich die Software-Entwicklung auf die Tätigkeiten Definition, Entwurf und Implementierung reduzieren läßt, dann sind die Gruppierungen Definition und Konstruktion (mit Entwurf und Implementierung) sinnvoll, die Gruppierungen Definition und Implementierung sowie die Gruppe Entwurf nicht sinnvoll.

Interdepende zwischen Arbeitsablau

Es lassen sich drei Arten von Interdependenzen unterscheiden (Abb. 3.2-8). Werden nur Ressourcen gemeinsam benutzt, dann liegt ein gepoolter Arbeitsablauf vor. Schließt sich eine Aufgabe an die nächste an, dann ist der Arbeitsablauf sequentiell. Verläuft die Arbeit zwischen den verschiedenen Aufgaben hin und her, dann ist der Arbeitsablauf reziprok.

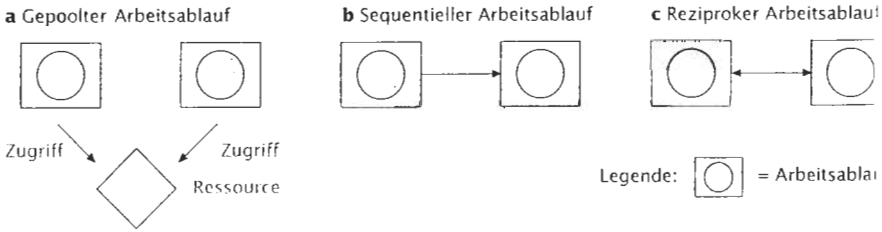


Abb. 3.2-8: Interdepende zwischen Arb abläufen

Gruppen auf der untersten Ebene sollten so gebildet werden, daß die wichtigsten reziproken Interdependenzen berücksichtigt sind. Gruppen auf den höheren Ebenen erfassen die noch verbliebenen sequentiellen Interdependenzen. Die letzte Gruppierung berücksichtigt eventuell verbliebene gepoolte Interdependenzen.

Beispiel Bestehen zwischen Entwurf und Implementierung reziproke Interdependenzen, dann werden diese Aktivitäten zu einer Gruppe Konstruktion zusammengefaßt. Besteht zwischen der Definition und der Konstruktion eine sequentielle Interdependenz, dann werden auf der nächsten Ebene beide Gruppen zu der Einheit Software-Entwicklung zusammengefaßt (Abb. 3.2-9).

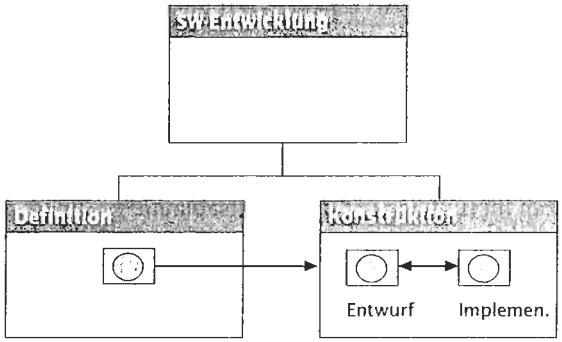


Abb. 3.2-9: Aufbauorganisation unter Berücksichtigung von Arbeitsablauf-Interdependenzen

Arbeitsprozesse **2** Interdependenzen im Hinblick auf Arbeitsprozesse
Es kann sinnvoll sein, Positionen so zu gruppieren, daß Interaktionen bezüglich der Arbeitsprozesse auch auf Kosten der Koordination des Arbeitsablaufs gefördert werden. Werden Spezialisten gleicher Fachrichtung in einer Gruppe zusammengefaßt, dann lernen sie voneinander und können bei ihrer spezialisierten Tätigkeit noch größere Fähigkeiten entwickeln. Nachteilig kann aber sein, daß sie dabei die Kommunikationsfähigkeit zu Personen außerhalb ihrer Fachrichtung verlieren.

Beispiel Vom Arbeitsablauf her sollten Software-Ergonomen der Gruppe Systemanalyse zugeordnet werden. Vom Arbeitsprozeß her kann es aber auch sinnvoll sein, alle Software-Ergonomen in einer Gruppe zusammenzufassen.

Interdependenzen im Zusammenhang mit der Spezialisierung können also eine Gruppierung nach Funktionen nahelegen.

optimale Arbeitsbereiche **3** Interdependenzen im Hinblick auf wirtschaftlich optimale Arbeitsbereiche
Gruppen werden so eingeteilt, daß sie groß genug sind, um effizient arbeiten zu können.

soziale Beziehungen **4** Interdependenzen im Hinblick auf soziale Arbeitsbeziehungen
Mitarbeiter möchten am liebsten so gruppiert werden, daß sie auch »miteinander auskommen«.

Diese vier Kriterien sind bei der Gruppierung nach Funktionen bzw. Märkten zu beachten.

Funktionen Eine **Gruppierung nach Funktionen** berücksichtigt die Interdependenzen im Hinblick auf Arbeitsprozesse und wirtschaftlich optimale Arbeitsbereiche unter Vernachlässigung der Interdependenzen beim Arbeitsablauf. Die Spezialisierung wird gefördert, z.B. durch Einrichtung von Aufstiegsmöglichkeiten für Spezialisten innerhalb des eigenen Bereichs, durch Zuordnung von Vorgesetzten aus demselben Fachgebiet. Durch die Spezialisierung nimmt aber das Interesse an den organisatorischen Gesamtleistungen ab. Die Mitarbeiter sind nicht an den umfassenderen Zielsetzungen der Organisation ausgerichtet. Außerdem läßt sich die Leistung in einer funktionalen Struktur nicht ohne weiteres messen. Der funktionalen Struktur fehlt ein eingebauter Mechanismus zur Koordinierung des Arbeitsablaufs. Die gegenseitige Abstimmung unter den verschiedenen Spezialisten und die persönliche Weisung durch die jeweiligen Führungskräfte werden behindert. Die funktionale Struktur ist unvollständig. Es werden zusätzliche Koordinationsinstrumente benötigt. Funktionale Strukturen neigen zu größerem Bürokratismus. In den höheren Hierarchien werden mehr Führungskräfte benötigt, um die funktional übergreifenden Abläufe und Entscheidungen zu koordinieren.

Markt Eine **Gruppierung nach Märkten** bildet Einheiten, die alle wichtigen sequentiellen und reziproken Interdependenzen umfaßt, so daß nur noch die gepoolten Interdependenzen übrigbleiben. Jede Einheit bezieht ihre Ressourcen und vielleicht auch bestimmte Hilfsdienste aus der gemeinsamen Struktur. Sie führt alle anfallenden Funktionen für eine bestimmte Kategorie von Produkten, Dienstleistungen, Kunden oder Orten aus. Sie tendiert daher dazu, sich mit diesen »Märkten« zu identifizieren. Ihre Leistung ist entsprechend einfach zu beurteilen. Da gegenseitige Abstimmung und persönliche Weisung innerhalb einer Einheit zu leisten sind, ist die Organisation in aller Regel weniger bürokratisch.

Wenn es jedoch überwiegend darum geht, verschiedene Spezialbereiche übergreifend zu koordinieren, dann ist eine Spezialisierung von Arbeitsprozessen nur in geringem Umfang möglich.

Eine marktorientierte Struktur ist im allgemeinen weniger geeignet, spezialisierte oder repetitive Aufgaben effizient durchzuführen. Sie kann jedoch eine größere Anzahl von Aufgaben bewältigen und sich auch leichter auf neue Aufgaben einstellen. Flexibilität ergibt sich dadurch, daß nach Märkten gruppierte Einheiten relativ unabhängig voneinander sind. Neue Einheiten können leicht hinzugefügt, alte weggenommen werden.

Das Selbstbewußtsein der Mitarbeiter kann abnehmen, teilweise dadurch bedingt, daß ihre Arbeit von Bereichsleitern und nicht mehr von fachlich kompetenten Vorgesetzten beurteilt wird. Eine marktorientierte Struktur steht im Widerspruch zu einer umfassenderen Spezialisierung. Dies kann zu einem Rückgang in der Qualität der spezialisierten Arbeitsgänge führen.

Die marktorientierte Struktur verbraucht mehr Ressourcen. Es muß ein doppelter Aufwand an Personal und Ausstattung betrieben werden, da sonst die Vorteile der Spezialisierung verlorengehen. Aufgrund der geringeren funktionalen Spezialisierung können die wirtschaftlichen Vorteile, die mit größenmäßig optimalen Arbeitsbereichen verbunden sind, nicht genutzt werden.

Empfehlungen

Es lassen sich folgende Empfehlungen ableiten:

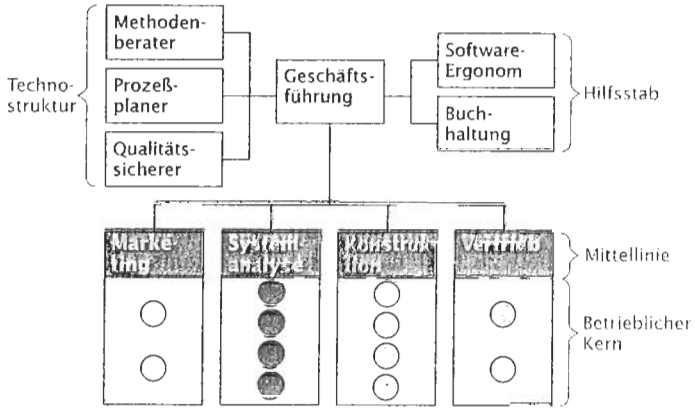
- Eine marktorientierte Struktur empfiehlt sich, wenn die mit dem Arbeitsablauf verbundenen Interdependenzen von entscheidender Bedeutung sind und sich durch Standardisierung nicht gut erfassen lassen.
- Eine funktionsorientierte Struktur empfiehlt sich, wenn die Interdependenzen beim Arbeitsablauf sich durch Standardisierung leicht auffangen lassen und wirtschaftlich optimale Arbeitsbereiche im Vordergrund stehen.

Beispiel 1b

3 Gruppierung von Positionen

Eine funktionsorientierte und eine marktorientierte Struktur für das Software-Haus zeigen die Abb. 3.2-10 und 3.2-11.

Abb. 3.2-10: Gruppierung des Software-Hauses nach Funktionen



Eine Gruppierung nach Funktionen ist die häufigste Gruppierungsform im betrieblichen Kern. In großen Organisationen ist eine Gruppierung nach Märkten auf den höheren Führungslinien häufig anzutreffen.

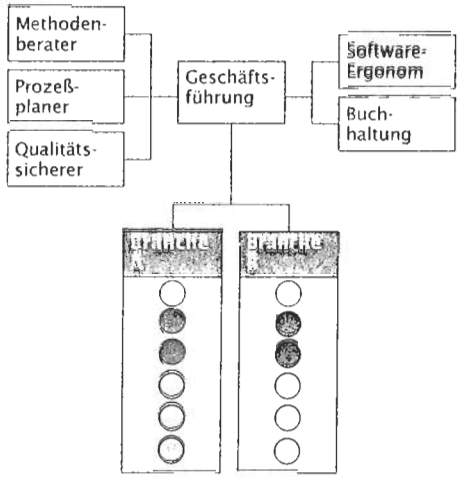


Abb. 3.2-11: Gruppierung d Software-Haus nach Märkten

Größe der Einheiten

groß

Eine Arbeitseinheit kann umso größer sein, je umfassender die Koordinierung durch Standardisierung im Vergleich zur persönlichen Weisung erfolgt. Je besser die Mitarbeiter ausgebildet sind, desto weniger müssen sie angeleitet und kontrolliert werden und desto größer können somit ihre Arbeitseinheiten sein.

klein

Eine Arbeitseinheit muß umso kleiner sein, je stärker die gegenseitige Abstimmung zur Koordination interdependenter Aufgaben erforderlich ist.

Zur Lösung komplexer, miteinander verflochtener Arbeitsaufgaben müssen die Mitarbeiter unmittelbar miteinander sprechen. Daher muß die Arbeitseinheit klein genug sein, um zweckmäßige, häufige und informelle Interaktionen der Mitglieder untereinander zu fördern. Gruppen mit mehr als zehn Mitgliedern begünstigen die Cliquenbildung und zerfallen leicht in kleinere Gruppen.

Diese Aussage steht scheinbar im Widerspruch zu der Aussage, daß Professionalismus (d.h. Standardisierung von Qualifikationen) die Bildung großer Einheiten zuläßt. Der Widerspruch löst sich, wenn man beachtet, daß professionelle Arbeit immer komplex, aber nicht immer interdependent ist.

Es gibt zwei Arten von professioneller Arbeit: Die eigenständigen, nach außen hin abgegrenzten Aufgaben, die ganz andere Strukturformen verlangen als die interdependenten Aufgaben.

Kommunikationsaufwand

Wird ein Produkt von einer Arbeitseinheit erstellt, die ohne gegenseitige Kommunikation auskommt, dann nimmt der Zeitbedarf t für eine Entwicklung mit der Anzahl n der eingesetzten Mitarbeiter entsprechend ab (Abb. 3.2-12):

$$t \sim \frac{1}{n} \qquad 1$$

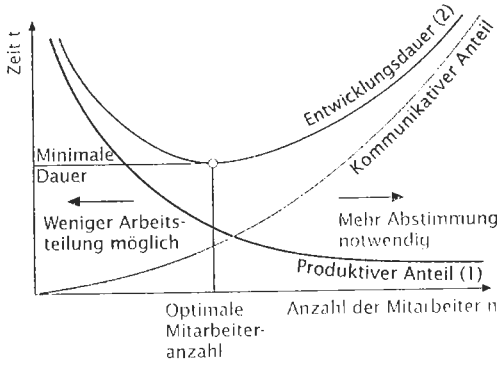


Abb. 3.2-12: Einfluß des Kommunikationsaufwands auf den Gesamtaufwand

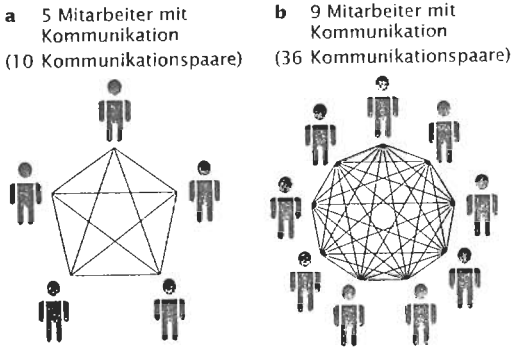
Berücksichtigt man bei der Arbeitseinheit den Kommunikationsaufwand k pro Kommunikationspfad für die gegenseitige Kommunikation der Mitglieder, dann verläuft die Entwicklungszeit proportional der folgenden Formel:

$$t \sim \frac{1}{n} + k\binom{n}{2} = \frac{1}{n} + k\,\frac{n(n-1)}{2} \approx \frac{1}{n} + k\,\frac{n^2}{2} \qquad 2$$

$\binom{n}{2} = n \cdot (n-1)/2$ = Anzahl der Kommunikationspfade

Abb. 3.2-12 zeigt einen grafischen Vergleich der Formeln 1 und 2. Abb. 3.2-13 verdeutlicht die Anzahl der Kommunikationspaare.

Abb. 3.2-13: Anzahl der Kommunikationspaare



Der Kommunikationsaufwand darf also nicht unterschätzt werden. Bei anspruchsvollen Entwicklungen mit starker gegenseitiger Abhängigkeit der Arbeitsaufgaben dürfte die optimale Mitarbeiterzahl verhältnismäßig niedrig sein (etwa 3 bis 7 Mitarbeiter).

Umgekehrt muß bei sehr kleinen Gruppengrößen wegen des Produktivitätsrückgangs aufgrund der geringen Arbeitsteilung mit längeren Entwicklungszeiten gerechnet werden.

Mit dem Kommunikationsaufwand hängt auch das Brooks'sche Gesetz /Brooks 75, S. 25/ zusammen:

»Adding manpower to a late software project makes it later.«

Brooks'sches Gesetz

Ein in Terminverzug geratenes Projekt kann durch zusätzliche – zu spät eingesetzte –Mitarbeiter nicht »termintreuer« gemacht werden. Im Gegenteil, es kann dadurch noch mehr in Verzug gebracht werden. Zusätzliche Mitarbeiter müssen von den vorhandenen geschult und eingearbeitet werden. Es muß eine weitere Aufgabenteilung und eine erneute Abstimmung vorgenommen werden. Dadurch sinkt zunächst die Produktivität der »alten« Mitarbeiter. Die »neuen« Mitarbeiter benötigen mehr Betreuung, erschweren die Kommunikation und sind nicht mit der laufenden Entwicklung vertraut.

Die »optimale« Mitarbeiteranzahl sollte also rechtzeitig zur Verfügung stehen, da eine verspätete Mitarbeiteraufstockung fast immer zu Terminverzögerungen führt.

Steile Strukturen (mit kleinen Einheiten pro Ebene und somit vielen Ebenen) unterstützen das Sicherheitsbedürfnis der Mitarbeiter (eine Führungskraft ist stets in der Nähe), schränken aber auf der anderen Seite Selbständigkeit und Selbstverwirklichung ein. Sie können den vertikalen Informationsfluß nach oben behindern.

Flache Strukturen (mit großen Einheiten und somit wenigen Ebenen) können bei der Entscheidungsfindung ein höheres Maß an Diskussion und Konsultation erfordern. Führungskräfte der unteren Ebenen fühlen sich in flachen Strukturen oft wohler, da sie einen größeren Freiraum haben.

Zusammenfassend läßt sich folgendes sagen:

Größere Einheiten sollten gebildet werden, wenn

1 alle drei Formen der Standardisierung vorliegen,
2 die auszuführenden Aufgaben in einer Einheit ähnlich geartet sind,
3 die Mitarbeiter selbständig sein und sich selbst verwirklichen wollen,
4 die Informationsvermittlung von unten nach oben optimal funktionieren soll.

Kleinere Einheiten sollten gebildet werden, wenn

1 eine straffe persönliche Weisung notwendig ist,
2 komplexe, interdependente Aufgaben eine gegenseitige Abstimmung erfordern,

Prof. Dr. Fred P. Brooks, Jr. *1931 in Durh USA, Wegbere des Software-Managements. *»The Mythical Month«* 1975; Promotion an Harvard-Univ 1956 (angewa Mathematik), langjähriger Mitarbeiter de Firma IBM, he Professor für Informatik an der *University North Carolin*

3 die Führungskraft einer Einheit neben der Kontrollfunktion viele andere Aufgaben wahrzunehmen hat,
4 die Mitglieder der Einheit häufige Beratung beim Vorgesetzten suchen.

Bezug zur Software-Entwicklung

Reflektiert man diese Kriterien auf die Software-Entwicklung, dann treffen für größere Einheiten die Punkte **2** bis **4** zu. Für kleinere Einheiten spricht vor allem **2**.

Um in der Software-Entwicklung zu größeren Einheiten zu kommen, ist es also vor allem erforderlich, durch geeignete Software-Methoden die Interdependenz zwischen Aufgaben drastisch zu reduzieren, so daß eigenständige Aufgaben entstehen.

Bezogen auf die einzelnen Organisationsteile lassen sich bezüglich der Größe der Einheiten folgende Aussagen machen:

- Die größten Einheiten findet man im betrieblichen Kern, da dort die Koordination vorwiegend durch Standardisierung der Arbeitsprozesse erfolgt.
- Nur wenige funktionsorientierte Einheiten lassen sich in einer übergeordneten Einheit zusammenfassen. Das Gegenteil ist bei marktorientierten Einheiten der Fall.
- Die gesamte Führungshierarchie wird zur Spitze hin immer steiler, da auf den oberen Rängen mehr gegenseitige Abstimmung erforderlich ist.
- Bei umfangreicher Technostruktur und vielen Hilfsstabseinheiten sind in der Mittellinie kleine Einheiten zu bilden.
- Bei professionellen Stabseinheiten sind ebenfalls kleine Gruppen zu bilden.

3.2.5 Projektleiter und Matrixstrukturen

lateral: seitlich, seitwärts

Eine Aufbauorganisation muß um laterale – im Gegensatz zu streng vertikalen – Verbindungsstrukturen ergänzt werden.

Hauptkapitel 6

Bei Planungs- und Kontrollsystemen handelt es sich um Verbindungsstrukturen, die eine Standardisierung der Arbeitsprodukte ermöglichen. Auf diese Systeme wird in Hauptkapitel 6 eingegangen.

Kontakt-instrumente

Um eine reibungslose gegenseitige Abstimmung sicherzustellen, werden Kontaktinstrumente eingesetzt: Kontaktpositionen, Arbeitskreise, permanente Ausschüsse, Projektleiter, Matrixstrukturen.

Kontaktposition

Erfordert die Koordinierung der Arbeitsabläufe in zwei Einheiten viel Kommunikation, dann kann eine Kontaktposition eingerichtet werden. Über diese läuft dann die direkte Kommunikation unter Umgehung der vertikalen Kanäle.

Beispiel

In der Fachabteilung, die Aufträge an die interne Software-Abteilung vergibt, wird eine Kontaktposition »Software-Beauftragter« eingerichtet.

Arbeitskreis

Zur Durchführung einer bestimmten Aufgabe kann ein Arbeitskreis gebildet werden, der sich anschließend wieder auflöst.

permanenter Ausschuß

Sind regelmäßige Sitzungen zur Diskussion gemeinsamer Probleme erforderlich, dann ist ein permanenter Ausschuß einzurichten, dem Vertreter aus verschiedenen Einheiten angehören.

Projektleiter

Projektleiter

Ist ein höheres Maß an Koordination durch gegenseitige Abstimmung nötig, dann kann ein **Projektleiter** als Kontaktposition mit formaler Autorität eingerichtet werden.

Eine zunächst unbeteiligte Führungskraft wird der bestehenden Einheitenstruktur vorangestellt und übernimmt einen Teil der Machtbefugnisse, die zuvor von den verschiedenen Einheiten wahrgenommen wurden.

Der Projektleiter erhält für einige Aspekte der Entscheidungsprozesse der betroffenen Einheiten formale Machtbefugnisse, meist bezogen auf fachliche Entscheidungen. Er hat aber niemals die formale Autorität über die Mitarbeiter dieser Einheiten, d.h. keine disziplinarischen Vollmachten.

Der Projektleiter benötigt vielmehr Entscheidungsautorität, Überzeugungskraft und Verhandlungsgeschick, um das Verhalten der zu koordinierenden Mitarbeiter zu steuern.

marktorientierte Projektleiter

Oft werden marktorientierte Projektleiter funktionsorientierten Strukturen vorangestellt, um die Koordinierung von Arbeitsabläufen sicherzustellen.

Beispiel

Abb. 3.2-14 zeigt die funktionsorientierte Strukturierung des Software-Hauses ergänzt um zwei Projektleiterpositionen.

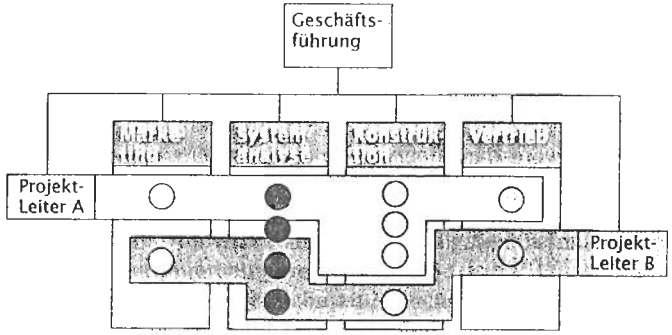


Abb. 3.2-14: Funktionsorientierte Struktur ergänzt um marktorientierte Projektleiter

funktionsorientierte Projektleiter

Es können aber auch funktionsorientierte Projektleiter marktorientierten Strukturen vorangestellt werden, um die Spezialisierung zu fördern.

Beispiel Eine für die Qualität der Software verantwortliche Führungskraft wird einer formal auf Projektbasis organisierten Software-Abteilung vorgeschaltet.

Ein Projektleiter hat das Problem, Einfluß auf Mitarbeiter zu nehmen, über die er keine formale Autorität hat.

Matrixstrukturen

Funktionsorientierte Strukturen führen zu Problemen bei den Arbeitsabläufen, marktorientierte Strukturen verhindern Kontakte unter den Spezialisten.

Matrixstruktur Eine **Matrixstruktur** behält beide grundlegenden Gruppierungsalternativen (marktorientiert und funktionsorientiert) bei. Es entsteht eine doppelte Autoritätsstruktur. Das Prinzip der Alleinverantwortung bzw. der einheitlichen Auftragsvergabe wird aufgegeben.

Es gibt zwei Arten von Matrixstrukturen:

permanent Bei einer **permanenten Matrixstruktur** bleiben die Interdependenzen und damit auch die Einheiten und die zugehörigen Mitarbeiter mehr oder weniger stabil.

alternierend, projektorientiert Bei einer alternierenden, **projektorientierten Matrixstruktur** wechseln die Interdependenzen, die marktorientierten Einheiten und die zugehörigen Mitarbeiter häufig. Diese Struktur wird für Projektarbeiten eingesetzt, bei denen die Arbeitsprodukte häufig wechseln.

Projektteams In solchen Fällen richtet die Organisation eine Reihe von **Projektteams**, d.h. marktorientierte Einheiten auf Zeit, ein, die sich aus Mitgliedern der funktionsorientierten Abteilungen zusammensetzen.

Merkmal der Teams ist, daß ihre Leiter hauptamtliche Führungskräfte (der marktorientierten Einheiten) sind, die formale Autorität (in gemeinsamer Verantwortung mit den Führungskräften der funktionsorientierten Einheiten) über die Mitglieder ihres Teams ausüben. Dies unterscheidet sie von den Leitern der Arbeitskreise und den Projektleitern.

Beispiel 1d Arbeitet das Software-Haus dauernd an zwei Branchenpaketen, dann bietet sich eine permanente Matrixstruktur an.
Werden für die zwei Branchen jeweils neue Produkte zu unterschiedlichen Zeiten entwickelt, dann bietet sich eine projektorientierte Matrixstruktur an (Abb. 3.2-15).

Hinweis: Der Begriff Organisation wird in zweierlei Form verwendet: Organisation als Institution (Man ist eine Organisation) und Organisation als Struktur (Man hat eine Organisation). In der Regel wird die jeweilige Form aus dem Kontext klar.

Die Matrixorganisation scheint die effektivste für die Entwicklung neuer Aktivitäten und für die Koordinierung komplexer und multipler Interdependenzen zu sein. Sie ist aber nicht für Organisationen geeignet, in denen Sicherheit und Stabilität geboten ist. Insbesondere erweist es sich als problematisch, das empfindliche Gleichgewicht zwischen den Machtbefugnissen der verschiedenartigen Führungskräfte aufrechtzuerhalten. Außerdem wird mehr Zeit für Sitzungen

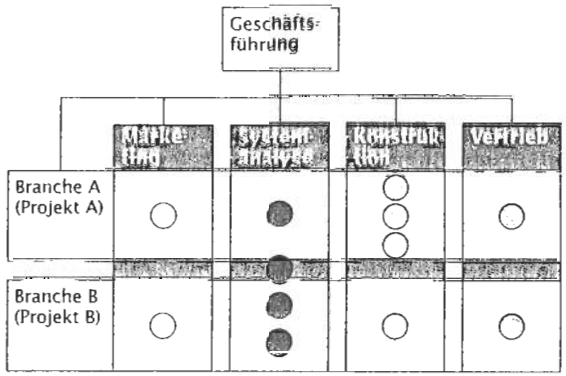


Abb. 3.2-15: Permanente oder projektorientierte Matrixstruktur

benötigt, und es sind mehr Führungskräfte erforderlich. Dadurch steigen die administrativen Kosten.

Abb. 3.2-16 zeigt ein Kontinuum mit einer rein funktionalen Struktur an einem Ende und einer marktorientierten Struktur am anderen Ende.

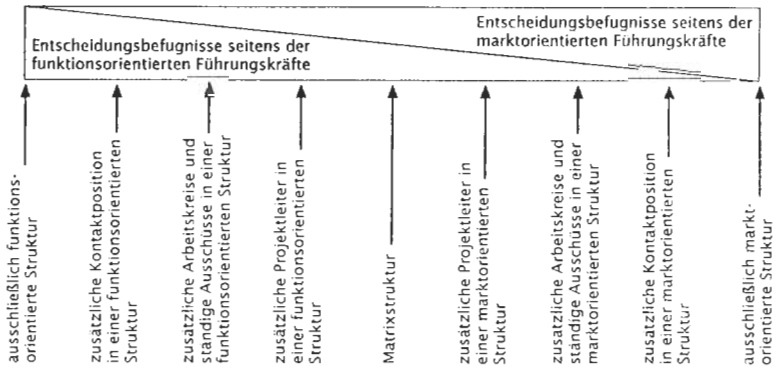


Abb. Ein K von I instr /Min S. 12

Kontaktinstrumente werden am häufigsten in funktionsorientierten Organisationen eingesetzt, um eine zusätzliche Ausrichtung auf die jeweiligen Marktbereiche zu erreichen. Die Einführung von Projektleitern bzw. einer Matrixstruktur führt zu einer starken Erhöhung der Anzahl von Führungskräften. Kontaktinstrumente werden in erster Linie in organischen Strukturen benötigt. Sie kommen im allgemeinen dort zum Einsatz, wo die Arbeit zugleich horizontal spezialisiert, komplex und in hohem Maße interdependent ist.

Es lassen sich zwei Kategorien von professionellen Organisationen unterscheiden:

In der einen Organisation erledigen die Spezialisten ihre Arbeit selbständig als Individuum (z.B. als Berater), während in der anderen Gruppenarbeit geleistet wird. In dieser zweiten Kategorie sind Kontaktinstrumente die wichtigsten Gestaltungsparameter.

Entscheidungsparameter

Neben dem vertikalen und lateralen Aufbau einer Organisation müssen noch die Entscheidungsstrukturen festgelegt werden. Mit Entscheidungsstrukturen verbunden ist die Frage der Zentralisation und Dezentralisation von Entscheidungen. Auf diese Frage wird im Hauptkapitel 5 eingegangen.

Hauptkapitel 5

3.2.6 Situative Faktoren

Eine erfolgreiche Organisation besitzt eine Struktur, die konsistent mit den äußeren Bedingungen ist, welche die Organisation beeinflussen.

Alter

Das Verhalten einer Organisation wird immer formalisierter, je älter sie ist. Mit zunehmendem Alter wiederholen sich die Aktivitäten, bei sonst gleichbleibenden Bedingungen. Dadurch wird die Arbeit leichter formalisierbar und besser planbar.

Größe

Dies gilt auch für größere Organisationen, in denen sich in kürzerer Zeit die Abläufe wiederholen. Je größer eine Organisation wird, desto formalisierter wird auch ihr Verhalten. Ebenso werden die Arbeitsaufgaben spezialisierter. Das führt wiederum zu differenzierteren Einheiten, die durch eine stark strukturierte Administration verwaltet werden müssen. Die größere Spezialisierung führt außerdem zu größeren Organisationseinheiten.

Technisches System

Das verwendete technische System in einer Organisation hat wesentlichen Einfluß auf die Organisationsstruktur. Als technisches System werden alle Verfahren, Methoden und Werkzeuge bezeichnet, die im betrieblichen Kern eingesetzt werden, um anhand der Qualifikationen und Kenntnisse der Mitarbeiter Produkte und Leistungen zu erzeugen.

Es lassen sich drei grundlegende Produktionssysteme unterscheiden:

- Einzelfertigung (vor allem Kundenaufträge),
- Massenfertigung (von vielen Standardprodukten),
- Prozeßfertigung (ununterbrochener oder kontinuierlicher Strom z.B. flüssiger Substanzen).

Einzelfertigung

Bei der Einzelfertigung werden Einzelstücke, Prototypen und große Ausrüstungen in mehreren Produktionsstufen gefertigt. Die Arbeitsprodukte werden ad hoc und ohne Standardisierung erstellt. Die Arbeit kann daher auch nicht standardisiert oder formalisiert werden. Die Betriebe sind organisch strukturiert. Die Führungskräfte der untersten Ebene arbeiten eng mit den Mitarbeitern der betrieblichen Basis – meist in kleinen Arbeitsgruppen – zusammen. Dadurch ergeben sich auf dieser Führungsebene kleine Leitungsspannen, d.h. ein

Vorgesetzter führt nur wenige Mitarbeiter. Die Einzelfertigung ist dem Handwerk vergleichbar, wo sich die Struktur nach den Qualifikationen der Arbeitskräfte im betrieblichen Kern richtet.

Massenfertigung

In Firmen mit Massenfertigung sind die technischen Systeme standardisiert. Dies führt zu einer Verhaltensformalisierung und zu einer Bürokratie. Die im betrieblichen Kern durchzuführenden Arbeiten sind Routinetätigkeiten und weisen einen hohen Grad an Formalisierung auf. Die Technostruktur ist ausgebaut, um die Arbeit zu formalisieren.

Prozeßfertigung

Firmen, die auf die kontinuierliche Produktion von Flüssigsubstanzen oder ähnlichem, z.B. Granulat, Pulver, spezialisiert sind, verfügen über ein weitgehend vollautomatisches technisches System. Beispielsweise läßt sich eine Ölraffinerie mit nur sechs Mitarbeitern betreiben, und auch diese dienen nur zur Überwachung. Alle Regeln, Vorschriften und Standards gelten nur für Maschinen, nicht mehr für Mitarbeiter. Es werden viele technische Spezialisten benötigt, die das technische System gestalten und für einen reibungslosen Ablauf sorgen. Der betriebliche Kern besteht aus hochqualifizierten, indirekten Arbeitskräften wie Wartungsexperten.

Bezug zur Software-Technik

Alle drei Produktionssysteme beschreiben die heutige Software-Entwicklung nicht ausreichend. Die Software-Entwicklung beinhaltet Elemente der verschiedenen Produktionssyteme. Am besten läßt sich die Software-Entwicklung als standardisierte Einzelfertigung mit anspruchsvollen, komplexen, teilweise formalisierten Routinetätigkeiten bezeichnen, die in definierten Produktionsstufen erstellt wird. Teilbereiche sind automatisiert.

Umwelt

Die Faktoren Stabilität, Komplexität und Marktdiversität haben von der Umwelt her den stärksten Einfluß auf eine Organisation.

Stabilität

Ist eine Organisation mit unvorhersagbarer Kundennachfrage, schnell wachsenden Techniken (Fachwissen), häufigen Produktwechseln oder hoher Arbeitskräftefluktuation konfrontiert, also einer dynamischen Umwelt, dann benötigt sie eine organische Struktur. Dynamische Bedingungen haben einen größeren Einfluß auf eine Organisation als statische Bedingungen. Deshalb gilt die Umkehrung nicht.

Komplexität

Die Wissensbasis, d.h. die Techniken, die eine Organisation zur Erledigung ihrer Arbeit benötigt, kann einfach oder komplex sein. Um Software zu erstellen, ist es erforderlich, komplexe Software-Techniken zu beherrschen.

Hauptkapitel 5

Je komplexer die Techniken einer Organisation sind, desto dezentralisierter ist die Struktur, d.h. umso dezentralisierter werden Entscheidungen getroffen.

Die Faktoren Stabilität und Komplexität führen zu vier Organisationstypen (Abb. 3.2-17).

In der Software-Technik ist in der Regel von einer komplexen Umwelt auszugehen.

Abb. 3.2-17: Die Auswirkungen der Stabilität und Komplexität auf eine Organisationsstruktur /Mintzberg 92, S. 196/

	Stabile Umwelt	Dynamische Umwelt
Komplexe Umwelt (komplexe Technologie)	dezentralisiert bürokratisch (Standardisierung von Qualifikationen) Beispiele: Universitäten, Krankenhäuser	dezentralisiert organisch (gegenseitige Abstimmung)
Einfache Umwelt (einfache Technologie)	zentralisiert bürokratisch (Standardisierung von Arbeitsprozessen)	zentralisiert organisch (persönliche Weisung)

komplex, stabil

Eine komplexe, stabile Umwelt führt zu bürokratischen, aber dezentralisierten Strukturen. Die Koordination erfolgt durch die Qualifikationen der Mitarbeiter. Da die organisatorischen Arbeitsabläufe vorhersagbar sind, kann standardisiert werden. Die Machtbefugnisse werden an die Mitarbeiter im betrieblichen Kern abgegeben, die sich auf ihre komplexe, aber routinemäßige Arbeit verstehen. Typische Beispiele hierfür sind Universitäten und Krankenhäuser.

komplex, dynamisch

Eine komplexe, dynamische Umwelt erfordert eine schnelle Reaktion auf nicht vorhersehbare Veränderungen. Entscheidungsbefugnisse werden auf Führungskräfte und Spezialisten verteilt (dezentralisiert), die den erforderlichen Sachverstand haben. Als Koordinationsmechanismus wird im wesentlichen die gegenseitige Abstimmung eingesetzt.

Marktdiversität

Die in Abb. 3.2-17 dargestellten Organisationstypen sind tendenziell funktional strukturiert, wenn die Märkte integriert sind. Hingegen sind sie marktorientiert gegliedert, wenn die Märkte diversifiziert sind, d.h. Produkte, Dienstleistungen oder Kunden vielfältig sind.

Macht

Neben den bisher aufgeführten Bestimmungsfaktoren beeinflußt die Macht der sozialen Normen die Organisationsgestaltung. Je größer die von außen auf eine Organisation ausgeübte Kontrolle ist, desto zentralisierter und formalisierter ist ihre Struktur. Machtbedürfnisse von Organisationsmitgliedern führen zu stärker zentralisierten Strukturen.

3.2.7 Die Projektstruktur

Die Koordinationsmechanismen, die Gestaltungsparameter und die situativen Faktoren führen zu fünf relativ stabilen Konfigurationen von Organisationsstrukturen:

- Einfachstruktur,
- Maschinenbürokratie,
- Spartenstruktur,
- Projektstruktur,
- Profibürokratie.

Bei diesen Konfigurationen sind die Gestaltungsparameter konsistent und die Konfiguration ist kompatibel mit ihren situativen Faktoren. Zwischen den Konfigurationen gibt es außerdem noch Übergänge (Abb. 3.2-1).

Für das Software-Management sind besonders die Projektstruktur und die Profibürokratie relevant, die im folgenden näher betrachtet werden.

Projektstruktur

Eine **Projektstruktur** entwickelt mit Projektteams, die aus professionellen Mitarbeitern verschiedener Disziplinen bestehen, anspruchsvolle Innovationen. Sie läßt sich folgendermaßen charakterisieren:

- Ausgeprägte organische Struktur mit geringer Verhaltensformalisierung.
- Hohe horizontale Aufgabenspezialisierung auf der Grundlage formaler Ausbildung.
- Auf der einen Seite Tendenz zur verwaltungsinternen Gruppierung der Mitarbeiter in funktionale Einheiten, auf der anderen Seite Einsatz der Mitarbeiter in marktorientierten Projektteams (projektorientierte Matrixstruktur).
- Verwendung von Kontaktinstrumenten zur Förderung der gegenseitigen Abstimmung.

Projektmanager

Projektteams werden von einem **Projektmanager** geleitet. Die meisten Projektmanager sind keine klassischen Führungskräfte, die durch persönliche Weisung Aufträge erteilen. Vielmehr widmen sie sich der Kontaktpflege und Verhandlungsführung. Dadurch erfolgt eine laterale Koordination unter den verschiedenen Teams und zwischen diesen Teams und den funktionalen Einheiten. Viele dieser Manager sind selbst Experten, die im Team mitarbeiten. Die Teams sind Arbeitskreise, die am Projektende aufgelöst werden. Die Entscheidungsbefugnisse sind auf Führungskräfte und Nichtführungskräfte aller Hierarchieebenen verteilt.

Administrative und betriebliche Aufgaben gehen ineinander über. Planung und Entwurf einer Arbeit sind kaum von der Durchführung zu trennen. Es werden für beide Aspekte dieselben spezialisierten Qualifikationen benötigt. Mittlere Ebenen sind daher u.U. nicht mehr vom betrieblichen Kern zu unterscheiden.

Die Mitarbeiter für die Projektteams rekrutieren sich aus dem Hilfsstab, den Linienführungskräften und aus dem betrieblichen Kern. Die verschiedenen Organisationsteile sind daher in der Mitte zu einer amorphen Masse verschmolzen.

keine Standardisierung

Da keine Koordination durch Standardisierung erfolgt, besteht kein Bedarf für eine Technostruktur, die regulierende Systeme entwickelt.

Eine Projektstruktur lebt von Projekt zu Projekt. Ohne Projekte überlebt sie nicht. Da jedes Projekt anders ist, weiß man im voraus nicht, welche Problemstellungen als nächste zu lösen sind. Aufgabe der strategischen Spitze ist es, für einen stetigen und ausgeglichene-

nen Auftragseingang zu sorgen. Nur dann ist eine ausgeglichene Arbeitsauslastung sicherzustellen. Die Führungskräfte benötigen viel Zeit für die Überprüfung der Projekte, da innovative Arbeiten schwer zu kontrollieren sind.

dynamisch komplexe Umwelt

Die Projektstruktur ist die einzige, die mit einer dynamischen und komplexen Umwelt zurechtkommt.

Beispiele

Forschungsorientierte Organisationen, forschungsabhängige *High-Tech*-Industrien, Unternehmen mit einem häufigen Produktwechsel und innovative Beratungsfirmen neigen zu einer Projektstruktur. Ihre Arbeit ist komplex, nicht vorhersagbar und oft wettbewerbsintensiv. Häufig ist jeder Kundenauftrag ein neues Projekt. Ein Produkt wird in Einzelfertigung hergestellt.

Alter

Mit zunehmendem Alter tendiert eine Projektstruktur zur Bürokratisierung.

Probleme

Trotz aller Vorteile ist eine Projektstruktur nicht für alle Organisationen geeignet. Sie erfordert, daß der Experte seine individuellen Ziele den Erfordernissen des Teams unterordnet. Konflikt und Aggressivität gehören zur Projektstruktur. Sie müssen durch die Führungskräfte in Produktivität umgesetzt werden.

Eine Projektstruktur ist *nicht* effizient. Für einmalige Projekte ist sie ideal, aber nicht für die Abwicklung gewöhnlicher Vorgänge. Die Ineffizienz entsteht vor allem durch den hohen Kommunikationsanteil und die ungleichmäßige Arbeitsauslastung.

Bezug zur Software-Entwicklung

Obwohl viele Software-Produkte heute im Rahmen einer Projektstruktur entwickelt werden, treffen einige Faktoren, die eine Projektstruktur ausmachen, in den meisten Fällen bei einer Software-Entwicklung *nicht* zu:

- In der Regel werden keine anspruchsvollen Innovationen entwikkelt.
- Die Produkte lassen sich bestimmten Anwendungsklassen zuordnen.
- Viele Software-Häuser entwickeln nicht im Kundenauftrag, sondern für den anonymen Markt.
- Die Umwelt ist nicht immer dynamisch, sondern kann oft als weitgehend stabil angesehen werden.
- Die Arbeit ist komplex, aber meistens vorhersehbar.
- Der Arbeitsprozeß ist zumindest in einem gewissen Rahmen standardisiert.

3.2.8 Die Profibürokratie

Eine **Profibürokratie** erledigt mit professionellen Mitarbeitern im betrieblichen Kern komplexe Arbeiten, um Standardprodukte zu erstellen oder Standarddienstleistungen anzubieten. Sie läßt sich folgendermaßen charakterisieren:

- Die Koordination erfolgt durch Standardisierung der Qualifikationen.
- Der betriebliche Kern ist der wichtigste Organisationsteil. Die Ausführung der Arbeiten erfolgt durch professionelle Mitarbeiter, die entsprechend ausgebildet sind.
- Jeder Mitarbeiter erledigt seine Arbeit weitgehend allein und in eigener Verantwortung.
- Die Arbeitsaufgaben sind horizontal spezialisiert, aber vertikal erweitert.
- Sie besteht in einer komplexen, aber stabilen Umwelt. Die technischen Systeme sind nicht regulativ und nicht kompliziert.
- Die Struktur ist funktions- und marktorientiert (Matrixstruktur).

Beispiele für Profibürokratien sind Universitäten, Krankenhäuser, Schulen, Handwerksbetriebe, Unternehmensberatungsfirmen, Anwaltbüros.

Die Komplexität der Arbeiten hat zur Folge, daß bei ihrer Anwendung immer noch erhebliche Ermessensfreiheit besteht. Zwei Experten setzen ihre Qualifikationen nie in exakt derselben Weise in die Praxis um. Es bleiben immer noch viele Entscheidungen zu treffen.

Die Organisationsstruktur ist im wesentlichen bürokratisch. Die Koordination wird durch bestimmte Ausführungsstandards erzielt. Die Arbeitsprozesse und die Produkte sind so komplex, daß sie sich nicht oder nur eingeschränkt standardisieren lassen. Die Mitarbeiter haben in ihrer Ausbildung eine Reihe von Standardverfahren erlernt. Jeder Kundenauftrag wird in eine Kategorie eingeordnet. Entsprechend der Kategorie werden ein oder mehrere Standardverfahren ausgewählt und damit das Produkt erstellt bzw. die Dienstleistung erbracht.

Eine Profibürokratie ist immer dann sinnvoll,

- wenn der betriebliche Kern überwiegend aus hochqualifizierten, professionellen Mitarbeitern besteht, die schwer zu erlernende, aber gut zu definierende Verfahren anwenden,
- wenn die Umwelt so komplex ist, daß schwierige Verfahren erforderlich sind, die nur in umfassenden, formalen Ausbildungsgängen zu erlernen sind,
- wenn die Umwelt so stabil ist, daß die von den Mitarbeitern zu erwartenden Qualifikationen gut zu definieren und folglich auch zu standardisieren sind.

Die Mitarbeiter arbeiten weitgehend unabhängig voneinander. Es gibt keine in hohem Maße regulativen, komplizierten oder gar automatisierten technischen Systeme. Die Mitarbeiter betreuen Kunden unmittelbar und persönlich.

Die Wissensbasis der Organisation ist kompliziert, aber das technische System, d.h. die zur Anwendung dieser Wissensbasis eingesetzten Instrumente, ist unkompliziert.

Die Profibürokratie ist gut geeignet für die Produktion ihrer Standardprodukte, aber schlecht geeignet, sich auf die Produktion neuer Produkte umzustellen.

Bezug zur Software-Entwicklung

Eine ganze Reihe von Charakteristika beschreiben die heutige Software-Situation korrekt:

- Erhebliche Ermessensfreiheit bei der Ausführung der Arbeiten.
- Nur ansatzweise Standardisierung der Arbeitsprozesse.
- Noch keine Standardisierung der Arbeitsprodukte.
- Standardverfahren zur Lösung wohldefinierter Probleme.
- Kategorisierung der Probleme und Auswahl von Standardverfahren (kommerzielle, technische, Echtzeit-Anwendungen).

Folgende Charakteristika treffen *nicht* zu:

- Die Ausbildung ist noch nicht standardisiert. Daher reicht die Standardisierung der Qualifikationen zur Koordinierung nicht aus.
- Mitarbeiter können nicht allein und autonom arbeiten, um das Produkt oder die Dienstleistung zu erbringen. Gegenseitige Abstimmung ist nötig.
- Es wird ein kompliziertes, z.T. automatisiertes technisches System eingesetzt (CASE-Umgebungen).

3.2.9 Mischstrukturen

Weder die Projektstruktur noch die Profibürokratie berücksichtigen vollständig die Charakteristika einer Software-Entwicklung.

In Tab. 3.2-1 sind die verschiedenen Charakteristika gegenübergestellt. Wie man sieht, sind viele Charakteristika sowohl einer Software-Entwicklung und einer Projektstruktur als auch einer Software-Entwicklung und einer Profibürokratie identisch.

Auch eine Projektstruktur und eine Profibürokratie stimmen in vielen Charakteristiken überein. Zwischen einer Projektstruktur und einer Profibürokratie gibt es Übergänge und Mischstrukturen. Eine Projektstruktur wandelt sich oft in eine Profibürokratie, die sich darauf konzentriert, erfolgreiche Projekte zu wiederholen und dadurch zu einem Standardrepertoire gelangt. Manchmal tendiert sie sogar zur Maschinenbürokratie, um ein einziges Projekt oder eine spezielle Erfindung zu verwalten. Eine Profibürokratie kann Organisationselemente der Projektstruktur übernehmen, um Dynamik und Experimentierfreude zu erhöhen.

Mischt man eine Projektstruktur und eine Profibürokratie, dann deckt man bereits die meisten Charakteristika ab. Zusätzlich ist bei einer Software-Entwicklung aber noch folgendes zu berücksichtigen:

Kapitel 3.3

- Die Arbeitsprozesse werden zumindest teilweise standardisiert.
- Es gibt eine Technostruktur, die für die Innovation, den Arbeitsprozeß und die Qualität zuständig ist.
- Das technische System ist teilweise regulativ, kompliziert und teilweise automatisiert (CASE-Umgebungen).

Charakteristika	**Software-Entwicklung**	Projektstruktur	Profibürokratie
Vorrangiger Koordinations-mechanismus	gegenseitige Abstimmung & Standardisierung der – Qualifikationen und – Arbeitsprozesse	gegenseitige Abstimmung	Standardisierung der Qualifikationen
Wichtigster Organisationsteil	betrieblicher Kern und Hilfsstab sowie Technostruktur	betrieblicher Kern und Hilfsstab	betrieblicher Kern
Gestaltungsparameter:			
Aufgabenspezialisierung	horizontale Spezialisierung, vertikale Erweiterung		
Ausbildung		viel Ausbildung	
Verhaltens-formalisierung	beschränkte Formalisierung organisch & bürokratisch	kaum Formalisierung organisch	kaum Formalisierung bürokratisch
Gruppierung	funktional und marktorientiert		
Größe der Einheiten	eher klein	überall klein	groß unten, sonst klein
Kontaktinstrumente		überall & viel	in der Administration
Funktionen:			
Strategische Spitze		externe Kontakte, Konfliktlösung, gleichmäßige Arbeits-auslastung, Projektüberprüfung	externe Kontakte, Konfliktlösung
Betrieblicher Kern	qualifizierte, grob standardisierte Arbeit im Team mit innovativen, repetitiven und routinehaften Aspekten	qualifizierte, innovative Arbeit, multidisziplinäre Arbeit im Team	qualifizierte, standardisierte Arbeit, viel individuelle Autonomie, weitgehend Einzelarbeit
Mittellinie		umfassend, Trennung vom Stab verwirklicht; Beteiligung an Projektarbeit	kontrolliert durch professionelle Mitarbeiter, viel gegenseitige Abstimmung
Technostruktur	Methodenberater, Prozeßplaner, Qualitätsplaner	klein und mitten in der Projektarbeit verwischt	kaum
Hilfsstab	z.B. Software-Ergonom	ausgebaut, aber in Projektarbeit verwischt	ausgebaut zur Unterstützung der professionellen Mitarbeiter

Tab. 3.2-1a: Charakteristika verschiedener Strukturen im Vergleich
(Die Charakteristika der Software-Entwicklung sind blau hervorgehoben.)

Charakteristika	**Software-Entwicklung**	Projektstruktur	Profibürokratie
Situative Faktoren:			
Technisches System	regulativ, kompliziert, teilweise automatisiert (standardisierte Einzelfertigung)	nicht regulativ, unkompliziert (individuelle Einzelfertigung)	(standardisierte Einzelfertigung)
Umwelt	komplex und semi-stabil bzw. semi-dynamisch	komplex und dynamisch	komplex und stabil
Macht	Kontrolle durch Technostruktur, Experten und externe Kontrolle	Kontrolle durch Experten	Kontrolle durch professionellen betrieblichen Kern
Beispiele		forschungsorientierte Organisationen, forschungsabhängige *High-Tech*-Industrien, Einzelfertigungsbetrieb	Universitäten, Krankenhäuser, Schulsysteme, Handwerksbetriebe, Beratungsunternehmen, Anwaltsbüros

Tab. 3.2-1b: Charakteristika verschiedener Strukturen im Vergleich
(Die Charakteristika der Software-Entwicklung sind blau hervorgehoben.)

Eine »optimale« Organisationsstruktur für eine Software-Entwicklung kann daher folgendermaßen aussehen:

- horizontal spezialisierte, vertikal erweiterte Aufgaben,
- definierte Prozeßmodelle mit Zuordnung der durchzuführenden Aufgaben,
- Computerunterstützung der Prozeß-Modelle,
- projektorientierte Matrixstruktur mit relativ kleinen Einheiten,
- ausgebaute Technostruktur sowie Hilfsstab als Kompetenzzentrum für spezielle Aufgaben.

Eine effektive Strukturierung liegt dann vor, wenn es gelingt, die Gestaltungsparameter und situativen Faktoren konsistent zu kombinieren.

3.2.10 Kooperation Fachabteilung – Systemanalyse

Besitzt ein Unternehmen eine eigene Software-Abteilung, dann muß bei der Gestaltung der Organisation auf eine klare Aufgabenteilung und Aufgabenabgrenzung zwischen den Fachabteilungen und der Systemanalyse geachtet werden. Die Praxis zeigt, daß oft eine »Verwischung« der Verantwortlichkeiten vorliegt. Oft übernimmt die Systemanalyse Aufgaben der Fachabteilung, z.B. Festlegung von Nummernkreisen, Voreinstellungen usw. Umgekehrt erstellt in einigen Fällen die Fachabteilung sogar ein formales Analysemodell. Dazu werden teilweise Mitarbeiter aus der Software-Abteilung in die Fachabteilung »abgeworben«.

Um zu einer reibungslosen und erfolgreichen Zusammenarbeit zu gelangen, sollte sich jede Abteilung auf ihre Stärken konzentrieren. Die Fachabteilung muß folgende Aufgaben erledigen:

Aufgaben Fachabteilung

- Aufstellung der fachlichen Anforderungen an ein zu entwickelndes Software-Produkt.
- Festlegung des Funktions-, Daten-, Leistungs- und Qualitätsumfangs des Produkts.
- Angabe, welche Arbeitsabläufe mit dem Produkt ausgeführt werden.
- Festlegung der Standardvoreinstellungen.
- Aufstellung der Wünsche und Anforderungen an die Benutzungsoberfläche.
- Vorgabe von Prioritäten.
- Erstellung des Benutzerhandbuches in Zusammenarbeit mit der Systemanalyse.
- Überprüfung der von der Systemanalyse erstellten Modelle und Benutzungsoberflächen des Produkts.

Die Systemanalyse ist für folgende Aufgaben zuständig:

Aufgaben der Systemanalyse

- Erstellung einer Produkt-Definition (Pflichtenheft, Produktmodell, Konzept Benutzungsoberfläche).
- Verständliche, übersichtliche Präsentation des Produkt-Modells.
- Bereitstellung von Prototypen zum Ausprobieren der Konzepte.
- Darstellung der Auswirkungen der Anforderungen auf die Benutzungsoberfläche.
- In kurzfristigen Abständen (ca. halbjährlich) liefern einsatzfähiger Teilsysteme.

Da ein Systemanalytiker Profi auf seinem Gebiet ist und tagtäglich Produkt-Modelle erstellt, sollte er in der Regel auch die Projektleitung einer Software-Entwicklung übernehmen. Er weiß besser als ein Mitarbeiter aus der Fachabteilung, welche Fragen gestellt werden müssen, um ein Produkt-Modell zu erhalten. Daher kann er zielgerichteter vorgehen. Der oder die Mitarbeiter der Fachabteilung, die im Projektteam mitarbeiten, tragen natürlich die fachliche Verantwortung für ihre Anforderungen.

Damit die Mitarbeiter der Fachabteilung ihre Aufgaben im Rahmen einer Systemanalyse erfüllen können, sollten sie in den Konzepten, die die Systemanalyse für die Modellierung der Anforderungen verwendet, ausgebildet werden. Jedoch sollte sich diese Ausbildung nur auf das Lesen und Verstehen der Modelle beziehen, nicht auf die Erstellung der Modelle. Der Ausbildungsaufwand steht dafür im Verhältnis 20:80 (Lesen und Verstehen : Erstellen). Oft ist es auch angebracht, nur ein oder zwei Mitarbeiter der Fachabteilung entsprechend zu qualifizieren und sie als Software-Koordinatoren innerhalb der Fachabteilung heranzuziehen.

Ablauforganisation Raum-zeitliche Strukturierung von aufgabenbezogenen Arbeitsvorgängen; Abstimmung der Aktivitäten beinhaltet die Festlegung von Inhalten, Reihenfolge, Zeitdauer und kalendarischen Zeitpunkten der Teilaufgaben. Die räumliche Komponente spezifiziert die Teilaufgaben hinsichtlich Arbeitsbereich, -weg und -ort (→Aufbauorganisation).

Aufbauorganisation Bildung und Koordination aufgabenteiliger funktionsfähiger Organisationseinheiten. (→Ablauforganisation); bezieht sich auf die Aufgaben-, die Rollen-, die hierarchische und die kommunikative Dimension der Organisationsstruktur.

Aufgabenspezialisierung Arbeitsteilung erreicht man durch eine Spezialisierung in horizontaler Richtung, d.h. das Arbeitsgebiet wird eingeengt. Eine vertikale Spezialisierung reduziert die Arbeit auf ihre Durchführung, während eine vertikale Erweiterung auch die Planung und Kontrolle umfaßt (→professionelle Arbeit).

betrieblicher Kern Teil einer Organisation, in der die ausführenden Mitarbeiter Produkte und Dienstleistungen erstellen.

bürokratische Struktur Verhalten der Mitarbeiter durch Vorschriften und Regeln stark festgelegt (standardisiert). (→organische Struktur)

Gruppierung nach Funktionen Zusammenfassung von organisatorischen Einheiten nach den Kriterien Funktion, Arbeitsprozeß oder Qualifikation (→ Gruppierung nach Märkten).

Gruppierung nach Märkten Zusammenfassung von organisatorischen Einheiten nach den Kriterien Produkt, Kunde oder Ort (→Gruppierung nach Funktionen).

Hilfsstab Teil einer Organisation, in der Dienste verrichtet werden, die die Organisation außerhalb des betrieblichen Arbeitsablaufs unterstützen.

Matrixstruktur Kombiniert die →Gruppierung nach Funktionen und die → Gruppierung nach Märkten. Das Prinzip der Alleinverantwortung wird aufgegeben.

Mittellinie Teil einer Organisation, in der Führungskräfte (Manager) durch persönliche Weisung für Koordination in großen Organisationen sorgen. Die Mittellinie verbindet die →strategische Spitze mit dem →betrieblichen Kern.

Organigramm Gibt in grafischer Form an, welche Positionen es in der Organisation gibt, wie diese in verschiedenen Einheiten gruppiert sind und wie die formale Autoritätshierarchie angeordnet ist.

organische Struktur Hohe Entscheidungsfreiheit der Mitarbeiter, keine oder geringe Verhaltenseinschränkungen durch Vorschriften oder Regeln (→bürokratische Struktur).

permanente Matrixstruktur Dauerhafte Einrichtung einer →Matrixstruktur.

professionelle Arbeit Komplexe Aufgaben, die horizontal spezialisiert, vertikal erweitert (→Aufgabenspezialisierung), nicht rationalisiert, aber zum Teil formal erfaßt und spezifiziert sind. Professionelle Arbeit kann eigenständig und nach außen abgegrenzt oder interdependent sein.

Profibürokratie Professionelle Mitarbeiter arbeiten weitgehend allein, um Standardprodukte zu erstellen oder Standarddienstleistungen anzubieten. Zur Erstellung wird auf Verfahren zurückgegriffen, die aus einem Repertoire von Standardverfahren ausgewählt werden.

Projektleiter Führungskraft, die einer bestehenden Einheitenstruktur vorangestellt wird, um die gegenseitige Abstimmung sicherzustellen. Er besitzt formale Autorität für bestimmte Entscheidungen, aber keine disziplinarischen Vollmachten gegenüber den Mitarbeitern der betreffenden Einheiten.

Projektmanager Leitet ein →Projektteam; besitzt formale Autorität (zusammen mit den Führungskräften der funktionsorientierten Einheiten) über die Mitglieder des Teams (→Projektstruktur).

projektorientierte Matrixstruktur Jeweils neu formierte →Matrixstruktur zur Durchführung von projektorientierten Aufgaben; zeitlich befristet.

Projektstruktur Linienführungskräfte, Stabsexperten und betriebliche Mitarbeiter arbeiten in ständig wechselnder Zusammensetzung in ad hoc-→Projektteams zusammen, um Innovationen und Problemlösungen im Kundenauftrag zu entwickeln.

Projektteam Marktorientierte Einheiten auf Zeit (→Projektmanager, →projektorientierte Matrixstruktur).

Prozeß-Modell →Ablauforganisation.

strategische Spitze Teil einer Organisation, in der Spitzenführungskräfte (Top-Management) mit globalen Aufgabenbereichen die Gesamtverantwortung für die Organisation tragen.

Technostruktur Teil einer Organisation, in der Analytiker die Arbeit anderer effektiver gestalten.

Eine Organisationsstruktur besteht aus einer Aufbauorganisation und einer Ablauforganisation (Prozeß-Modell). Die Aufbauorganisation zeigt, wie die fünf Teile einer Organisation ausgeprägt sind: strategische Spitze, Mittellinie, betrieblicher Kern, Technostruktur und Hilfsstab. Die grafische Darstellung dieser Struktur erfolgt durch ein Organigramm.

Die Aufteilung der durchzuführenden Arbeiten und ihre Zuordnung zu Positionen hängt wesentlich von der Aufgabenspezialisierung ab. Komplexe, eng begrenzte Aufgaben, die eigenverantwortlich ausgeführt werden, machen eine professionelle Arbeit aus. Ist das Verhalten der Mitarbeiter durch Vorschriften und Regeln stark festgelegt, dann liegt eine bürokratische Struktur, sonst eine organische Struktur vor.

Organisatorische Positionen können nach zwei Prinzipien zu Einheiten gruppiert werden:

Gruppierung nach Funktionen und Gruppierung nach Märkten. Eine Matrixstruktur kombiniert beide Gruppierungsalternativen, wobei das Prinzip der Alleinverantwortung aufgegeben wird.

Man unterscheidet eine permanente Matrixstruktur und eine projektorientierte Matrixstruktur. Projektleiter koordinieren Einheiten in einer Matrixstruktur, Projektmanager leiten ein Projektteam in einer projektorientierten Matrixstruktur.

Eine Projektstruktur und eine Profibürokratie sind stabile organisatorische Konfigurationen, die viele Charakteristika einer Software-Entwicklung berücksichtigen und daher in erster Annäherung geeignete Modelle für die Aufbauorganisation einer Software-Abteilung sind. In beiden Fällen sind jedoch Anpassungen an das technische System der Software-Entwicklung erforderlich.

Wichtig für eine kooperative Zusammenarbeit zwischen Fachabteilung und Systemanalyse ist eine klare Aufgabenteilung und genaue Festlegung der jeweiligen Verantwortlichkeiten.

/Mintzberg 92/
Mintzberg H., *Die Mintzberg-Struktur: Organisationen effektiver gestalten*, Landsberg: Verlag Moderne Industrie 1992, 413 Seiten. Amerikanische Originalausgabe bei Prentice Hall 1983
Ausgezeichnetes, systematisch aufgebautes Buch, das die Gestaltungsparameter einer Organisation und ihre Wechselwirkungen ausführlich beschreibt. Es enthält jedoch keine Bezüge zur Software-Technik.

/Brooks 75/
Brooks F.P., *The Mythical Man-Month*, Reading: Addison-Wesley 1975, 195 Seiten
Eines der ersten Bücher, das sich mit Fragen des Software-Managements befaßte.

Muß-Aufgabe *30 Minuten*

1 *Lernziel: Die Auswirkung des Kommunikationsaufwands auf die Produktivität und Zeitdauer berechnen können.*
Ein Softwaresystem im Umfang von 80.000 LOC *(lines of code)* soll in drei Jahren realisiert werden. Die Programmierleistung eines Mitarbeiters beträgt im Jahr durchschnittlich 6.000 LOC. Wegen der stark interdependenten Aufgaben muß jedes Teammitglied mit jedem anderen Teammitglied kommunizieren.
Lösen Sie bitte folgende Teilaufgaben:

- **a** Modifizieren Sie die Formeln **1** und **2** in Abschnitt 3.2.4 so, daß LOC_{gesamt} den Gesamtumfang des Softwareprojekts darstellt und LOC/Jahr die Einzelproduktivität eines Mitarbeiters.
- **b** Stellen Sie eine Tabelle mit folgenden Spalten auf:
  - Kommunikationsaufwand pro Kommunikationspfad p in Prozent
  - Anzahl der Mitarbeiter
  - Gesamter Kommunikationsaufwand
  - Verbleibende Produktivität

  Legen Sie die oben angegebenen Produktivitätszahlen zugrunde und variieren Sie den Kommunikationsaufwand von 1 Prozent bis zu 10 Prozent in 1 Prozent-Schritten.
- **c** Wieviele Mitarbeiter können durch Reduktion der Kommunikation eingespart werden?
- **d** Ermitteln Sie die minimale Zeit t_0 und Mitarbeiterzahl n_0 in Abhängigkeit vom Kommunikationsaufwand p. Erstellen Sie dazu eine Tabelle, die t_0 in Jahren, n_0, p, p_{gesamt} und die Produktivität (jeweils in Prozent) darstellt.

Kann-Aufgabe *15 Minuten*

2 *Lernziel: Die Auswirkung des Kommunikationsaufwands auf die Produktivität und Zeitdauer bei interdependenten Aufgaben berechnen können.*
Überlegen Sie sich den Zusammenhang zwischen der Mitarbeiteranzahl und dem erforderlichen Entwicklungsaufwand. Auch hier gibt es eine optimale Mitarbeiteranzahl. Zeigen Sie, daß die optimale Mitarbeiteranzahl, bezogen auf den Aufwand, immer kleiner ist als die optimale Mitarbeiteranzahl bezogen auf die Zeit.

Muß-Aufgabe *10 Minuten*

3 *Lernziel: Ausgehend von vorgegebenen Charakteristika Vorschläge für Organisationsstrukturen erstellen können.*
Ein Software-Haus erstellt Software fast ausschließlich für zwei Großkunden, die dem Software-Haus jeweils die einzusetzenden Methoden und CASE-Systeme vorschreiben. Die Methoden und CASE-Systeme der Großkunden sind unterschiedlich. Welche Organisationsform wählen Sie, wenn Sie sicherstellen wollen, daß die Spezialisten ihr Wissen regelmäßig austauschen?

Muß-Aufgabe *10 Minuten*

4 *Lernziel: Gruppierung nach Märkten und nach Funktionen unterscheiden können.*
Zählen Sie die Vor- und Nachteile einer Gruppierung nach Funktionen im Vergleich zu einer Gruppierung nach Märkten auf.

Muß-Aufgabe *5 Minuten*

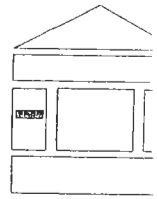
5 *Lernziel: Die fünf grundlegenden Koordinationsmechanismen aufzählen können.*
Auf welche Weise lassen sich Arbeitsabläufe koordinieren?

Muß-Aufgabe *10 Minuten*

6 *Lernziel: Die fünf Teile einer Organisation einschließlich ihrer Aufgaben nennen können.*
Beschreiben Sie die Strukturen, die im allgemeinen zum Aufbau betrieblicher Organisationen verwendet werden.

Weitere Aufgaben befinden sich auf der beiliegenden CD-ROM.

3 Organisation – Prozeß-Modelle



wissen

verstehen

anwenden

beurteilen

- Aufzählen können, welche Aspekte ein Prozeß-Modell festlegen soll.
- Die acht beschriebenen Prozeß-Modelle erklären und ihre Unterschiede darstellen können.
- Beschreiben können, was man unter einer Rolle versteht und Beispiele dafür angeben können.
- Die Begriffe Validation und Verifikation im V-Modell erläutern können.
- Die verschiedenen Arten von Prototypen kennen und ihre Unterschiede aufzeigen können.
- Den Unterschied zwischen einem horizontalen und einem vertikalen Prototypen schildern können.
- Für vorgegebene Szenarien den Ablauf entsprechend einem Prozeß-Modell in der vorgestellten genormten Weise darstellen können.
- Für vorgegebene Szenarien ein Prozeß-Modell oder eine Kombination von Prozeß-Modellen vorschlagen und begründen können.

3.3 Prozeß-Modelle

Jede Software-Erstellung soll in einem festgelegten organisatorischen Rahmen erfolgen.

Ein Prozeß-Modell beschreibt jeweils einen solchen Rahmen. Ein definiertes Prozeß-Modell soll folgendes festlegen:

Prozeß-Modell
- Reihenfolge des Arbeitsablaufs (Entwicklungsstufen, Phasenkonzepte),
- Jeweils durchzuführende Aktivitäten,
- Definition der Teilprodukte einschließlich Layout und Inhalt,
- Fertigstellungskriterien (Wann ist ein Teilprodukt fertiggestellt?),
- Notwendige Mitarbeiterqualifikationen,
- Verantwortlichkeiten und Kompetenzen,
- Anzuwendende Standards, Richtlinien, Methoden und Werkzeuge.

code & fix Das Grundmodell, das in den Anfangstagen der Software-Entwicklung verwendet wurde, enthält zwei Schritte /Boehm 88/:

1 Schreibe ein Programm.

2 Finde und behebe die Fehler in dem Programm.

Die Nachteile dieses Modells sind:

a Nach der Behebung von Fehlern wurde das Programm so umstrukturiert, daß weitere Fehlerbehebungen immer teurer wurden. Dies führte zu der Erkenntnis, daß eine Entwurfsphase vor der Programmierung benötigt wird.

b Selbst gut entworfene Software wurde vom Endbenutzer oft nicht akzeptiert. Dies führte zu der Erkenntnis, daß eine Definitionsphase vor dem Entwurf benötigt wird.

c Fehler waren schwierig zu finden, da Tests schlecht vorbereitet und Änderungen unzureichend durchgeführt wurden. Dies führte zu einer separaten Testphase.

Ausgehend von diesem Grundmodell entstanden im Laufe der Zeit folgende Modelle, die im weiteren genauer betrachtet werden:
- Das Wasserfall-Modell
- Das V-Modell
- Das Prototypen-Modell
- Das evolutionäre/inkrementelle Modell
- Das objektorientierte Modell
- Das nebenläufige Modell
- Das Spiralmodell

Notation Um die Modelle besser vergleichen zu können, werden – neben den Orginaldarstellungen – alle Modelle zusätzlich in einer einheitlichen Notation dargestellt. Aktivitäten werden durch ein Rechteck dargestellt, Ergebnisse einer Aktivität durch ein Oval beschrieben. Pfeile geben an, welche Aktivitäten zu welchen Ergebnissen bzw. Produkten führen bzw. welche Produkte in welche Aktivitäten eingehen.

Aktivität

Produkt

Die horizontale Ausdehnung von Rechteck und Oval symbolisiert die Aktivitäts- bzw. Produktbreite. Umfaßt eine Definitionsaktivität

die volle Produktfunktionalität und -leistung, dann ist das Rechteck breit dargestellt. Wird nur ein Teilprodukt definiert, dann ist das Symbol schmal gezeichnet.

Um eine Vorstellung von den Auswirkungen der Prozeß-Modelle zu erhalten, wird als Beispiel die Entwicklung der Seminarorganisation gewählt.

Hauptkapitel I und 3 sowie Anhang I A

3.3.1 Das Wasserfall-Modell

Das Wasserfall-Modell ist eine Weiterentwicklung des »*stagewise models*« /Benington 56/. Dieses Modell legt fest, daß Software in sukzessiven Stufen entwickelt wird. Das in /Royce 70/ vorgeschlagene Wasserfall-Modell erweitert das »*stagewise model*« um Rückkopplungsschleifen zwischen den Stufen. Die Rückkopplungen werden gleichzeitig auf angrenzende Stufen begrenzt, um teure Überarbeitungen über mehrere Stufen hinweg zu vermeiden (Abb. 3.3-1).

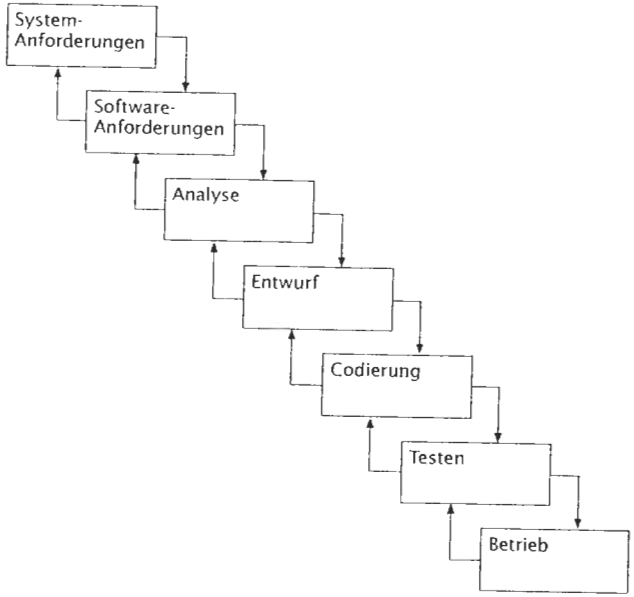


Abb. 3.3-1: Wasserfall-Model mit den ursprünglich vorgesehene Phasen /Royce 70/

Dieses Modell wurde in /Boehm 81/ **Wasserfall-Modell** genannt, weil die Ergebnisse einer Phase wie bei einem Wasserfall in die nächste Phase fallen. Abb. 3.3-2 zeigt das Wasserfall-Modell in der oben beschriebenen Vergleichsnotation, beschränkt auf die Phasen Definition, Entwurf und Implementierung.

Wasserfall-Modell

Das Wasserfall-Modell besitzt folgende Charakteristika:
- Jede Aktivität ist in der richtigen Reihenfolge und in der vollen Breite vollständig durchzuführen.

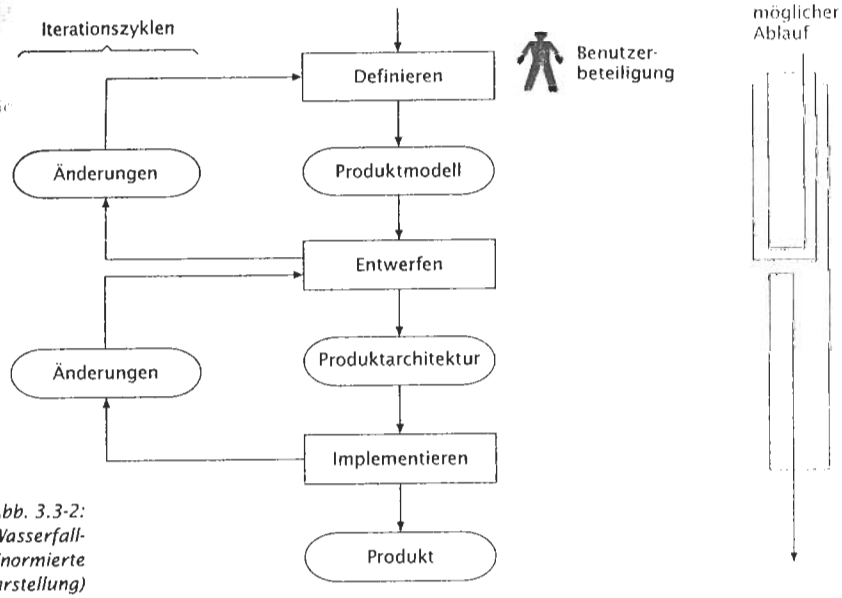


Abb. 3.3-2: Das Wasserfall-Modell (normierte Darstellung)

- Am Ende jeder Aktivität steht ein fertiggestelltes Dokument, d.h. das Wasserfall-Modell ist ein dokumenten-getriebenes Modell.
- Der Entwicklungsablauf ist sequentiell, d.h. jede Aktivität muß beendet sein, bevor die nächste anfängt.
- Es orientiert sich am *top-down*-Vorgehen.

Kapitel IV 1.2

- Es ist einfach, verständlich und benötigt nur wenig Managementaufwand.
- Eine Benutzerbeteiligung ist nur in der Definitionsphase vorgesehen, anschließend erfolgen der Entwurf und die Implementierung ohne Beteiligung des Benutzers bzw. Auftraggebers.

Beispiel Das Produkt »Seminarorganisation« wird im Wasserfall-Modell folgendermaßen erstellt:

1. Unter Einbeziehung des Auftraggebers/Benutzers wird eine Produktdefinition für alle Anforderungen des Auftraggebers ermittelt. Nach der Abnahme des Produktmodells durch den Auftraggeber ist die Definitionsphase abgeschlossen.
2. In der Entwurfsphase wird ausgehend vom Produktmodell eine Produktarchitektur für das gesamte Produkt entwickelt. Stellt sich heraus, daß einige Anforderungen des Produktmodells nicht realisierbar sind, dann wird ein Änderungsdokument erstellt. Man kehrt in die Definitionsphase zurück und arbeitet die Änderungen unter Beteiligung des Auftraggebers in das Produktmodell ein. Anschließend beginnt wieder die Entwurfsphase. Als Ergebnis der Entwurfsphase wird die Produktarchitektur an die Implementierungsphase übergeben.
3. Die Produktarchitektur wird implementiert. Es entsteht das fertige Produkt.

Die aufgeführten Charakteristika haben dem Wasserfall-Modell zu einer weiten Verbreitung verholfen. Es ist die Basis für die meisten Auftragsstandards in Behörden und der Industrie.

Das Wasserfall-Modell besitzt jedoch auch einige Nachteile: Nachteile

- Nicht immer ist es sinnvoll, alle Entwicklungsschritte in der vollen Breite und vollständig durchzuführen.
- Nicht immer ist es sinnvoll, alle Entwicklungsschritte sequentiell durchzuführen.
- Es besteht die Gefahr, daß die Dokumentation wichtiger wird als das eigentliche System.
- Die Risikofaktoren werden u.U. zu wenig berücksichtigt, da immer der einmal festgelegte Entwickungsablauf durchgeführt wird.

Trotz dieser Nachteile hat das Wasserfall-Modell wesentlich zu einem disziplinierten, sichtbaren und kontrollierbaren Prozeßablauf beigetragen.

3.3.2 Das V-Modell

Das V-Modell ist eine Erweiterung des Wasserfall-Modells. Es integriert die Qualitätssicherung in das Wasserfall-Modell /Boehm 81, 84/. Die Verifikation und Validation der Teilprodukte sind Bestandteile des V-Modells.

Unter **Verifikation** wird die Überprüfung der Übereinstimmung zwischen einem Software-Produkt und seiner Spezifikation verstanden. Verifikation

Unter **Validation** wird die Eignung bzw. der Wert eines Produktes bezogen auf seinen Einsatzzweck verstanden. Validation

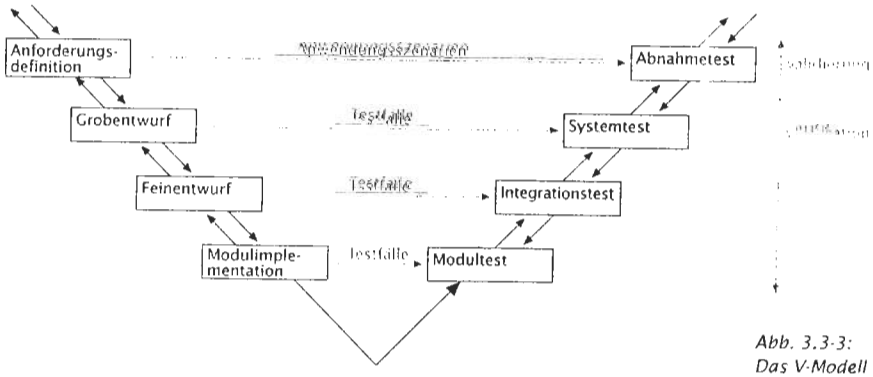
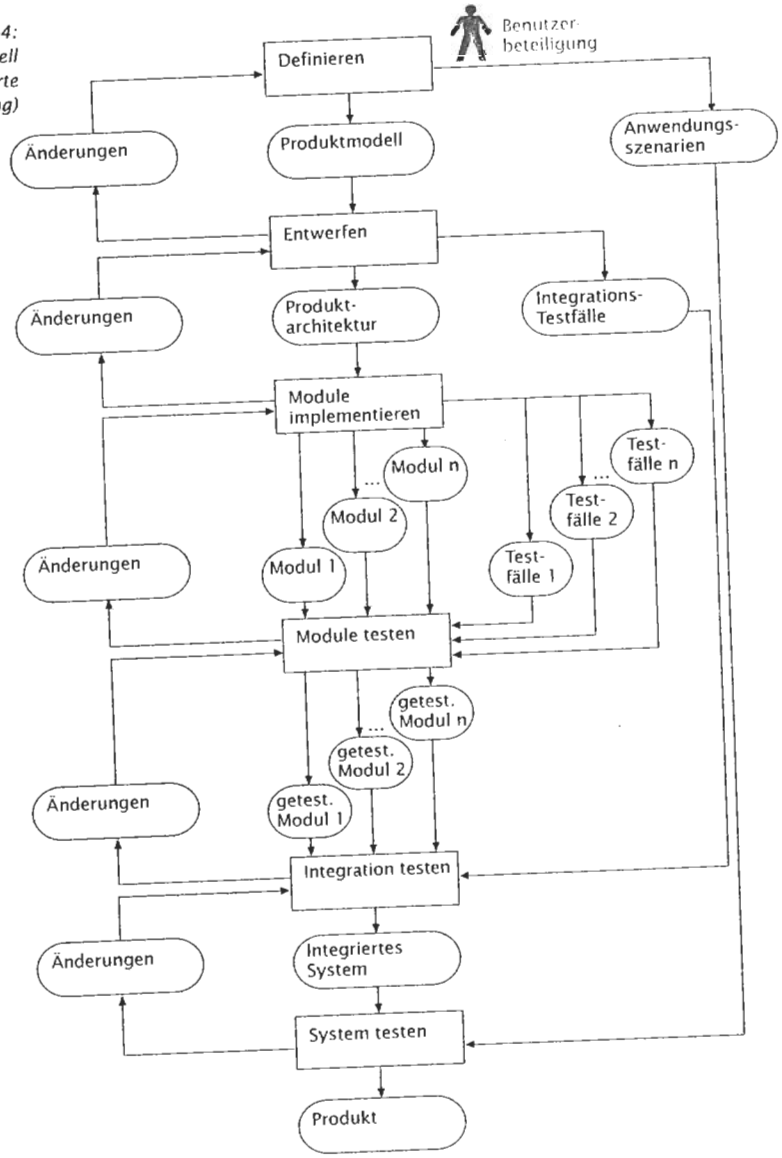


Abb. 3.3-3: Das V-Modell

Umgangssprachlich bedeuten beide Begriffe folgendes:
Verifikation: »Wird ein korrektes Produkt entwickelt?«
Validation: »Wird das richtige Produkt entwickelt?«
Berücksichtigt man die Verifikation und die Validation, dann läßt sich dieses Prozeß-Modell in Form eines V darstellen (Abb. 3.3-3). Die normierte Darstellung des V-Modells zeigt Abb. 3.3-4.

Abb. 3.3-4: Das V-Modell (normierte Darstellung)



Ausgehend von diesem V-Modell wurde ein Vorgehensmodell zunächst für die Bundeswehr und anschließend für Behörden entwikkelt. Es wurde vom Bundesinnenminister im Bundesanzeiger veröffentlicht /KBSt 92/ (siehe auch /Bröhl, Dröschel 93/) und dient der Standardisierung der Software-Bearbeitung im Bereich der Bundesverwaltung. Auch in der Industrie wird dieser Standard inzwischen angewandt. Die folgenden Ausführungen beziehen sich auf die weiterentwickelte Fassung von 1997, die auch die Entwicklung von Hardware umfaßt (/Versteegen 96/, /V-Modell 97/).

Im folgenden bezieht sich der Begriff V-Modell auf das Vorgehensmodell für IT-Systeme des Bundes.

Dieses **V-Modell** (Vorgehensmodell) legt die Aktivitäten und Produkte des Entwicklungs- und Pflegeprozesses fest. Außerdem werden die Produktzustände und die logischen Abhängigkeiten zwischen Aktivitäten und Produkten dargestellt. Aktivitäten und Produkte werden auf verschiedenen Abstraktionsebenen beschrieben.

Die Dokumentation zum V-Modell befindet sich auf der CD-ROM

V-Modell

Das V-Modell ist in vier Submodelle gegliedert:

Submodelle

- Systemerstellung (SE),
- Qualitätssicherung (QS),
- Konfigurationsmanagement (KM) und
- Projektmanagement (PM).

Da das V-Modell ursprünglich für eingebettete Systeme entwickelt wurde, wird Software immer als Bestandteil eines informationstechnischen Systems (IT-System) angesehen.

Ein System ist eine Funktionseinheit der obersten Ebene. Jedes System kann – muß aber nicht – in Segmente unterteilt werden. Es gibt IT-Segmente, die IT-Anteile enthalten, und Nicht-IT-Segmente. IT-Segmente werden weiter untergliedert in Software-Einheiten (SW-Einheiten) und Hardware-Einheiten (HW-Einheiten).

System
Segment
IT = Informationstechnik

SW-Einheiten bestehen aus SW-Komponenten, die wiederum aus SW-Komponenten niederer Ordnung und letztlich aus SW-Modulen und/oder Datenbanken zusammengesetzt sind. Diese hierarchisch gegliederte Erzeugnisstruktur zeigt Abb. 3.3-5.

SW-Komponente

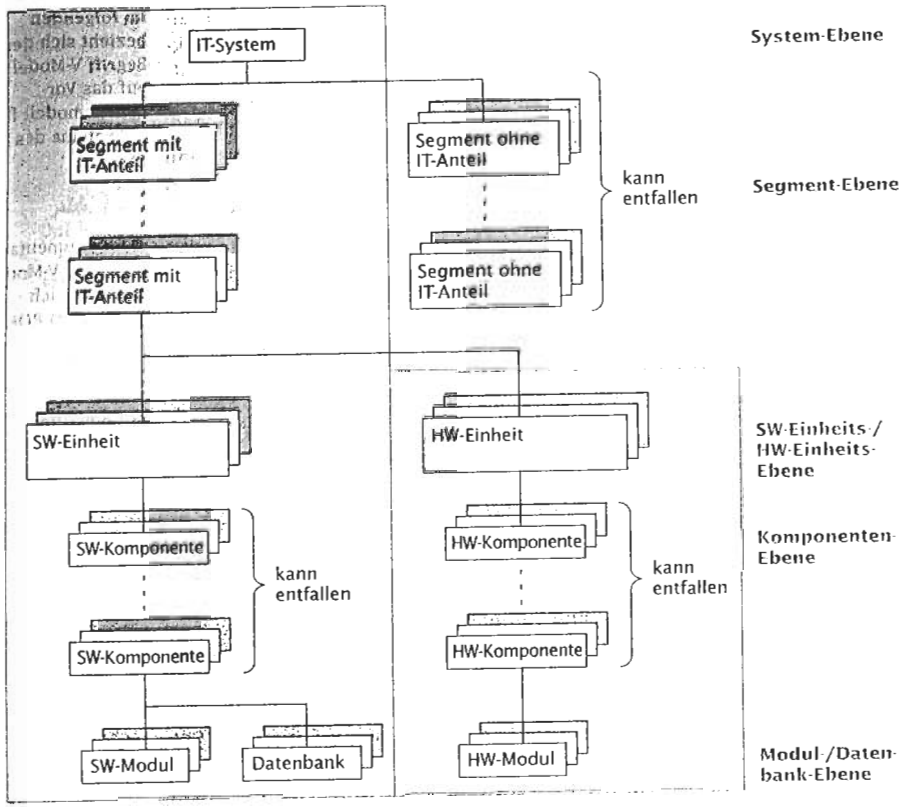
Die Grundelemente des V-Modells sind die Aktivitäten und Produkte, die während des IT-Systementwicklungsprozesses durchgeführt bzw. bearbeitet werden. Eine Aktivität ist eine Tätigkeit, die bezogen auf ihr Ergebnis und ihre Durchführung genau beschrieben werden kann. Sie wird durch ein Rechteck dargestellt. Aktivitäten können aus Teilaktivitäten bestehen, die zu definierten Zwischenergebnissen führen. Auf der obersten Darstellungsebene spricht man von Hauptaktivitäten. Ein Produkt ist das Ergebnis einer Aktivität bzw. der Bearbeitungsgegenstand einer Aktivität. Produkte werden durch Ovale dargestellt. Sie können in Teilprodukte zerlegt werden.

Aktivität

Produkt

Ziel einer Aktivität kann es sein,

- ein Produkt zu erstellen,
- den Zustand eines Produkts zu ändern oder
- den Inhalt eines Produkts zu ändern.



Legende:
- Entwicklung vollständig durch das Vorgehensmodell geregelt.
- Entwicklung in der Handbuchsammlung beschrieben (Handbuch »Hardware-Erstellungen«)

Abb. 3.3-5: Erzeugnisstruktur des V-Modells /V-Modell 97, S. 2-2/

Eine Aktivitätenbeschreibung ist eine Arbeitsanleitung, der bei der Ausführung der Aktivität zu folgen ist. Eine Produktbeschreibung legt die Inhalte des Produkts fest. Sie erfolgt nach einem festen Muster, dem Produkt-Muster.

Produkt-Muster

Gewisse Produkte durchlaufen im Entwicklungsprozeß verschiedene Zustände. Ein Zustandswechsel wird durch eine Aktivität ausgelöst. Abb. 3.3-6 zeigt den Zustandsautomaten für Produkte.

Zustände

- Im Zustand »geplant« ist das Produkt in der Planung vorgesehen. (geplant)
- Im Zustand »in Bearbeitung« befindet sich das Produkt im privaten Entwicklungsbereich des Entwicklers oder unter seiner Kontrolle in der Produktbibliothek. (in Bearbeitung)
- Im Zustand »vorgelegt« ist das Produkt aus Erstellersicht fertig und wird in die Konfigurationsverwaltung übernommen. Es wird einer Qualitätsüberprüfung unterzogen. (vorgelegt)

Abschnitt 6.3.4

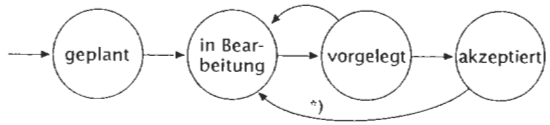


Abb. 3.3-6: Zulässige Zustandsübergänge von Produkten /V-Modell 97, S. 2-6/

*) Dieser Übergang wird durch das Konfigurationsmanagement verursacht und führt zu einer neuen Produkt-Version

Bei Ablehnung geht es in den Zustand »in Bearbeitung« zurück, sonst geht es in den Zustand »akzeptiert« über. Vom Zustand »vorgelegt« ab führen Modifikationen zu einer Fortschreibung der Versionsangabe.

- Im Zustand »akzeptiert« wurde das Produkt von der Qualitätssicherung überprüft und freigegeben. Es darf nur innerhalb einer neuen Version geändert werden. (akzeptiert)

Abb. 3.3-7 zeigt das Submodell »Systemerstellung« im Funktionsüberblick. Abb. 3.3-8 zeigt alle Produkte des V-Modells und ihre Einordnung in die hierarchische Erzeugnisstruktur (siehe Abb. 3.3-5).

Zu jedem Produkt gibt es ein inhaltliches Gliederungsschema (Produktmuster). Die Anwenderforderungen werden entsprechend dem Produktmuster der Abb. 3.3-9 beschrieben.

Das prinzipielle Zusammenwirken der vier Submodelle zeigt Abb. 3.3-10.

Rollen beschreiben die notwendigen Erfahrungen, Kenntnisse und Fähigkeiten, um Aktivitäten durchzuführen. In jedem Submodell gibt es

Rollen

- einen Manager, der die Rahmenbedingungen für die Aktivitäten des Submodells festlegt und die oberste Entscheidungsinstanz ist,
- einen Verantwortlichen, der die Aufgaben des Submodells plant, steuert und kontrolliert,
- einen bzw. mehrere Durchführende, die die geplanten Aufgaben der Submodelle bearbeiten.

Tab. 3.3-1 zeigt die Rollen im V-Modell. Die Zuordnung von Rollen zu Aktivitäten für das SE-Submodell zeigt Tab. 3.3-2. Zu jeder Rolle gibt es ein Anforderungsprofil.

Als Beispiele für die Verfeinerung der Hauptaktivitäten SE 1 und SE 2 sind die Abb. 3.3-11 und die Abb. 3.3-12 angegeben.

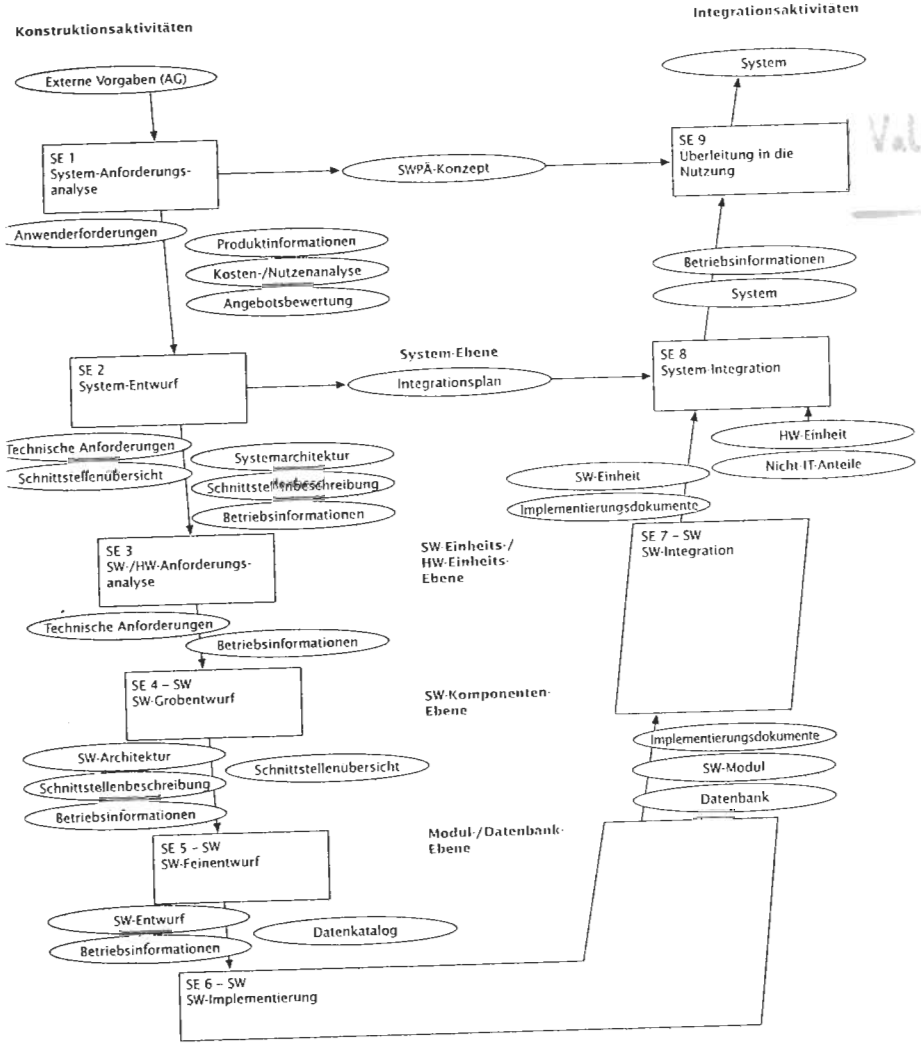
Als Beispiel für eine weitere Verfeinerung wird die Aktivität SE 1.1 »Ist-Aufnahme/-Analyse« näher betrachtet. Diese Aktivität ist folgendermaßen abzuwickeln:

»Der Schwerpunkt der Ist-Aufnahme/-Analyse liegt auf der fachlichen, anwenderorientierten Seite. Im Rahmen dieser Aktivität sind Informationen über den Ist-Zustand zu beschaffen, zu analysieren und zu dokumentieren.

Hier kann eine Organisationsanalyse durchgeführt werden. Im Rahmen einer Organisationsanalyse sind folgende Aspekte zu berücksichtigen:

- Organisationsanalyse vorbereiten (Zielsetzung/Leistungen der fachlichen Aufgaben ermitteln, Analysetiefe festlegen, entsprechend Unterlagen für Workshops oder Fragebögen für Interviews vorbereiten),
- Geschäftsprozesse aufnehmen (Haupt- und Teilgeschäftsprozesse abgrenzen, Schnittstellen zwischen Prozessen festlegen, Geschäftsprozesse in einem Ist-Modell abbilden),

Abb. 3.3-7: nktionsüberblick Submodell SE ystemerstellung) /V-Modell 97, S. 4-50 (vereinfacht)/



Legende: AG – Auftraggeber
SWPA – Software-Pflege und -Änderung
SE – Systemerstellung

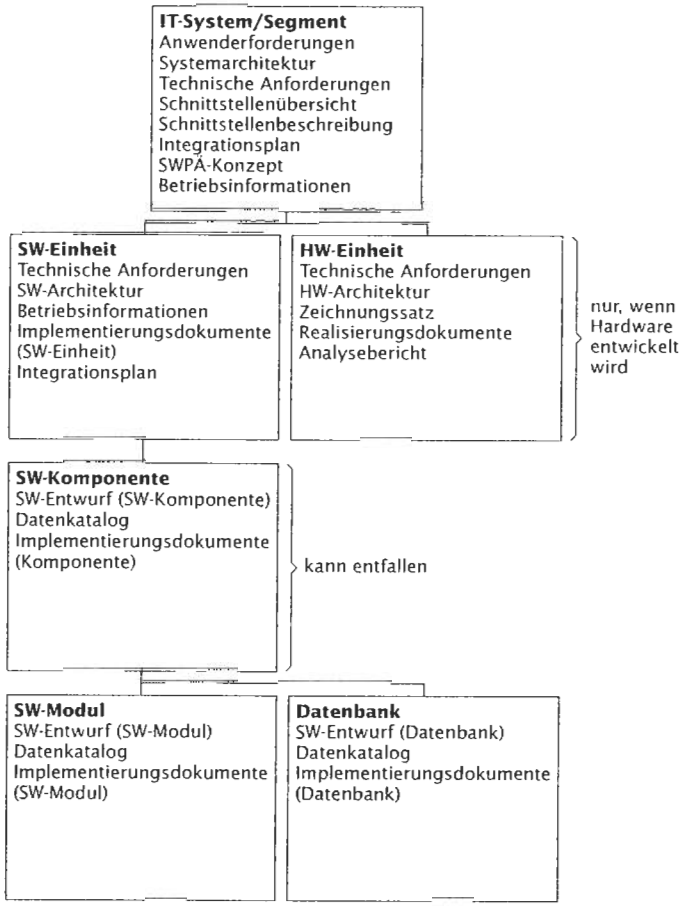


Abb. 3.3-8: Produkte des Submodells Systemerstellung (Produktbaum) (in Anlehnung an /V-Modell 97, S. 2-10/)

Legende: SWPÄ = Software-Pflege und -Änderung

Zitat

- Geschäftsprozesse analysieren (Ist-Modell durch Einbeziehung der Prozeßbetroffenen und des Prozeßverantwortlichen evaluieren, Schwachstellen z.B. Redundanzen, Liegezeiten und deren Ursachen ermitteln. Bei sehr komplexen Geschäftsprozessen kann eine Simulation des Ist-Modells zur Evaluation sinnvoll sein.).
- Analyseergebnisse auswerten (Vorschläge für Verbesserungen sammeln und priorisieren, u.a. Möglichkeiten für IT-Unterstützung ermitteln).

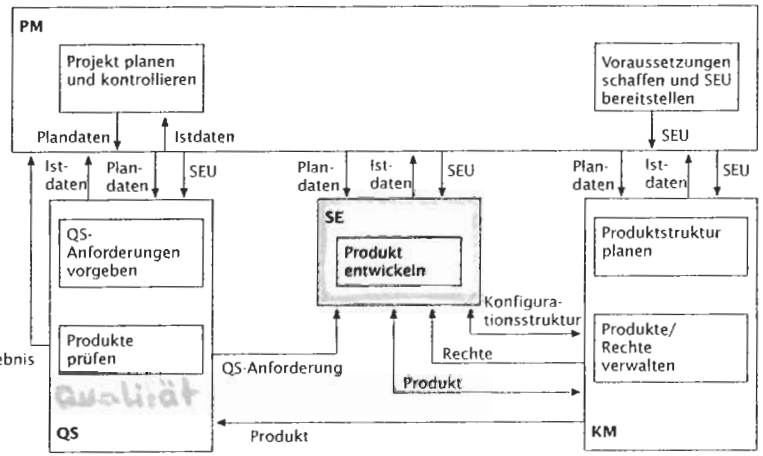
Die Analyseergebnisse dienen als Ausgangspunkt für die Definition von Anforderungen an eine künftige Aufbau- und Ablauforganisation beim Nutzer und als Basis für eine nachfolgende Geschäftsprozeßmodellierung sowie deren Umsetzung in die Praxis. Eine Organisationsanalyse sollte vor allem dann in Betracht gezogen werden, wenn

Abb. 3.3-9: Produktmuster »Anwenderforderungen«

1 **Allgemeines**
Vorspann für alle Dokumente (Deckblatt, Kurzbeschreibung, Verzeichnisse, Änderungsübersicht).
2 **Ist-Aufnahme und Ist-Analyse**
Nur relevant, wenn ein System dieser Art existiert und durch ein neues ersetzt werden soll.
3 **IT-Sicherheitsziel**
Anforderungen an Verfügbarkeit, Integrität oder Vertraulichkeit von bestimmten Funktionen oder Informationen.
4 **Bedrohungs- und Risikoanalyse**
Relevante Bedrohungen für das System und die damit verbundenen Risiken und zu erwartenden Schäden sind zu ermitteln.
5 **IT-Sicherheit**
Maßnahmen in der Umgebung des Systems (z.B. Zutrittssicherung) oder im System selbst (z.B. Paßwortschutz, Protokollierung).
6 **Fachliche Anforderungen**
6.1 Grobe Systembeschreibung
Beschreibung des Gesamthorizonts (Gesamtfunktionalität im Endausbau, quantitative Abschätzungen)
6.2 Organisatorische Einbettung
6.3 Nutzung
6.4 Kritikalität des Systems
6.5 Externe Schnittstellen
6.6 Beschreibung der Funktionalität
Fachliche Strukturierung der Funktionalität aus Anwendersicht; Definition von Geschäftsprozessen, gegebenenfalls auch Beschreibung der Daten. Die Funktionalität wird in Bereiche gegliedert, z.B. *Mailing* und Kommunikation, Daten- und Dokumentenhaltung, IT-Sicherheit, Mensch-Maschine-Schnittstelle, Bürofunktionen.
6.7 Qualitätsanforderungen
Basis ist die ISO 9126.
7 **Randbedingungen**
7.1 Technische Randbedingungen
7.2 Organisatorische Randbedingungen
7.3 Sonstige Randbedingungen

Quelle: /V-Modell 97, S. 8-3ff/

Abb. 3.3-10: Zusammenwirken der vier Submodelle /V-Modell 97, S. 2-9/



Legende: SEU = Software-Entwicklungs-Umgebung

	Manager	Verantwortliche	Durchführende
Submodell PM	Projektmanager	Projektleiter Rechtsverantwortlicher Controller	Projektadministrator
Submodell SE	Projektmanager IT-Beauftragter Anwender	Projektleiter	Systemanalytiker Systemdesigner SW-Entwickler HW-Entwickler Technischer Autor SEU-Betreuer Datenadministrator IT-Sicherheitsbeauftragter Datenschutzbeauftragter Systembetreuer Prüfer
Submodell QS	Q-Manager	QS-Verantwortlicher	
Submodell KM	KM-Manager	KM-Verantwortlicher	KM-Administrator

Legende: IT = Informations-Technik, Q = Qualität, KM = Konfigurationsmanagement, QS = Qualitätssicherung, SEU = Software-Entwicklungs-Umgebung, PM = Projektmanagement

Tab. 3.3-1: Rollen im V-Modell /V-Modell 97, S. R-2/

die Erledigung fachlicher Aufgaben ohne eine wirksame IT-Unterstützung nicht oder nicht mehr gewährleistet werden kann. Dabei sind die möglichen Wechselwirkungen zwischen Organisationsentwicklung und Realisierung der zugehörigen IT-Unterstützung unbedingt zu beachten, da eine optimale Gestaltung der Aufbau- und Ablauforganisation beim Nutzer gleichzeitig einen Haupteinflußfaktor für die IT-Unterstützung darstellt (z.B. notwendige Ausstattung von Arbeitsplätzen mit Rechnerleistung, Peripherie, Software, Vernetzungs- und Kommunikationseinrichtungen) und umgekehrt technologische Neuerungen auf dem Gebiet der IT die Möglichkeiten zur Organisationsgestaltung unmittelbar beeinflussen können (z.B. beim Einsatz von Vorgangssteuerungs- und Archivierungssystemen)./V-Modell 97, S. 4-4/.

Das V-Modell erhebt den Anspruch auf Allgemeingültigkeit, d.h. es soll für unterschiedliche Produktentwicklungen einsetzbar sein. Eine Anpassung an eine konkrete Entwicklung geschieht durch das »Maßschneidern« *(Tailoring)* der Produkte und Aktivitäten. *Tailoring*

Dieses »Maßschneidern« geschieht in zwei Stufen:

- Ausschreibungsrelevantes *Tailoring*
  Vor Entwicklungsbeginn werden die sinnvollen Aktivitäten und Produkte festgelegt. Dies kann durch Streichungen geschehen oder durch Wahl eines Vorhabentyps, für den die Streichungen bereits vorgenommen wurden (standardisiertes *Vortailoring*).
- Technisches *Tailoring*
  Im Entwicklungsverlauf werden situationsabhängig Aktivitäten und Produktinhalte beim Beginn jeder Hauptaktivität festgelegt. Es wird entschieden, ob eine Aktivität oder ein Produkt im konkreten Fall sinnvoll ist oder entfallen kann.

Aktivität		Rolle	Systemanalytiker	Systemdesigner	SW-Entwickler	HW-Entwickler	Technischer Autor	Systembetreuer	SEU-Betreuer	Datenadministrator	Datenschutzbeauftragter	IT-Sicherheitsbeauftragter	IT-Beauftragter	Anwender	Projektleiter
SE 1.1	Ist-Aufnahme/-Analyse durchführen		v	b										m	
SE 1.2	Anwendungssystem beschreiben		v											m	
SE 1.3	Kritikalität und Anforderungen an die Qualität definieren		v											m	
SE 1.4	Randbedingungen definieren		v											m	
SE 1.5	System fachlich strukturieren		v				m	m		m	b	m	m	m	
SE 1.6	Bedrohung und Risiko analysieren											v			
SE 1.7	Forderungscontrolling durchführen		m	m										v	m
SE 1.8	Software-Pflege und Änderungs-Konzept erstellen			v										m	m
SE 2.1	System technisch entwerfen			v			m						m		m
SE 2.2	Wirksamkeitsanalyse durchführen											v			
SE 2.3	Realisierbarkeit untersuchen		m	v								m	m		m
SE 2.4	Anwenderforderungen zuordnen			v											
SE 2.5	Schnittstelle beschreiben			v											
SE 2.6	System-Integration spezifizieren			v			m								
SE 3.1	Allgemeine Anforderungen an die SW-/HW-Einheit definieren			v											
SE 3.2	Anforderungen an die externen Schnittstellen der SW-/HW-Einheit präzisieren			v											
SE 3.3	Anforderungen an die Funktionalität definieren			v											
SE 3.4	Anforderungen an die Qualität der SW-/HW-Einheit definieren			v											
SE 3.5	Anforderungen an Entwicklungs- und SWPA-Umgebung definieren			v			m		m						
SE 4.1-SW	SW-Architektur entwerfen					v	m								
SE 4.2-SW	SW-interne und externe Schnittstellen entwerfen					v									
SE 4.3-SW	SW-Integration spezifizieren					v	m								
SE 5.1-SW	SW-Komponente/-Modul/Datenbank beschreiben					v	m								
SE 5.2-SW	Betriebsmittel- und Zeitbedarf analysieren					v									
SE 6.1-SW	SW-Module codieren					v									
SE 6.2-SW	Datenbank realisieren					v									
SE 6.3-SW	Selbstprüfung des SW-Moduls/ der Datenbank durchführen					v									
SE 7.1-SW	Zur SW-Komponente integrieren					v									
SE 7.2-SW	Selbstprüfung der SW-Komponente durchführen					v									
SE 7.3-SW	Zur SW-Einheit integrieren					v									
SE 7.4-SW	Selbstprüfung der SW-Einheit durchführen					v									

Legende: v = verantwortlich m = mitwirkend b = beratend

Ziel des *Tailoring* ist es, eine übermäßige Papierflut sowie sinnlose Dokumente zu vermeiden, auf der anderen Seite aber zu verhindern, daß wichtige Dokumente fehlen.

Aktivität		Rolle	Systemanalytiker	Systemdesigner	SW-Entwickler	HW-Entwickler	Technischer Autor	Systembetreuer	SEU-Betreuer	Datenadministrator	Datenschutzbeauftragter	IT-Sicherheitsbeauftragter	IT-Beauftragter	Anwender	Projektleiter
SE 4.1-HW	Lösungsvorschläge erarbeiten und bewerten					v									
SE 4.2-HW	Grobentwurf einer HW-Einheit erarbeiten					v	m								
SE 4.3-HW	HW-interne Schnittstellen spezifizieren					v									
SE 4.4-HW	HW-Integration spezifizieren					v	m								
SE 5.1-HW	Detailentwürfe für HW-Komponenten/ HW-Module erstellen					v	m								
SE 5.2-HW	Zeichnungssatz erstellen					v									
SE 5.3-HW	Analysen und Nachweise durchführen					v									
SE 6.1-HW	HW-Komponente/HW-Modul anfertigen					v									
SE 6.2-HW	Selbstprüfung durchführen					v									
SE 7.1-HW	Zur HW-Teilstruktur integrieren					v									
SE 7.2-HW	HW-Teilstrukturen-Selbstprüfung durchführen					v									
SE 7.3-HW	Zur HW-Einheit integrieren					v									
SE 7.4-HW	HW-Einheiten-Selbstprüfung					v									
SE 8.1	Zum System integrieren		b	v	m	m									
SE 8.2	Selbstprüfung des Systems durchführen			v	m	m									
SE 8.3	Produkt bereitstellen			v	m	m	m	m							
SE 9.1	Beitrag zur Einführungsunterstützung leisten						m	v						m	
SE 9.2	System installieren							v	m						
SE 9.3	In Betrieb nehmen							v						m	

Legende: v = verantwortlich m = mitwirkend b = beratend

Tab. 3.3-2b: Aktivitäten/ Rollen-Matrix für das Subsystem SE /V-Modell 97, S. R-14/

Beispiel

Um ein Gefühl für die Anzahl der Aktivitäten und Teilprodukte bei der Anwendung des V-Modells zu bekommen, wird ein Beispiel durchgeführt.

IT = Informationstechnik

Für »kleine administrative IT-Vorhaben« gibt es ein standardisiertes *Vortailoring*. Ein solches Vorhaben besitzt folgende Charakteristika:

- Das Projekt wird meist von einem oder zwei Mitarbeitern durchgeführt.
- Der Aufwand ist kleiner oder gleich einem halben Mitarbeiterjahr.
- Die Komplexität sowohl der Funktionen als auch der Daten ist gering.
- Die Wartbarkeitsanforderungen sind gering, d.h. es sind nur minimale Änderungen zu erwarten.
- Beispiele: Statistikprogramme, Geschäftsgrafik, dBase-Anwendungen mit Auswertungen.

Folgende Produkte müssen für ein solches Vorhaben erstellt werden (in Klammern sind die zugehörigen Aktivitäten angegeben, siehe auch Tab. 3.3-2):

- Anwenderforderungen (1.1, 1.2, 1.4, 1.5, 1.6)
- Systemarchitektur (2.1)

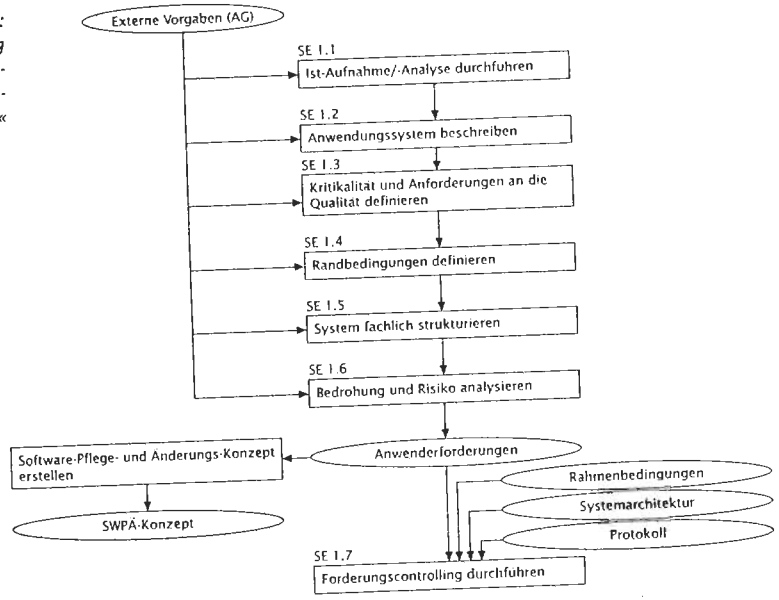


Abb. 3.3-11: Aktivitätszerlegung von SE 1 »System-Anforderungs-analyse«

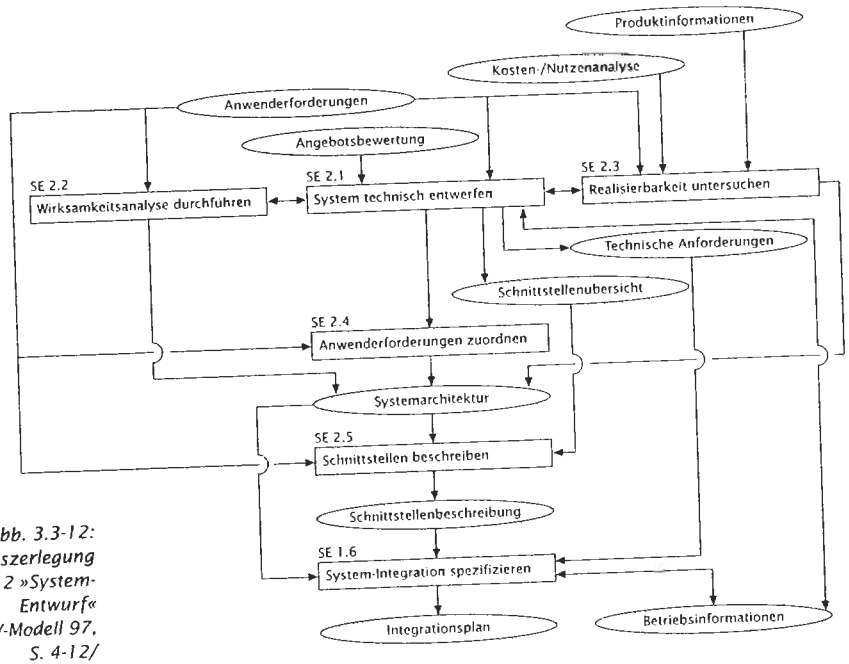


Abb. 3.3-12: Aktivitätszerlegung von SE 2 »System-Entwurf« /V-Modell 97, S. 4-12/

- Technische Anforderungen (2.1, 3.1, 3.3, 3.4)
- Schnittstellenübersicht (2.1, 4.1)
- Betriebsinformationen (2.1, 4.1, 8.3)
- SW-Modul (6.1)
- SW-Einheit (7.3)
- Segment (8.1)
- System (8.1, 9.2, 9.3)

Von 17 im V-Modell festgelegten Teilprodukten müssen für dieses kleine Vorhaben bereits 11 Teilprodukte erstellt werden. Von 37 Aktivitäten sind 18 durchzuführen.

Bewertung

Das in/V-Modell97/ beschriebene V-Modell besitzt folgende Vorteile: Vorteile

- ⊞ Integrierte, detaillierte Darstellung von Systemerstellung, Qualitätssicherung, Konfigurationsmanagement und Projektmanagement.
- ⊞ Generisches Vorgehensmodell mit definierten Möglichkeiten zum Maßschneidern auf projektspezifische Anforderungen.
- ⊞ Ermöglicht eine standardisierte Abwicklung von Systemerstellungs-Projekten.
- ⊞ Gut geeignet für große Projekte, insbesondere für eingebettete Systeme.

Folgende Nachteile besitzt das V-Modell: Nachteile

- ⊟ Die Vorgehenskonzepte, die für große eingebettete Systeme sinnvoll sind, werden unkritisch auf andere Anwendungstypen übertragen.
- ⊟ Für kleine und mittlere Softwareentwicklungen (wenn es sich um keine eingebetteten Systeme handelt) führt das V-Modell zu einer unnötigen Produktvielfalt und zu einer Software-Bürokratie (siehe Beispiel).
- ⊟ Ohne geeignete CASE-Unterstützung ist das V-Modell nicht handhabbar.
- ⊟ Die 25 definierten Rollen im V-Modell sind – außer für Großprojekte – für den Durchschnitt der Software-Entwicklungen unrealistisch.
- ⊟ Es fehlen in der Dokumentation vollständige Beispiele.
- ⊟ Gefahr, daß bestimmte Software-Methoden festgeschrieben werden, da das V-Modell nicht methodenneutral ist (Trennung in Funktions- und Datensicht).
- ⊟ Unnötige Verwendung englischer Begriffe wie *Tailoring, Baseline, Prototyping.*
- ⊟ Verwendung unüblicher deutscher Begriffe, z.B. Forderungen anstelle von Anforderungen.

3.3.3 Das Prototypen-Modell

Erfahrungsgemäß treten bei einer Software-Entwicklung eine Reihe von Problemen auf, die mit den traditionellen Prozeß-Modellen – wie Wasserfallmodell und V-Modell – nicht gelöst werden können:

Probleme

- Der Auftraggeber und der Endbenutzer sind oft nicht in der Lage, ihre Anforderungen an ein neues System explizit und/oder vollständig zu formulieren. Traditionelle Prozeß-Modelle verlangen jedoch zu Beginn der Entwicklung – in der Definitionsphase – eine vollständige Spezifizierung der Anforderungen.
- Während der Entwicklung ist oft eine wechselseitige Koordination zwischen Entwicklern und Anwendern erforderlich, wobei jede Gruppe kontinuierlich von der anderen lernt. Traditionelle Prozeß-Modelle beenden diese Kooperation, wenn die Anforderungen fertiggestellt sind.
- Software-Entwicklungsabteilungen ziehen sich nach der Definitionsphase vom Auftraggeber zurück und präsentieren erst nach der Fertigstellung das Ergebnis dem Auftraggeber. Diese Organisationsstruktur wird durch traditionelle Prozeß-Modelle unterstützt.
- Für manche Anforderungen gibt es unterschiedliche Lösungsmöglichkeiten. Diese Lösungsmöglichkeiten müssen experimentell erprobt und mit dem Auftraggeber diskutiert werden, bevor eine Entscheidung für eine Lösung getroffen werden kann.
- Die Realisierbarkeit mancher Anforderungen läßt sich manchmal theoretisch nicht garantieren, z.B. bei Echtzeitanforderungen. Es ist daher erforderlich, diese speziellen Anforderungen vor Abschluß der Definitionsphase zu realisieren.
- In der Akquisitionsphase muß der Auftraggeber von der prinzipiellen Durchführbarkeit einer Idee oder der Handhabung überzeugt werden. Traditionelle Modelle bieten für diese Aufgabe keine Lösungsmöglichkeiten.

Diese Probleme können durch das Prototypen-Modell gelöst werden – zumindest teilweise.

Software-Prototyp vs. Prototyp

Ein Software-Prototyp unterscheidet sich von einem Prototyp in anderen Ingenieurdisziplinen in folgenden Punkten:

- Ein Software-Prototyp ist *nicht* das erste Muster einer großen Serie von Produkten, wie es z.B. bei der Massenproduktion in der Automobilindustrie der Fall ist. Die Vervielfältigung eines Software-Produktes ist kein Ingenieurproblem.
- Das Wesen eines Software-Prototyps unterscheidet sich z.B. von einem Windkanal- oder einem Architekturmodell. Ein Software-Prototyp zeigt ausgewählte Eigenschaften des Zielproduktes im praktischen Einsatz. Er ist nicht nur eine Simulation des Zielproduktes.

Es gibt jedoch Ähnlichkeiten in der Anwendung der Prototypen:

Anwendung

- Prototypen werden verwendet, um relevante Anforderungen oder Entwicklungsprobleme zu klären.
- Prototypen dienen als Diskussionsbasis und helfen bei Entscheidungen.
- Prototypen werden für experimentelle Zwecke verwendet und um praktische Erfahrungen zu sammeln.

Prototypen-Modell

Das **Prototypen-Modell** unterstützt auf systematische Weise die frühzeitige Erstellung ablauffähiger Modelle (Prototypen) des zukünftigen Produkts, um die Umsetzung von Anforderungen und Entwürfen in Software zu demonstrieren und mit ihnen zu experimentieren. Eine solche Vorgehensweise wird als ***prototyping*** bezeichnet.

prototyping

Arten

Nach /Kieback et al. 92/ und /Budde et al. 92/ lassen sich vier Arten von Prototypen unterscheiden:

Demonstrationsprototyp

Ein **Demonstrationsprototyp** dient zur Auftragsakquisition. Er soll dem potentiellen Auftraggeber einen ersten Eindruck davon vermitteln, wie ein Produkt für das vorgesehene Anwendungsgebiet im Prinzip aussehen kann. Der Prototyp ist von einem tatsächlichen Produkt noch weit genug entfernt, so daß die Begrenzungen des Prototyps sichtbar werden.

In der Regel werden solche Prototypen schnell aufgebaut *(rapid prototyping)*, unter Vernachlässigung softwaretechnischer Standards. Sie werden nach der Erfüllung ihrer Aufgaben »weggeworfen«.

Prototyp i.e.S.

Ein **Prototyp im engeren Sinne** wird parallel zur Modellierung des Anwendungsbereichs erstellt, um spezifische Aspekte der Benutzungsschnittstelle oder Teile der Funktionalität zu veranschaulichen. Er trägt dazu bei, den Anwendungsbereich zu analysieren. Ein solcher Prototyp ist ein provisorisches, ablauffähiges Software-System.

Labormuster

Ein **Labormuster** dient dazu, konstruktionsbezogene Fragen und Alternativen zu beantworten. Es demonstriert die technische Umsetzbarkeit des Produktmodells. Endbenutzer nehmen an der Evaluation eines Labormusters im allgemeinen nicht teil. Die modellierten Aspekte beziehen sich meistens auf die Architektur oder die Funktionalität und sollten technisch mit dem späteren Produkt vergleichbar sein.

Pilotsystem

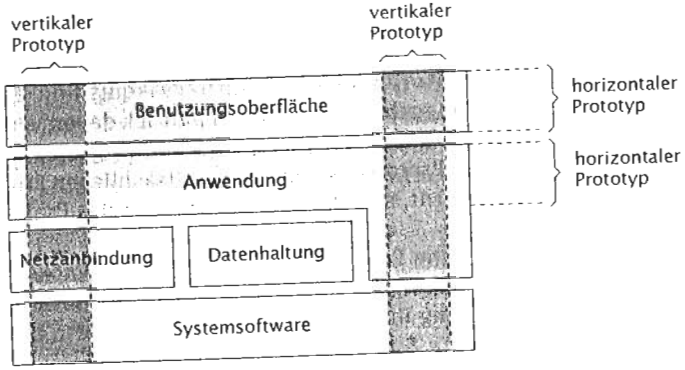
Ein **Pilotsystem** ist ein Prototyp, der nicht nur zur experimentellen Erprobung oder zur Veranschaulichung dient, sondern selbst der Kern des Produkts ist. Die Unterscheidung zwischen dem Prototyp und dem Produkt verschwindet. Ist ein gewisser Reifegrad erreicht, dann ist der Prototyp praktisch in Form eines Pilotsystems realisiert und wird in Zyklen weiterentwickelt.

Die Weiterentwicklungsstufen sollten sich exklusiv an den Benutzerprioritäten orientieren. Technisch gesehen benötigt ein Pilotsystem einen wesentlich sorgfältigeren Entwurf als die anderen Prototyp-Arten. Schließlich ist das Pilotsystem für die Benutzung in der Einsatzumgebung entworfen und nicht nur unter Laborbedingungen.

Selbst wenn der Einsatz auf eine einzelne Abteilung oder einen individuellen Arbeitsplatz beschränkt ist, so ist es doch unumgänglich, daß das System leicht zu bedienen und zuverlässig ist. Außerdem ist eine minimale Benutzerdokumentation erforderlich. Ein Pilotsystem hilft die organisatorische Integration des Produkts vorzubereiten, indem es dem Benutzer einen Vorgeschmack auf das System gibt.

Ein fertiges Software-Produkt besteht aus verschiedenen Komponenten bzw. Ebenen, von der Benutzungsschnittstelle bis zur systemnahen Ebene. Unter diesem Blickwinkel lassen sich horizontale und vertikale Prototypen unterscheiden /Floyd 84/ (Abb. 3.3-13).

Abb. 3.3-13: Horizontaler und vertikaler Prototyp



horizontal

Ein **horizontaler Prototyp** realisiert nur spezifische Ebenen des Systems, z.B. die Benutzungsschnittstelle oder funktionale Kernebenen wie Datenbanktransaktionen. Die betreffende Ebene wird möglichst vollständig realisiert. Bei der Benutzungsschnittstelle bedeutet dies, daß die gesamte Oberfläche zu sehen ist, aber die dahinterliegende Funktionalität fehlt.

vertikal

Ein **vertikaler Prototyp** implementiert ausgewählte Teile des Zielsystems vollständig durch alle Ebenen hindurch. Ist beispielsweise unklar, ob eine Echtzeitanforderung zu realisieren ist, dann wird die betreffende Echtzeitfunktion von der Bedienungsoberfläche bis zur systemnahen Ebene realisiert. Diese Technik ist dort geeignet, wo die Funktionalitäts- und Implementierungsoptionen noch offen sind. Dies ist in der Regel der Fall, wenn Pilotsysteme gebaut werden.

Zwischen einem Prototyp und den fertigen Software-Systemen kann es verschiedene Beziehungen geben:

- Prototypen dienen nur zur Klärung von Problemen.
  Prototypen werden entwickelt, um neue Erkenntnisse zu gewinnen, um Probleme zu klären und um Ideen zu überprüfen. Ziel ist es nicht, ein marktfähiges Software-System zu entwickeln. Daher ist die Beziehung zwischen Prototyp und fertigem Software-System nicht festgelegt. Solche Prototypen werden oft in Universitäten und Forschungsinstitutionen erstellt.
- Ein Prototyp ist Teil der Produktdefinition.
  Das Software-Produkt wird auf der Basis des akzeptierten Prototyps entwickelt. Der Prototyp dient dazu, das Produkt zu definieren. Er ist *nicht* Teil des Produktes selbst, sondern wird anschließend »weggeworfen«. Diese Prototypen werden so schnell wie möglich entwickelt, unter Vernachlässigung von Qualitätsanforderungen. Die technische Implementierung für den Prototyp und das Produkt können unterschiedlich sein. Ein Prototyp kann in LISP auf einer *Workstation*, das Produkt in C++ auf einem PC realisiert werden. Das Software-Produkt wird grundsätzlich neu aufgebaut, um geforderte Entwicklungs- und Qualitätsstandards zu erfüllen.
- Prototypen werden inkrementell weiterentwickelt, um ein marktfähiges Produkt zu erhalten.
  Durch den Einsatz moderner Software-Techniken ist ein gradueller Übergang vom Prototyp zum fertigen Produkt möglich.
  Beispielsweise kann durch den Einsatz von UIMS *(user interface management system)* eine Benutzungsoberfläche interaktiv erstellt und animiert werden. Ist der Auftraggeber mit der Oberfläche einverstanden, dann wird eine Schnittstelle zur Anwendung hin generiert. Die erstellte Oberfläche wird Teil des Endprodukts.
  Wesentlich ist, daß bei diesen Prototypen von vornherein auf die Einhaltung von geforderten Entwicklungs- und Qualitätsstandards geachtet wird.

Hauptkapitel I 3

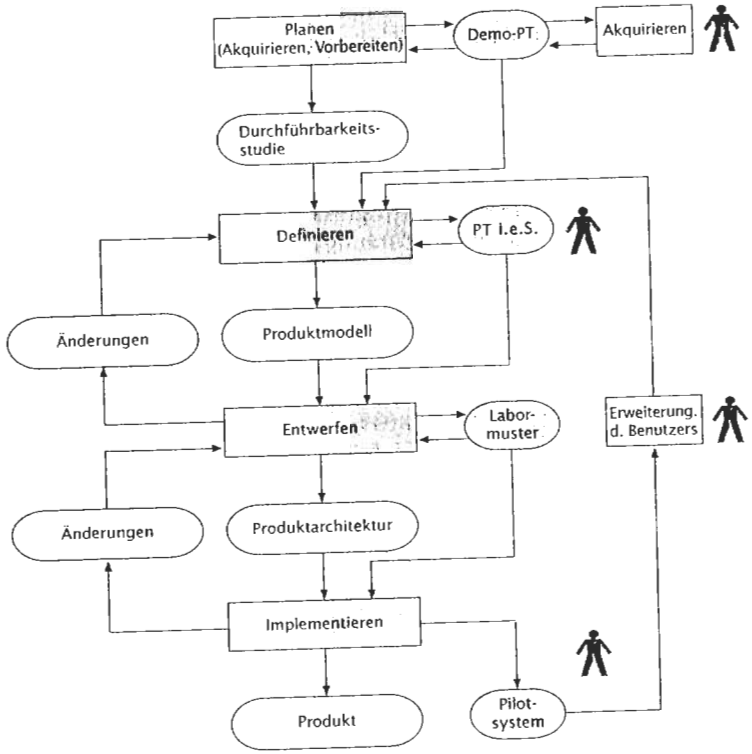
Wegen der verschiedenen Prototyp-Arten ist es wichtig, bei einer Software-Entwicklung jeweils anzugeben, von welchem Prototyp man spricht. Abb. 3.3-14 veranschaulicht das Prototypen-Modell.

Beispiele

Um einen Auftraggeber von einer computergestützten Seminar- und Kundenverwaltung zu überzeugen, kann ein **Demonstrationsprototyp** gezeigt werden, der es ermöglicht, ungeplante Anfragen an das System zu stellen wie:
»Mit welchen zehn Firmen wurden im letzten Jahr die meisten Umsätze erzielt?«, »Welche Seminarveranstaltungen erzielten den größten Umsatz im letzten halben Jahr?«
Ein solcher Prototyp kann mit einer relationalen Datenbank mit einer grafischen Oberfläche auf einem PC schnell erstellt werden.
Ein **Prototyp im engeren Sinne** kann die gesamte Benutzungsoberfläche der Seminarorganisation zeigen (horizontaler Prototyp). Ein solcher Prototyp kann mit einem UIMS erstellt und animiert werden. Alternativen können ebenfalls dargestellt werden. Wird eine Alternative akzeptiert, dann kann die Oberfläche in das Endprodukt integriert werden.

Soll geklärt werden, ob für die Datenhaltung – neben einer relationalen Datenbank – auch eine objektorientierte Datenbank bzgl. Modellierung und Leistung in Frage kommt, dann werden zwei **Labormuster** entwickelt und miteinander verglichen.
Sind die Anforderungen an die Kundenverwaltung innerhalb der Seminarorganisation bereits konsolidiert, z.B. durch einen Prototyp der Benutzungsoberfläche und ein Labormuster der Datenbankalternativen, dann kann ein **Pilotsystem** erstellt werden. Dies wird bei zwei Benutzern, die die Kunden verwalten, eingesetzt und schrittweise entsprechend den Benutzerwünschen erweitert.

Abb. 3.3-14: Das Prototypen-Modell (normierte Darstellung)



Legende: PT = Prototyp
Erläuterungen:
Die dunkelblauen Grafikteile kennzeichnen das Prototypen-Modell.
Die Gesamtgrafik zeigt eine Ergänzung der traditionellen Modelle um Prototypen.
Jeder Prototyp selbst muß definiert, entworfen und implementiert werden (hier nicht dargestellt).
Der dunkelblaue Teil der Aktivitäten und Produkte zeigt, daß Prototypen immer nur Ausschnitte oder Teilaspekte behandeln.

Bewertung

Das Prototypen-Modell besitzt folgende Vorteile: Vorteile

- ⊞ Reduzierung des Entwicklungsrisikos durch frühzeitigen Einsatz von Prototypen.
- ⊞ Prototypen können sinnvoll in andere Prozeß-Modelle integriert werden.
- ⊞ Prototypen können heute durch geeignete Werkzeuge schnell erstellt werden.
- ⊞ *Prototyping* verbessert die Planung von Software-Entwicklungen.
- ⊞ Labormuster fördern die Kreativität für Lösungsalternativen.
- ⊞ Starke Rückkopplung mit dem Endbenutzer und dem Auftraggeber.
- ⊞ Auch für die Entwicklung von Expertensystemen geeignet /Weitzel, Kerschberg 89/.

Dem stehen folgende Nachteile gegenüber: Nachteile

- ⊟ Höherer Entwicklungsaufwand, da Prototypen im allgemeinen zusätzlich erstellt werden.
- ⊟ Gefahr, daß ein »Wegwerf«-Prototyp aus Termingründen doch Teil des Endprodukts wird.
- ⊟ Verträge für die Software-Erstellung berücksichtigen noch nicht das Prototypen-Modell.
- ⊟ Prototypen werden oft als Ersatz für die fehlende Dokumentation angesehen.
- ⊟ Die Beschränkungen und Grenzen von Prototypen sind oft nicht bekannt.

Damit das Prototypen-Modell erfolgreich eingesetzt werden kann, sind nach /Kieback et al. 92/ folgende Voraussetzungen zu schaffen:

- Es muß ausreichendes Wissen über das Anwendungsgebiet vorhanden sein.
- Allein auf der Basis vorgegebener schriftlicher Dokumente kann kein Prototyp erstellt werden. Die Entwickler müssen Zugang zu den Benutzern haben.
- Die Endbenutzer müssen am *Prototyping*-Prozeß beteiligt werden.
- Die Benutzerbeteiligung ersetzt nicht die kreativen Ideen und Lösungsvorstellungen der Entwickler.
- Alle an der Entwicklung beteiligten Personengruppen müssen in direktem Kontakt stehen.
- Prototypen müssen im richtigen Umfang dokumentiert werden.
- Die Vorgehensweise hängt von der untersuchten Fragestellung ab.
- Geeignete Werkzeuge müssen verfügbar sein.

Das Motto des Prototypen-Modells lautet: *»Redo until Right«.*

3.3.4 Das evolutionäre/inkrementelle Modell

Sowohl beim Wasserfallmodell als auch beim V-Modell ist es das Ziel, ein vollständiges Software-Produkt zu entwickeln, das den definierten Anforderungen entspricht. Es wird bei diesen Modellen – zumindest implizit – davon ausgegangen, daß in der Definitionsphase alle Anforderungen des Auftraggebers ermittelt werden. Unklare Anforderungen können durch Prototypen noch geklärt werden. Anschließend wird das Produkt in der vollen Breite entwickelt. Es werden ein vollständiger Entwurf und eine vollständige Implementierung durchgeführt. Erst dann steht dem Auftraggeber das Produkt zur Verfügung.

Diese Entwicklung in voller Breite führt dazu, daß die Entwicklungsdauer oft mehrere Jahre beträgt und der Auftraggeber entsprechend lange auf sein Produkt warten muß. Neben diesem Nachteil hat sich in der Praxis herausgestellt, daß der Auftraggeber seine Produktanforderungen oft nicht vollständig formulieren kann. Manche Wünsche ergeben sich auch erst beim Produkteinsatz. Ähnlich ist die Situation, wenn ein Produkt für den anonymen Markt entwickelt werden soll. Diese Probleme bzw. Nachteile haben zu dem Modell der evolutionären Entwicklung geführt.

evolutionär

Ausgangspunkt bei dem **evolutionären Modell** sind die Kern- oder Mußanforderungen des Auftraggebers. Diese Anforderungen definieren einen Produktkern. Nur dieser Produktkern wird anschließend entworfen und implementiert. Das Kernsystem, auch Nullversion genannt, wird an den Auftraggeber ausgeliefert.

Der Auftraggeber sammelt Erfahrungen mit dieser Nullversion und ermittelt daraus seine Produktanforderungen für eine erweiterte Version. Die Nullversion wird anschließend um die neuen Anforderungen ergänzt. Die neue Produktversion wird eingesetzt; anhand der gewonnenen Erfahrungen werden neue Anforderungen aufgestellt.

Eine solche evolutionäre Entwicklung besitzt folgende Charakteristika (siehe auch /Boehm 88, S. 122/):

- Das Software-Produkt wird allmählich und stufenweise entwickelt, gesteuert durch die Erfahrungen, die der Auftraggeber und die Benutzer mit dem Produkt machen.
- Pflegeaktivitäten werden ebenfalls als Erstellung einer neuen Version betrachtet.
- Gut geeignet, wenn der Auftraggeber seine Anforderungen noch nicht vollständig überblickt: *»I can't tell you what I want, but I'll know it when I see it«.*
- Die Entwicklung ist code-getrieben, d.h. man konzentriert sich jeweils auf lauffähige Teilprodukte.

Die evolutionäre Entwicklung wird auch als Versionen-Entwicklung *(versioning)* bezeichnet. Oft wird auch die Entwicklung eines Pilot-

versioning

systems mit der evolutionären Entwicklung gleichgesetzt. Teilweise spricht man auch von evolutionärem *prototyping* (siehe auch /Budde et al. 92/). Abb. 3.3-15 zeigt die Vorgehensweise beim evolutionären Modell.

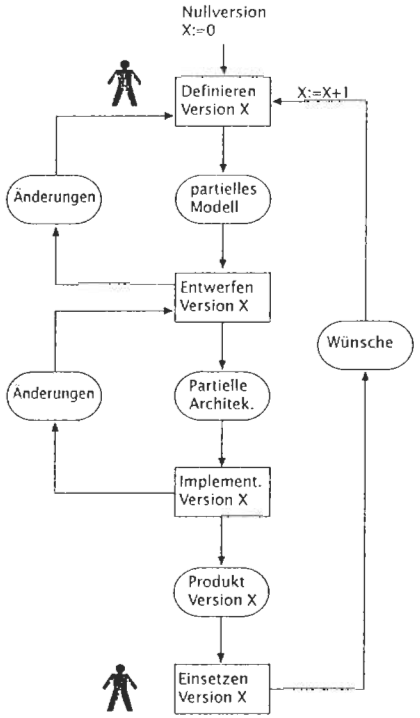


Abb. 3.3-15: Das evolutionäre Modell (normierte Darstellung)

Beispiel

Der Auftraggeber benötigt für seine Seminarorganisation kurzfristig eine Kundenverwaltung. Da er den Bereich der Dozenten- und Seminarverwaltung noch nicht überblickt, wird dieser Verwaltungsteil zurückgestellt.
Nach Einführung der Kundenverwaltung wird in der nächsten Version die Kundenverwaltung um eine Seminarverwaltung ergänzt. Anschließend erfolgt noch eine Erweiterung um eine Dozentenverwaltung.

Bewertung

Vorteile

Das evolutionäre Modell hat folgende Vorteile:

- Der Auftraggeber erhält in kürzeren Zeitabständen einsatzfähige Produkte, z.B. im Halbjahresrhythmus.
- Es kann mit dem Prototypen-Modell kombiniert werden, insbesondere ist eine Pilotsystem-Entwicklung ähnlich wie eine evolutionäre Entwicklung.

- ⊞ Der frühzeitige Einsatz einer eingeschränkten Produktversion erlaubt es, die Auswirkungen des Produkteinsatzes auf die Arbeitsabläufe zu studieren und diese Erfahrungen in die nächste Version einzubringen.
- ⊞ Ein Produkt wird in einer Anzahl kleiner Arbeitsschritte überschaubarer Größe erstellt. Dadurch ist es möglich, die Richtung der Entwicklung Schritt für Schritt zu korrigieren oder neu zu definieren.
- ⊞ Die Entwicklung ist nicht nur auf einen einzigen Endabgabetermin ausgerichtet – der meist mehrere Jahre in der Zukunft liegt – sondern es gibt einsatzfähige Zwischenergebnisse.

Nachteile

Dem stehen folgende Nachteile gegenüber:

- ⊟ Es besteht die Gefahr, daß in den nachfolgenden Versionen die komplette Systemarchitektur überarbeitet werden muß, weil bei der Nullversion Kernanforderungen übersehen wurden.
- ⊟ Es besteht die Gefahr, daß die Nullversion nicht flexibel genug ist, um sich an ungeplante Evolutionspfade anzupassen.

inkrementell

Diese gravierenden Nachteile werden bei dem **inkrementellen Modell** vermieden. Bei der inkrementellen Entwicklung werden die Anforderungen an das zu entwickelnde Produkt möglichst vollständig erfaßt und modelliert. Analog zur evolutionären Entwicklung wird dann jedoch nur ein Teil der Anforderungen entworfen und implementiert. Der Auftraggeber erhält in kurzer Zeit ein einsatzfähiges System. Anschließend wird die nächste Ausbaustufe realisiert, wobei Erfahrungen des Auftraggebers mit der laufenden Version berücksichtigt werden.

Da bei der inkrementellen Entwicklung bereits ein vollständiges Analysemodell entwickelt wurde, ist sichergestellt, daß die inkrementellen Erweiterungen zu dem bisherigen System passen, z.B. sind die Schnittstellen bekannt. Diese Sicherheit ist bei der evolutionären Entwicklung nicht vorhanden.

Beispiel

Für die Seminarorganisation wird ein vollständiges Produktmodell mit den zwei Subsystemen Kundenverwaltung und Seminarverwaltung erstellt.
Der Auftraggeber entscheidet, daß er die Kundenverwaltung zuerst benötigt. Daher wird zunächst dieses Subsystem realisiert. Anschließend erfolgt in der nächsten Version die Erweiterung um das Subsystem Seminarverwaltung. Zusätzlich werden einige Attribute und Operationen der Kundenverwaltung ergänzt, die sich aus dem Einsatz der Kundenverwaltung ergeben haben.

Die hier vorgenommene Unterscheidung zwischen evolutionärem und inkrementellen Modell ist in der Literatur so nicht zu finden. Abb. 3.3-16 veranschaulicht das inkrementelle Modell.

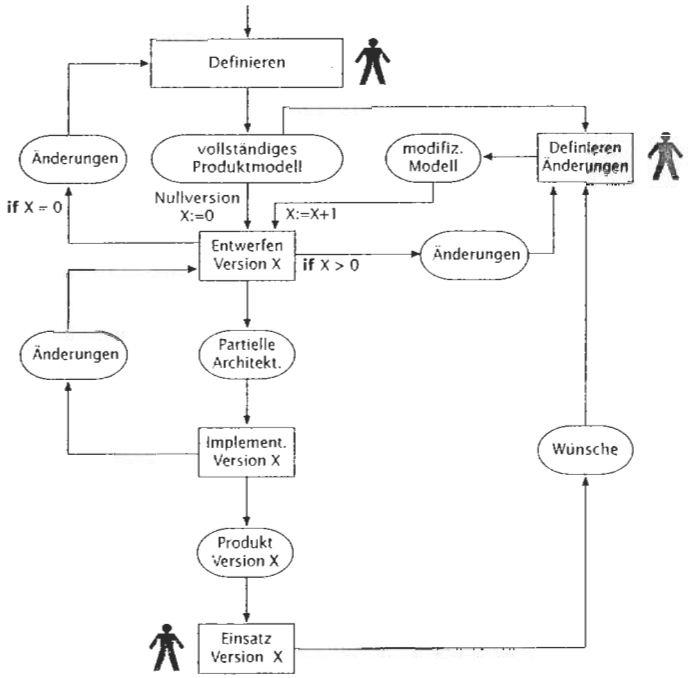


Abb. 3.3-16: Das inkrementelle Modell (normierte Darstellung)

3.3.5 Das objektorientierte Modell

Ein wesentlicher Vorteil einer objektorientierten Software-Entwicklung besteht darin, daß die Wiederverwendung von Klassen und Klassenhierarchien durch Komposition, Vererbung und Polymorphie unterstützt werden. Die Wiederverwendung kann auf verschiedenen Ebenen erfolgen. Es können

Ebenen

- OOA-Modelle (Subsysteme, Klassen, Klassenhierarchien),
- technische Entwürfe (OOD-Modelle) und
- implementierte Klassen und Klassenbibliotheken wiederverwendet werden.

Neben den Ebenen der Wiederverwendung können auch die Wiederverwendungsgebiete unterschieden werden:

Gebiete

- Es können anwendungs- bzw. branchenspezifische Klassen und Subsysteme wiederverwendet werden, z.B. Finanzanalysen.
- Es können Klassen wiederverwendet werden, die die Anbindung einer Anwendung an die Umgebung ermöglichen, z.B. Oberflächenklassen für die Benutzungsschnittstelle, Protokollklassen für die Netzwerkkommunikation, Klassen für die Datenhaltungsansteuerung.

- Es können Klassen wiederverwendet werden, die eine Anbindung an die Systemsoftware ermöglichen, z.B. Basisklassen.

Außerdem kann nach der Herkunft der Klassen unterschieden werden:

Herkunft
- Eigene Klassen aus früheren Entwicklungen.
- Auf dem Markt eingekaufte Klassenbibliotheken und OOA-Konzepte.

Beide können in einem Wiederverwendbarkeitsarchiv abgelegt werden. Abb. 3.3-17 zeigt die Wiederverwendbarkeitsklassifikation in Form eines Quaders.

Für das **objektorientierte Modell** ist es wesentlich, wann neue wiederverwendbare Klassen und Subsysteme in das eigene Wiederverwendbarkeitsarchiv eingeordnet werden.

Zeitpunkte Es lassen sich folgende Zeitpunkte unterscheiden:
- Bereits während der laufenden Entwicklung.
- Erst am Ende einer abgeschlossenen Entwicklung.
- Erst nachdem ein Team nachträglich abgeschlossene Entwicklungen analysiert und wiederverwendbare Komponenten identifiziert, überarbeitet sowie verallgemeinert hat.

Diese vielfältigen Aspekte der Wiederverwendbarkeit müssen bei dem Prozeß-Modell einer objektorientierten Entwicklung berücksichtigt werden.

Kapitel IV 1.2 Während alle bisher behandelten Modelle *top-down* orientiert waren, kommt durch die Berücksichtigung der Wiederverwendbarkeit ein starker *bottom-up*-Aspekt in die Vorgehensweise.

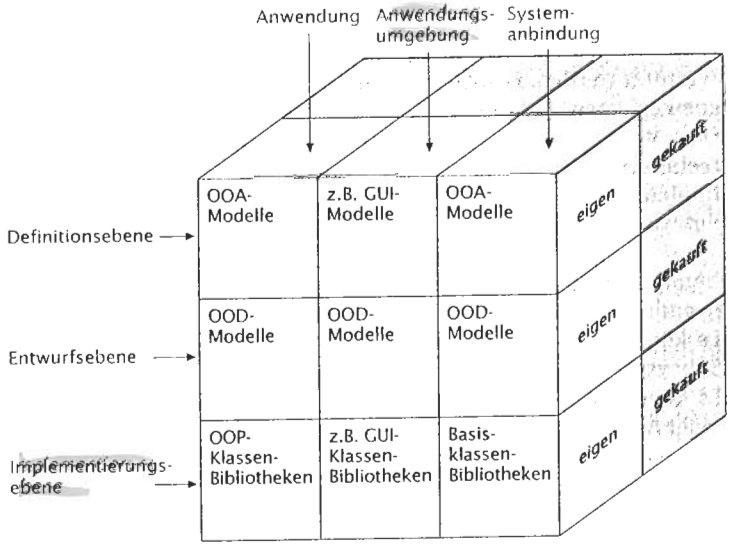


Abb. 3.3-17: Quader der Wiederverwendbarkeitsklassifikation

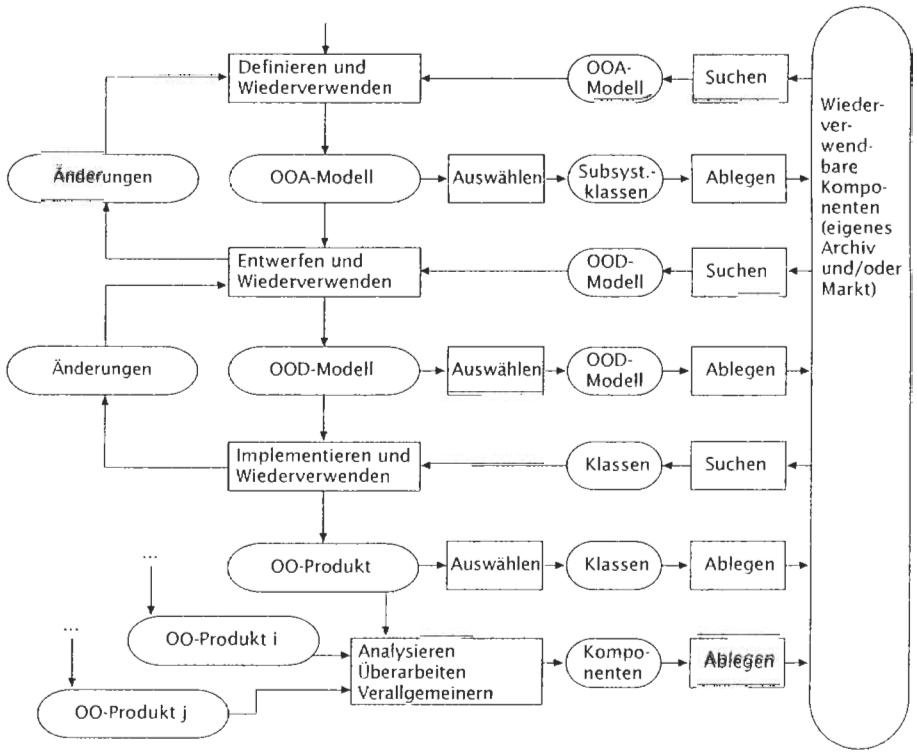


Abb. 3.3-18: Das objektorientierte Modell (normierte Darstellung)

Bei jeder Aktivität muß überlegt werden, ob Komponenten wiederverwendet werden können. In Abhängigkeit von den vorhandenen oder im Markt verfügbaren Komponenten wird die weitere Entwicklungsrichtung beeinflußt. Abb. 3.3-18 veranschaulicht dieses Prozeß-Modell.

Beispiel Der Auftraggeber erzählt, daß er eine Kunden- und eine Seminarverwaltung für seine Firma Teachware benötigt. Nach dem ersten Aufstellen eines OOA-Modells wird geprüft, ob es geeignete wiederverwendbare OOA-Modelle gibt. Es wird festgestellt, daß es sowohl im Markt ein Kundenverwaltungs-Subsystem gibt als auch in dem eigenen Archiv eine Kundenverwaltung im Rahmen eines Personalverwaltungssystems. Da das Subsystem des Personalverwaltungssystems es auch noch ermöglicht, die Dozenten und Mitarbeiter der Firma Teachware zu verwalten, wird dieses Subsystem wiederverwendet. Obwohl einige Leistungen dieses Subsystems nur mit niedriger Priorität benötigt werden, wird entschieden, diese von Anfang an mitzubenutzen.
Da dieses Subsystem jedoch über keine Firmenverwaltung verfügt, muß es noch um eine entsprechende Klasse und Assoziation erwei-

tert werden. Da außerdem die Seminarverwaltung fehlt, wird ein weiteres Subsystem hinzugefügt. Für die Benutzungsoberfläche wird eine GUI-Bibliothek verwendet, für die Ansteuerung einer relationalen Datenbank ebenfalls eine entsprechende Klassenbibliothek.

Das objektorientierte Modell läßt sich folgendermaßen charakterisieren:

- Tendenziell wird in voller Breite entwickelt, da der Wiederverwendungsgesichtspunkt einen Blick »über den Tellerrand« erfordert.
- Der Fokus verschiebt sich weg von der Eigenentwicklung und hin zur Wiederverwendung.
- Gut kombinierbar mit dem evolutionären und inkrementellen Modell sowie dem Prototypen-Modell, da Erweiterungen und Änderungen mit der objektorientierten Technik gut durchführbar sind.

Bewertung

Vorteile

Dieses Modell besitzt folgende Vorteile:

- ⊞ Verbesserung der Produktivität und Qualität.
- ⊞ Konzentration auf die eigenen Stärken, Rest zukaufen, d.h. weitgehende Nutzung von Halbfabrikaten.

Nachteile

Dem stehen folgende Nachteile gegenüber:

- ⊟ Nur voll nutzbar, wenn objektorientierte Methoden eingesetzt werden, d.h. gebunden an die objektorientierte Technik.
- ⊟ Geeignete Infrastruktur (Wiederverwendungsarchiv) muß vorhanden sein.
- ⊟ Firmenkultur der Wiederverwendung muß aufgebaut sein (Wiederverwendung sowie Erstellung wiederverwendbarer Komponenten muß belohnt werden).
- ⊟ Technische Probleme müssen überwunden werden (z.B. Inkompatibilitäten von Klassenbibliotheken).

Hauptkapitel IV 3

In der Literatur wird dieses Modell erst allmählich behandelt. /Henderson-Sellers, Edwards 90, 93/ beschreiben ein Fontänen-Modell *(fountain model)*, /Pittman 93/ stellt ein detailliertes Aktivitätenmodell vor, in /Davis 94/ wird gezeigt, wie der Wiederverwendbarkeitsgedanke in ein Unternehmen eingeführt werden kann. /Marty 94/ beschreibt eine Organisationsstruktur für die objektorientierte Software-Entwicklung, /Lausecker 93/ die Integration der Wiederverwendung in den Software-Entwicklungsprozeß.

3.3.6 Das nebenläufige Modell

Entscheidend für den Erfolg einer Software-Entwicklung ist oft die termingerechte Fertigstellung *(time-to-market)*. Ein Ansatz, um dies zu erreichen, ist die evolutionäre/inkrementelle Vorgehensweise mit dem Ziel, in kurzer Zeit zunächst nur ein minimales Kernsystem zu erstellen.

nebenläufiges Modell

Einen anderen Ansatz stellt das **nebenläufige Prozeß-Modell** dar *(concurrent engineering, simultaneous engineering, parallel engineering)* (/Eiff 91/, /Spectrum 91/). Dieses Modell stammt aus der Fertigungsindustrie, bei der es früher eine starke Trennung zwischen der Entwicklung eines Produktes und der Fertigung eines Produktes gab. Erst nach der Entwicklung eines Prototyps erfolgte eine Überprüfung auf Fertigungstauglichkeit, Qualität und Wartbarkeit. Das führte oft zu einer Überarbeitung des Produktes mit entsprechenden Zeitverzögerungen.

Bei der nebenläufigen Entwicklung sind alle betroffenen Entwicklungsabteilungen einschließlich Fertigung, Marketing und Vertrieb in einem Team vereint. Dadurch sollen von vornherein alle relevanten Gesichtspunkte berücksichtigt werden, um teure und langwierige Überarbeitungen zu sparen. Außerdem soll soviel wie möglich parallel ablaufen.

Die Parallelisierung birgt aber auch Risiken. Wenn bereits aufgrund erster Entwürfe eines Entwicklers Werkzeuge für die Teilefertigung konstruiert und gebaut werden, dann führen spätere konstruktive Änderungen an diesem Teil zur Verschrottung dieser Werkzeuge. Nebenläufiges Entwickeln erfordert daher ein ständiges Abwägen, inwieweit es vertretbar erscheint, finanzielle Risiken durch vorzeitige Parallelisierung von Arbeitsfolgen einzugehen.

Überträgt man dieses Konzept auf die Software-Entwicklung, dann bedeutet dies, möglichst viele Aktivitäten zu parallelisieren oder stark zu überlappen (Abb. 3.3-19).

Beispiel

In der Definitionsphase werden nach dem Erstellen des Pflichtenheftes parallel das OOA-Modell, die Benutzungsoberfläche und das Benutzerhandbuch entwickelt.
Ist ein Teil des OOA-Modells erstellt, beginnt bereits der Entwurf. Ausgehend vom ersten Entwurf wird bereits mit der Implementierung begonnen.

Das nebenläufige Modell besitzt folgende Charakteristika:

- Durch organisatorische und technische Maßnahmen wird versucht, die einzelnen Aktivitäten im Rahmen des prinzipiell sequentiell angelegten Entwicklungsprozesses zu parallelisieren.
- Das auf Problemlösung gerichtete Zusammenarbeiten der betreffenden Personengruppen wird gefördert.
- Zeitverzögerungen werden reduziert durch
  - die teilweise Parallelisierung vorwiegend sequentiell organisierter Arbeiten.
  - die Minimierung des Ausprobierens *(trial and error)* und Begrenzung des Improvisierens (Motto: *»right the first time«* im Gegensatz zum Prototypen-Modell: *»redo until right«*).
  - die Reduktion der Wartezeiten zwischen arbeitsorganisatorisch verbundenen Aktivitäten.

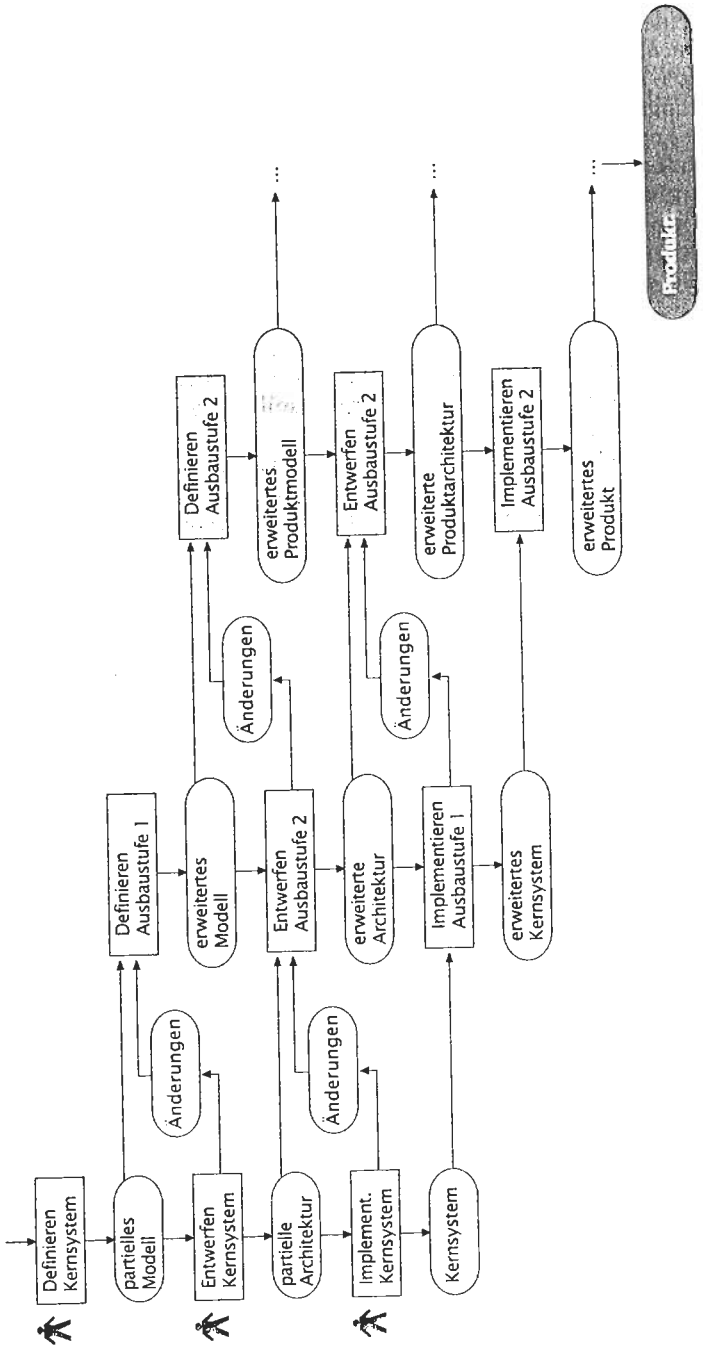


Abb. 3.3-19: Das nebenläufige Modell (normierte Darstellung)

- Alle Erfahrungen der betroffenen Personengruppen werden frühzeitig zusammengebracht.
- Im Gegensatz zum evolutionären/inkrementellen Modell ist es hier von vornherein das Ziel, das vollständige Produkt auszuliefern.

Bewertung

Folgende Vorteile kennzeichnen das nebenläufige Modell: Vorteile

⊞ Durch Beteiligung aller betroffenen Personengruppen frühes Erkennen und Eliminieren von Problemen.

⊞ Optimale Zeitausnutzung.

Als Nachteile ergeben sich: Nachteile

⊟ Es ist fraglich, ob es wegen der speziellen Charakteristika der Software möglich ist, das Ziel *»right the first time«* zu erreichen.

⊟ Risiko, daß die grundlegenden und kritischen Entscheidungen zu spät getroffen werden und dadurch Iterationen nötig werden.

⊟ Hoher Planungs- und Personalaufwand, um Fehler zu vermeiden bzw. Probleme frühzeitig zu antizipieren.

In der Software-Technik-Literatur wird dieses Modell noch nicht diskutiert.

3.3.7 Das Spiralmodell

Bei dem von /Boehm 86,88/ beschriebenen **Spiralmodell** handelt es sich um ein Metamodell. Abschnitt IV 1.1.1

Für jedes Teilprodukt und für jede Verfeinerungsebene sind vier zyklische Schritte zu durchlaufen.

Schritt 1:

- Identifikation der Ziele des Teilprodukts, das erstellt werden soll (Leistung, Funktionalität, Anpaßbarkeit usw.).
- Alternative Möglichkeiten, um das Teilprodukt zu realisieren (Entwurf A, Entwurf B, Wiederverwendung, Kauf usw.).
- Randbedingungen, die bei den verschiedenen Alternativen zu beachten sind (Kosten, Zeit, Schnittstellen usw.).

Schritt 2:

- Evaluierung der Alternativen unter Berücksichtigung der Ziele und Randbedingungen.
- Zeigt die Evaluierung, daß es Risiken gibt, dann ist eine kosteneffektive Strategie zu entwickeln, um die Risiken zu überwinden. Dies kann z.B. durch Prototypen, Simulationen, Benutzerbefragungen usw. geschehen.

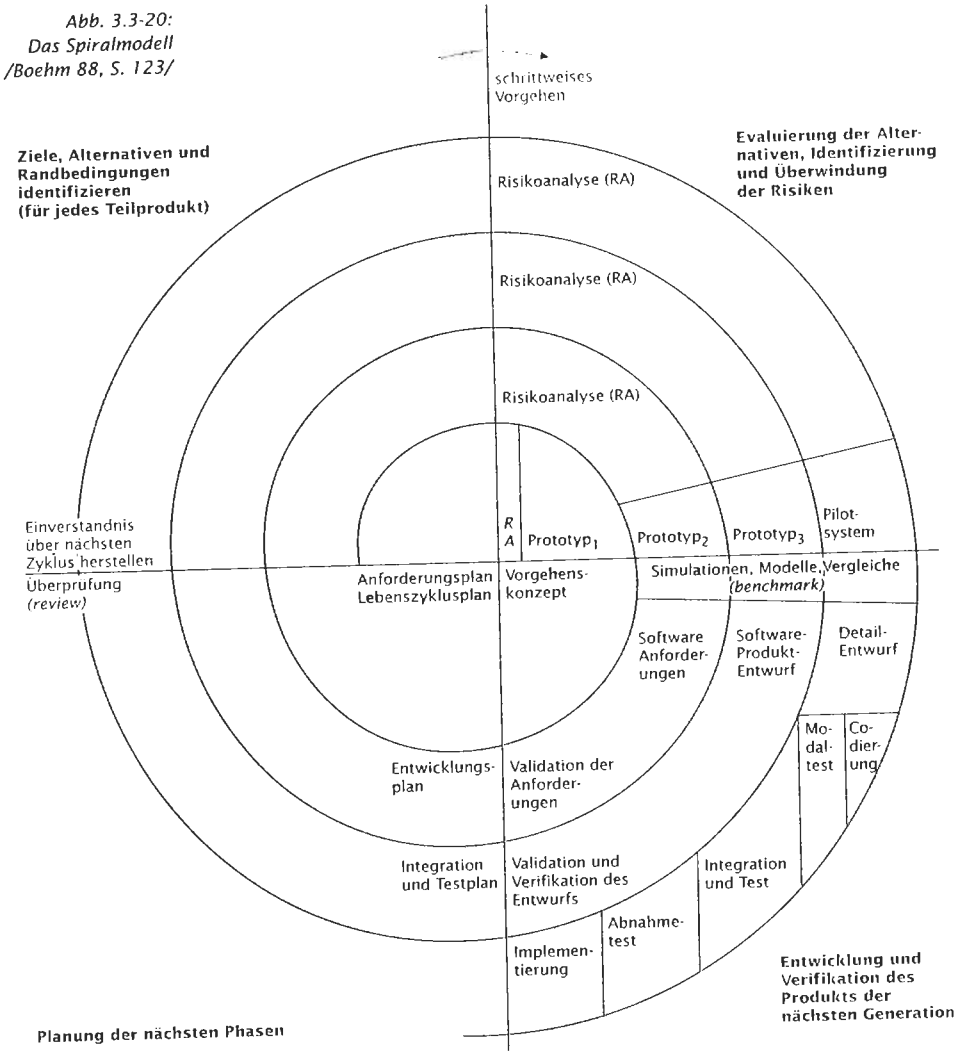
Schritt 3:

- In Abhängigkeit von den verbleibenden Risiken wird das Prozeß-Modell für diesen Schritt festgelegt, z.B. evolutionäres Modell, Prototypen-Modell oder Wasserfall-Modell.
- Es kann auch eine Kombination verschiedener Modelle vorgenommen werden, wenn dadurch das Risiko minimiert wird.

Schritt 4:

- Planung des nächsten Zyklus einschließlich der benötigten Ressourcen. Dies beinhaltet auch eine mögliche Aufteilung eines Produktes in Komponenten, die dann unabhängig weiterentwickelt werden.
- Überprüfung *(review)* der Schritte 1 bis 3 einschließlich der Planung für den nächsten Zyklus durch die betroffenen Personengruppen oder Organisationen.
- Einverständnis *(commitment)* über den nächsten Zyklus herstellen.

Abb. 3.3-20: Das Spiralmodell /Boehm 88, S. 123/



Das mehrfache Durchlaufen dieser Schritte wird als Spirale dargestellt (Abb. 3.3-20).

Die Fläche der Spirale repräsentiert die akkumulierten Kosten, die bei der bisherigen Entwicklung angefallen sind. Der Winkel der Spirale zeigt den Entwicklungsfortschritt des jeweiligen Zyklus an. Abb. 3.3-21 zeigt das Spiralmodell in der normierten Darstellung.

Folgende Merkmale charakterisieren das Spiralmodell:

- Risikogetriebenes Modell, bei dem oberstes Ziel die Minimierung des Risikos ist.
- Jede Spirale stellt einen iterativen Zyklus durch dieselben Schritte dar.
- Die Ziele für jeden Zyklus werden aus den Ergebnissen des letzten Zyklus abgeleitet.
- Separate Spiralzyklen können für verschiedene Software-Komponenten durchgeführt werden.
- Keine Trennung in Entwicklung und Wartung.

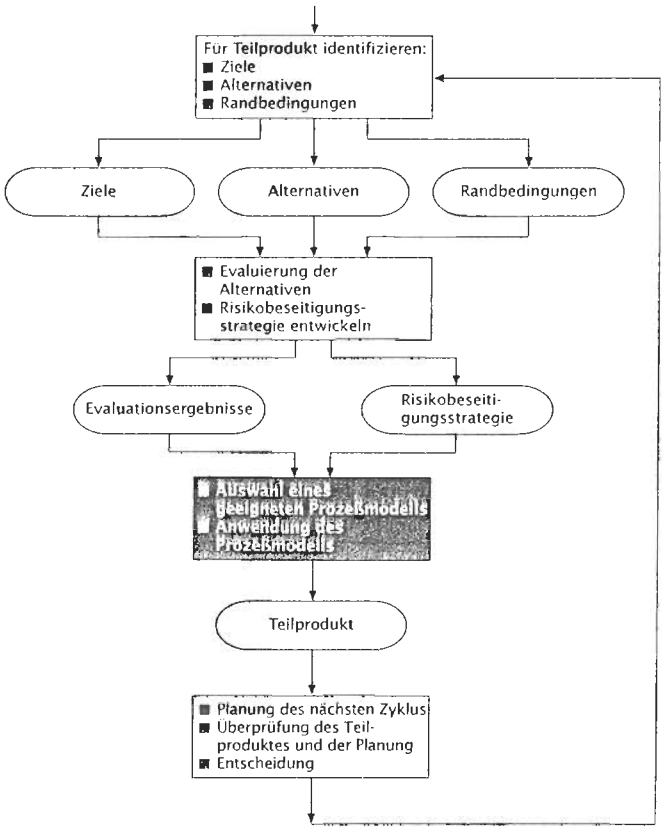


Abb. 3.3-21: Das Spiralmodell (normierte Darstellung)

- Ziel: Beginne im Kleinen, halte die Spirale so eng wie möglich und erreiche so die Entwicklungsziele mit minimalen Kosten.
- Bei der Zielbestimmung werden auch Qualitätsziele aufgeführt.
- Für jede Aktivität und jeden Ressourcenverbrauch wird gefragt »Wieviel ist genug?«. Dadurch wird ein »*Overengineering*« vermieden.

Beispiel

In tabellarischer Form werden die zwei ersten Zyklen für die Fallstudie Seminarorganisation angegeben (Tab. 3.3-3 und Tab. 3.3-4).

Tab. 3.3-3: 1. Zyklus des Spiralmodells (Beispiel)

Ziele	Verwaltung von Seminaren, Kunden, Firmen und Dozenten
Alternativen	**a** Kaufen **b** Kaufen und Anpassen **c** Individuell entwickeln
Randbedingungen	■ Kunden müssen in einem 1/2 Jahr verwaltet werden können. ■ Preis muß unter DM 50.000,- liegen. ■ Schnittstelle zur vorhandenen Buchhaltung muß ansteuerbar sein.
Evaluierung der Alternativen (Risiken)	**a** kein geeignetes Produkt im Markt, Anforderungen nur zum Teil erfüllt. **b** kein geeignetes Produkt im Markt, nicht genügend anpaßbar. **c** Kosten- und Zeitrahmen werden gesprengt.
Strategie zur Risikobeseitigung	■ Pflichtenheft erstellen **a** + **b** Marktanalyse und Testinstallation **c** Aufwandsschätzung
Auswahl Prozeßmodell	Nebenläufiges Modell
Anwendung des Modells	Aktivitäten parallel durchführen
Ergebnisse	**a** kein geeignetes Produkt im Markt. **b** Datenbankprodukte erfordern eine zu hohe Anpassung. **c** Aufwandsschätzung: 6 Mitarbeitermonate, Kosten und Zeitrahmen erscheinen einhaltbar.
Planung nächster Zyklus	Detaillierte Überprüfung der Alternative »Neues Produkt entwickeln«
Überprüfung	Die Ergebnisse **a**, **b**, **c** sowie die Planung werden überprüft.
Entscheidung	Durchführung der Planung für die nächste Phase.

Bewertung

Vorteile

Folgende Vorteile sprechen für das Spiralmodell:

- ⊞ In periodischen Intervallen Überprüfung und u.U. erneute Festlegung des Prozeßablaufs in Abhängigkeit von den Risiken.
- ⊞ Ein Prozeß-Modell wird nicht für die gesamte Entwicklung festgelegt.
- ⊞ Integration anderer Prozeß-Modelle als Spezialfälle.
- ⊞ Fehler und ungeeignete Alternativen werden frühzeitig eliminiert.

Tab. 3.3-4: 2. Zyklus des Spiralmodells (Beispiel)

Ziele	Überprüfung der Alternative »Neues Produkt entwickeln«
Alternativen	**a** Eigenentwicklung **b** Auftrag an Software-Haus
Randbedingungen	■ Zeit und Kosten einhalten. ■ Schnittstelle zur Buchhaltung sicherstellen. ■ Wartung und Weiterentwicklung sicherstellen.
Evaluierung der Alternativen (Risiken)	**a** + **b** Zeit und Kosten werden nicht eingehalten. **b** Wartung und Weiterentwicklung nicht sichergestellt. **b** Anforderungen werden nicht eingehalten.
Strategie zur Risikobeseitigung	**a** + **b** OOA-Modell erstellen, erneute Aufwandsschätzung, Wiederverwendbarkeitspotential prüfen, Angebote einholen. **b** Vertragliche Absicherung der Wartung und Weiterentwicklung (Übergabe des Quellcodes, Abnahme Systementwurf und Quellprogramm). **b** Oberflächenprototyp mit UIMS erstellen und Benutzerevaluation durchführen.
Auswahl Prozeßmodell	Nebenläufiges Modell und Prototypenmodell.
Anwendung des Modells	Zuerst OOA-Modell erstellen, dann alle anderen Aktivitäten nebenläufig, Oberfläche als Prototyp i.e.S.
Ergebnisse	OOA-Modell; Schätzung bestätigt Zeit-und Kostenrahmen; Wiederverwendung einer Kundenverwaltung möglich (im eigenen Hause vorhanden); Oberflächenprototyp nach mehreren Änderungen evaluiert, Angebote liegen im Zeit- und Kostenrahmen, Vertragsgestaltung macht Probleme.
Planung nächster Zyklus	Wegen Wiederverwendbarkeit Entwicklung im eigenen Hause.
Überprüfung	Ergebnisse und Planung überprüfen.
Entscheidung	Eigenentwicklung durchführen.

- ⊞ Flexibles Modell.
- ⊞ Eine Entwicklung kann wesentlich leichter umdirigiert werden, wenn die bisherigen Erkenntnisse dies erfordern.
- ⊞ Unterstützt die Wiederverwendung von Software durch die Betrachtung von Alternativen.

Nachteile

Dem stehen folgende Nachteile gegenüber:

- ⊟ Hoher Managementaufwand, da oft neue Entscheidungen über den weiteren Prozeßablauf getroffen werden müssen.
- ⊟ Für kleine und mittlere Projekte weniger gut geeignet.
- ⊟ Wissen über das Identifizieren und Managen von Risiken noch nicht weit genug verbreitet.

Demonstrationsprototyp Soll einen ersten Eindruck von einem möglichen Produkt vermitteln; noch weit vom realen Produkt entfernt, im allgemeinen ein Wegwerf-Prototyp (→Prototypen-Modell).

Evolutionäres Modell Stufenweise Entwicklung eines Produktes; ausgehend von einem eingeschränkten Produktmodell wird zunächst ein Kernsystem (Nullversion) erstellt. Die Weiterentwicklung erfolgt anhand der gemachten Erfahrungen mit dem bisher entwickelten System (→inkrementelles Modell).

Horizontaler Prototyp Realisiert möglichst vollständig eine Systemebene, z.B. die Benutzungsoberfläche (→Prototyp im engeren Sinne) (→vertikaler Prototyp).

Inkrementelles Modell Stufenweise Entwicklung eines Produktes; ausgehend von einem vollständigen Produktmodell wird zunächst ein Kernsystem (Nullversion) entwickelt. Anhand des Produktmodells und der gemachten Erfahrungen mit dem bisher entwickelten System erfolgt die Weiterentwicklung (→ evolutionäres Modell).

Labormuster Provisorisches Software-System, das während des Produktentwurfs erstellt wird, um konstruktionsbezogene Fragen zu klären (→Prototypen-Modell).

Nebenläufiges Modell Durch organisatorische und technische Maßnahmen wird versucht, die einzelnen Entwicklungsaktivitäten zu parallelisieren, wobei alle betroffenen Personengruppen in einem Team zusammenarbeiten *(concurrent engineering, simultaneous engineering, parallel engineering)*.

Objektorientiertes Modell Bei jeder Aktivität der objektorientierten Software-Entwicklung wird geprüft, ob und in welchem Umfang bereits vorhandene oder auf dem Markt verfügbare Software-Komponenten wiederverwendet werden können.

Pilotsystem Software-System, das nach und nach zum Kernsystem des zu entwickelnden Produkts wird (→Prototypen-Modell).

Prototypen-Modell Frühzeitige Erstellung ablauffähiger Modelle (Prototypen) des zukünftigen Produkts zur Überprüfung von Ideen oder zum Experimentieren.

Prototyp im engeren Sinne (i.e.S.) Provisorisches Software-System, das während der Produktdefinition erstellt wird, um Anforderungsfragen zu klären oder Anforderungen zu veranschaulichen (→Prototypen-Modell).

prototyping →Prototypen-Modell.

Rolle Beschreibt die notwendigen Erfahrungen, Kenntnisse und Fähigkeiten, über die ein Mitarbeiter verfügen muß, um eine bestimmte Aktivität durchzuführen.

Spiralmodell Eine Softwareentwicklung durchläuft mehrmals einen aus vier Schritten bestehenden Zyklus mit dem Ziel, frühzeitig Risiken zu erkennen und zu vermeiden. Pro Zyklus kann dann ein Prozeß-Modell oder eine Kombination von Prozeß-Modellen zur Erstellung eines Teilprodukts oder einer Ebene eines Teilprodukts festgelegt werden.

Vertikaler Prototyp Realisiert ausgewählte Teile des Zielsystems vollständig über alle Ebenen hinweg z.B. eine Echtzeitfunktion (→Labormuster), (→horizontaler Prototyp).

V-Modell Ein um die Aktivitäten Verifikation und Validation erweitertes → Wasserfall-Modell, ursprünglich für eingebettete, militärische Entwicklungen vorgesehen. Inzwischen gibt es in Deutschland eine Weiterentwicklung, die auch andere Anwendungsklassen abdeckt (Vorgehensmodell der Bundeswehr und Bundesbehörden).

Wasserfall-Modell Sequentielle, stufenweise und dokumentenorientierte Entwicklung eines Produkts, wobei jede Aktivität in der vollen Produktbreite durchgeführt wird und abgeschlossen sein muß, bevor die nächste Aktivität beginnt.

Prozeß-Modelle legen fest, in welcher Abfolge und wie welche Aktivitäten der Software-Entwicklung durchgeführt werden sollen.

Das Wasserfall-Modell und das V-Modell sind klassische Prozeß-Modelle. In das V-Modell integriert sind die Validation und Verifikation. Außerdem werden Rollen explizit Aktivitäten zugewiesen. Das Prototypen Modell *(prototyping)* erlaubt es, Demonstrationsprototypen, Prototypen im engeren Sinne, Labormuster und Pilotsysteme zu erstellen, in Abhängigkeit von der Zielsetzung. Horizontale und vertikale Prototypen ermöglichen es, spezielle Bereiche des zu entwickelnden Software-Produkts als Muster zu entwickeln. Im Gegensatz zum Wasserfall- und V-Modell werden bei einem Pilotsystem sowie beim evolutionären und inkrementellen Modell zunächst nur gut verstandene Teile des Software-Produkts entwickelt und anschließend

Prozeß-Modell	Primäres Ziel	Antreibendes Moment	Benutzer-beteiligung *)	Charakteristika
Wasserfall-Modell	minimaler Management-aufwand	Dokumente	gering	sequentiell, volle Breite
V-Modell	maximale Qualität *(safe-to-market)*	Dokumente	gering	sequentiell, volle Breite, Validation, Verifikation
Prototypen-Modell	Risiko-minimierung	Code	hoch	nur Teilsysteme (horizontal oder vertikal)
Evolutionäres Modell	minimale Entwicklungs-zeit *(fast-to-market)*	Code	mittel	sofort: nur Kernsystem
Inkrementelles Modell	minimale Entw. -zeit *(fast-to-market)*, Risikominimierung	Code	mittel	volle Definition, dann zunächst nur Kernsystem
Objekt-orientiertes Modell	Zeit- und Kostenminimierung durch Wieder-verwendung	Wiederver-wendbare Komponenten	?	volle Breite in Abhängigkeit von wiederver-wendbaren Komponenten
Neben-läufiges Modell	minimale Entwicklungs-zeit *(fast-to-market)*	Zeit	hoch	volle Breite, nebenläufig
Spiralmodell	Risiko-minimierung	Risiko	mittel	Entscheidung pro Zyklus über weiteres Vorgehen

*) Marketing, Vertrieb, Auftraggeber, Endbenutzer

Tab. 3.3-5: Überblick über Prozeß-Modelle

schrittweise ausgebaut. Das objektorientierte Modell stellt die Wiederverwendung von Software-Komponenten in der Vordergrund, das nebenläufige Modell versucht einen hohen Parallelisierungsgrad zu erreichen. Bei dem Spiralmodell handelt es sich um eine Art »Metamodell«. Jedes Teilprodukt durchläuft mehrfach einen Zyklus aus vier Schritten, wobei in jedem Zyklus in Abhängigkeit von den Risiken geeignete Prozeß-Modelle für die Realisierung des jeweiligen Teilprodukts ausgewählt und eingesetzt werden.

Die einzelnen Prozeß-Modelle können meistens sinnvoll miteinander kombiniert werden, in Abhängigkeit von der gegebenen Entwicklungssituation. Tab. 3.3-5 zeigt einen zusammenfassenden Überblick über die Prozeß-Modelle.

Literatur

/Bröhl, Droschel 93/
Bröhl A.-P., Dröschel W. (Hrsg.), *Das V-Modell*, München. Oldenbourg Verlag 1993, 174 Seiten plus Anhang.
Beschreibt das für die Bundeswehr und die Bundesbehörden entwickelte V-Modell. Der umfangreiche Anhang enthält die im Bundesanzeiger veröffentlichte Dokumentation des V-Modells.
/Kieback et al. 92/
Kieback A., Lichter H., Schneider-Hufschmidt M., Züllighoven H., *Prototyping in industriellen Software-Produkten*, in: Informatik-Spektrum (1992) 15: 65–77.
Überblicksartikel über die Prototypen-Entwicklung einschließlich industriellen Erfahrungsberichten.

Zitierte Literatur

/Benington 56/
Benington H.D., *Production of Large Computer Programs*, in: Proc. ONR Symposium on Advanced Programming Methods for Digital Computers, June 1956, pp. 15–27.
/Boehm 76/
Boehm B.W., *Software Engineering*, in: IEEE Transactions on Computers, Dec. 1976, pp. 1226–1241.
/Boehm 81/
Boehm B.W., *Software Engineering Economics*, Englewood Cliffs: Prentice Hall 1981.
/Boehm 84/
Boehm B.W., *Verifying and Validating Software Requirements and Design Specifications*, in: IEEE Software, Jan. 1984, pp. 75–88.
/Boehm 86/
Boehm B., *A Spiral Model of Software Development and Enhancement*, in: ACM SIGSOFT, Aug. 1986, pp. 14–24.
/Boehm 88/
Boehm B.W., *A Spiral Model of Software Development and Enhancement*, in: IEEE Computer, May 1988, pp. 61–72.
/Budde et al. 84/
Budde R., Kuhlenkamp K., Mathiessen L., Züllighoven H. (eds.), *Approaches to Prototyping*, Berlin: Springer-Verlag 1984.
/Budde et al. 92/
Budde R., Kantz K., Kuhlenkamp K., Züllighoven H., *Prototyping - An Approach to Evolutionary System Development*, Berlin: Springer-Verlag, 1992.
/Davis 94/
Davis T., *Adopting a policy of reuse*, in: IEEE Spectrum, June 1994, p. 44–48.

/Eiff 91/
Eiff W., *Prozesse optimieren – Nutzen erschließen*, in: IBM Nachrichten 41 (1991), S. 23–27.
/Floyd 84/
Floyd C., *A Systematic Look At Prototyping*, in: /Budde et al. 84/, S. 1–18.
/Henderson-Sellers, Edwards 90/
Henderson-Sellers B., Edwards J.M., *The Object-Oriented Systems Life Cycle*, in: Communications of the ACM, Sept. 1990, p. 143–159.
/Henderson-Sellers, Edwards 93/
Henderson-Sellers B., Edwards J.M., *The O-O-O methodology for the object-oriented life cycle*, in: ACM SIGSOFT, Oct. 1993, p. 54–60.
/KBSt 92/
Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung, *Vorgehensmodell*, Band 27/1, August 1992, Bundesanzeiger (ebenfalls enthalten in /Bröhl, Dröschel 93/).
/Lausecker 93/
Lausecker H., *Ein strategisches Programm zur Erreichung der Wiederverwendbarkeit in einem großen Unternehmen*, in: Konferenzunterlagen, I.I.R.-Konferenz Re-Use, 8.-9. Sept. 1993, München.
/Marty 94/
Marty R., *Anwendungsbericht der Schweizerischen Bankgesellschaft: Organisation und Management bei der Einführung von Objektorientierung*, in: Konferenzunterlagen, I.I.R. Konferenz Objektorientierung, 13-15.6.94 Zürich.
/Pittman 93/
Pittman M., *Lessons Learned in Managing Object-Oriented Development*, in: IEEE Software, Jan. 1993, p. 43–53.
/Royce 70/
Royce W.W., *Managing the development of large software systems*, in: IEEE WESCON, Aug. 1970, pp. 1-9 (Nachdruck in: Proceedings of the 9th International Conference of Software Engineering, 1987, Monterey, CA., pp. 328–338).
/Spectrum 91/
Special Report: *Concurrent Engineering*, in: IEEE Spectrum July 1991, pp. 22–37.
/Versteegen 96/
Versteegen G., *V-gefertigt – Das fortgeschriebene Vorgehensmodell*, in: iX 9/1996, S. 140–147.
/V-Modell 97/
Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell, Teil 1: Regelungsteil, Teil 3: Handbuchsammlung, Allgemeiner Umdruck Nr. 250/1, Juni 1997, BWB IT 15, Koblenz.
/Weitzel, Kerschberg 89/
Weitzel J.R., Kerschberg L., *Developing Knowledge-Based Systems: Reorganizing the System Development Life Cycle*, in: Communications of the ACM April 1989, pp. 452–488.

Muß-Aufgab
10 Minuten

1 *Lernziel: Den Unterschied zwischen einem horizontalen und einem vertikalen Prototyp schildern können.*
Im Rahmen einer Software-Entwicklung wird für eine große Anwendung zunächst die gesamte *Client/Server*-Verteilung erstellt. Alle anderen Aspekte des Systems interessieren zunächst nicht.

a Handelt es sich in diesem Fall um einen horizontalen oder einen vertikalen Prototypen?

b Wie könnte der Entwicklungsauftrag aussehen, wenn die in **a** nicht gefundene Variante eines Prototyps entwickelt werden soll?

Muß-Aufgabe
20 Minuten

2 *Lernziele: Die verschiedenen Arten von Prototypen kennen und ihre Unterschiede aufzeigen können. Die acht beschriebenen Prozeß-Modelle erklären und ihre Unterschiede darstellen können.*

a Beim Prototypen-Modell steht zunächst die Entwicklung von Prototypen im Vordergrund. Welche vier verschiedenen Arten von Prototypen kennen Sie? Welche Eigenschaften haben diese Prototypen?

b Welche Mängel der klassischen Prozeß-Modelle führten zum Prototypen-Modell?

Muß-Aufgabe
5 Minuten

3 *Lernziel: Aufzählen können, welche Aspekte ein Prozeß-Modell festlegen soll.*
Welche der folgenden Aspekte werden durch ein Prozeß-Modell festgelegt und welche nicht?

- Beginn des Projekts
- Reihenfolge des Arbeitsablaufs
- Arbeitszeiten
- Jeweils durchzuführende Aktivitäten
- Definition der Teilprodukte
- Fertigstellungszeitpunkt
- Fertigstellungskriterien
- Mitarbeiterqualifikationen
- Mitarbeiterauswahl
- Verantwortlichkeiten
- Verwaltung
- Methoden, Werkzeuge
- Richtlinien, Standards
- Programmiersprache

Kann-Aufgabe
5 Minuten

4 *Lernziel: Die acht beschriebenen Prozeß-Modelle erklären und ihre Unterschiede darstellen können.*
Beschreiben Sie den Unterschied zwischen dem evolutionären und dem inkrementellen Modell.

Muß-Aufgabe
10 Minuten

5 *Lernziele: Die Begriffe Validation und Verifikation im V-Modell erklären können. Die acht beschriebenen Prozeß-Modelle beschreiben und ihre Unterschiede darstellen können.*
Arbeitet man nach dem V-Modell, dann stellt man sich häufig zwei Fragen: »Wird das Produkt richtig entwickelt?« und »Wird das richtige Produkt entwikkelt?«

a Mit welchen Begriffen beschreibt das V-Modell die beiden Fragen?

b In welchem Submodell werden diese Fragen besonders berücksichtigt?

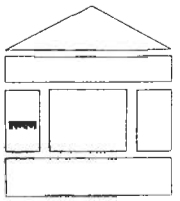
c Auf welchem Prozeß-Modell basiert das V-Modell? Wie sehen die wichtigsten Erweiterungen aus?

Muß-Aufgabe
15 Minuten

6 *Lernziel: Für vorgegebene Szenarien den Ablauf entsprechend einem Prozeß-Modell in der vorgestellten genormten Weise darstellen können.*
Im folgenden ist der Ablauf einer Software-Entwicklung beschrieben. Versuchen Sie, den Ablauf zu beschreiben, wenn Sie **a** das Wasserfall-Modell und **b** das Prototypen-Modell zugrunde legen.
Bei dem zu entwickelnden System handelt es sich um eine Roboter-Animation. Die Entwicklung begann mit der Erstellung eines objektorientierten Analysemodells. Dort war zunächst geplant, einen physikalischen Roboter grundsätzlich parallel zu der Animation zu betreiben. Während der Entwicklung der Hardware-Ansteuerung stellte sich jedoch heraus, daß die Ansteuerung nicht für beliebige Betriebssystemversionen funktioniert. Um sich dort eine gewisse Unabhängigkeit zu erhalten, wurde die Produktdefinition so geändert, daß auch ein Betrieb ohne einen physikalischen Roboter möglich ist (Simulation).
Damit konnte die Entwicklung der Benutzungsoberfläche und der Hardware-Ansteuerung – nach Festlegung entsprechender Schnittstellen – getrennt werden. Die Schnittstelle wurde während der Entwicklung einmal geändert, weil sich herausstellte, daß für die Benutzungsoberfläche zusätzliche Informationen notwendig waren.

Weitere Aufgaben befinden sich auf der beiliegenden CD-ROM.

4 Personal



wissen

- Angeben können, welche Qualifikationen für die beschriebenen Rollen erforderlich sind.
- Wissen, wie die Kosten der deutschen Software-Produktion im Verhältnis zu anderen Ländern aussehen.
- Personalaufgaben und zugehörige Aktivitäten nennen können.
- Die Bedeutung von Personalfragen anhand von Beispielen aufzählen können.

verstehen

- Allgemeine Qualifikationen für Mitarbeiter in der Software-Technik benennen und begründen können.
- Die horizontale und vertikale Spezialisierung sowie ihre Unterschiede und Vor- und Nachteile erklären können.
- Die Unterschiede zwischen einer Führungs- und einer Fachlaufbahn erklären können.
- Aufgaben und Probleme bei der Weiterbildung nennen und begründen können.
- Aufgaben und Probleme bei der Personalentwicklung nennen und begründen können.

4.1 Grundlagen

Die Autoren DeMarco und Lister berichten in ihrem Buch /DeMarco, Lister 91, S. 4/, daß ca. fünfzehn Prozent aller untersuchten Projekte in der Software-Entwicklung fehlgeschlagen sind. Sie wurden abgebrochen, zurückgestellt oder haben Ergebnisse geliefert, die nicht gebraucht wurden. Bei größeren Projekten, mit mehr als 25 Personenjahren, wurden 25 Prozent nicht fertiggestellt. Eine Analyse der Ursachen ergab, daß die überwältigende Mehrheit der Projekte *nicht* aufgrund von technischen Problemen gescheitert ist. Die Hauptursache waren personenbezogene Probleme. Dies veranlaßt DeMarco und Lister zu folgender These:

»Die größten Probleme bei unserer Arbeit sind keine technologischen Probleme, sondern soziologische Probleme« /a.a.O., S. 5/.

Bedeutung von Personalfragen

Software-Manager konzentrieren sich in der Regel nicht auf diesen Themenbereich. Personalfragen haben oft die niedrigste Priorität. Ein Teil dieses Phänomens läßt sich dadurch erklären, daß die meisten zuerst gelernt haben, wie die Arbeit gemacht wird und nicht, wie sie organisiert wird.

Erfolgreiche Software-Entwicklungen lassen sich auf gute menschliche Zusammenarbeit zurückführen; Fehler ergeben sich oft aus schlechter menschlicher Kooperation. Dennoch befassen sich die meisten Manager mehr mit der technischen Seite der Software-Entwicklung. Dies liegt sicher auch daran, daß menschliche Beziehungen kompliziert und ihre Auswirkungen oft nicht deutlich zu beobachten sind.

Abschnitt 1.4

Verschiedene empirische Untersuchungen haben gezeigt, daß der Produktivitätsunterschied zwischen dem besten und dem schlechtesten Mitarbeiter einer Firma den Faktor 10 erreichen kann. Aus der Managementsicht lohnt es sich also, gutes Personal einzustellen. Damit Mitarbeiter produktiv eingesetzt werden können, müssen sie über

- allgemeine Qualifikationen verfügen, die eine Tätigkeit im Bereich der Software-Technik generell erfordert, und über
- spezielle Qualifikationen verfügen, die für ihre Tätigkeit innerhalb der Software-Technik erforderlich sind.

In den nächsten Abschnitten wird auf beide Qualifikationen näher eingegangen. Außerdem wird gezeigt, wie eine Führungs- und eine Fachlaufbahn aussehen können.

4.1.1 Allgemeine Qualifikationen

Ein Software-Mitarbeiter benötigt eine Reihe von allgemeinen Qualifikationen, die ihn dazu befähigen Software zu erstellen:

- Fähigkeit zum Abstrahieren
  Abstraktionsvermögen ist eine der Grundfähigkeiten, die in der

  Software-Technik benötigt werden. Nur mit Hilfe der Abstraktion können komplexe Systeme bewältigt werden.
- Sprachliche und schriftliche Kommunikationsfähigkeit
  Da Software-Erstellung in Teams erfolgt, ist eine gute sprachliche Ausdrucksweise und Präsentation wichtig. Eine gute Schriftform ist für die Software-Dokumentation erforderlich.
- Teamfähigkeit
  Um produktiv zur Teamleistung beitragen zu können, muß ein Mitarbeiter Teamgeist besitzen und konstruktiv und kooperativ seine Beiträge zum Teamergebnis liefern.

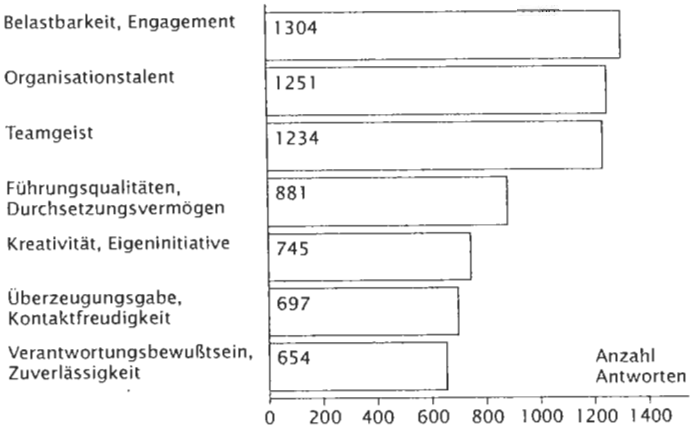
Abschnitt 5.3

- Wille zum lebenslangen Lernen
  Das Wissen in der Software-Technik verdoppelt sich etwa alle vier Jahre. Um auf dem Stand der Technik zu bleiben, ist daher ein permanenter Lernprozeß erforderlich.
- Intellektuelle Flexibilität und Mobilität
  Nicht nur die Software-Technik selbst entwickelt sich ständig weiter. Auch das gesamte Umfeld der Software-Technik ändert sich permanent. Daher sind flexibles Denken und Anpassungsfähigkeit an neue Situationen gefordert.
- Kreativität
  Da die Software-Technik noch kein breites Erfahrungspotential besitzt, ist hohe Kreativität gefordert, um neue Lösungen zu finden.
- Hohe Belastbarkeit
  Da die meisten Software-Entwicklungen unter Termin- und Kostendruck stehen sowie mit knappen Personalressourcen durchgeführt werden, muß ein Mitarbeiter »streßverträglich« sein.
  »Für 83% aller befragten EDV-Fachkräfte hat sich das Arbeitspensum in den letzten Jahren erhöht... 85% aller befragten EDV-Fachkräfte leisten Mehrarbeit/Überstunden.
  Etwa ein Drittel der Befragten leistet sie regelmäßig. 21% leisten mehr als fünf Überstunden pro Woche.
  Von 45% werden Überstunden sogar ohne Ausgleich in Form von Freizeit oder Bezahlung geleistet.« /Office Management 89, S. 27/
- Englisch lesen und sprechen
  Die Software-Branche wird stark von den USA dominiert. Daher sind Publikationen, Handbücher und Produkte im Original meist in Englisch abgefaßt. Selbst deutsche Firmen erstellen ihre Produkte und Dokumente oft zuerst in Englisch, um die internationalen Märkte bedienen zu können. Viele Produkte und Dokumente gibt es nur in Englisch. Auch die bevorzugte Kommunikationssprache in der Software-Technik ist Englisch.
  /Naisbitt 90, S. 180/ weist noch auf einen anderen Gesichtspunkt hin:
  »Mehr als 80 Prozent aller Informationen in den über 100 Millionen Computern, die es auf der Welt gibt, sind auf Englisch gespeichert.«

- Schreibmaschine schreiben
  Eingaben in ein Computersystem erfolgen heute noch überwiegend über die Tastatur. Da ein Software-Entwickler diese Eingaben nicht von einer Hilfskraft abgenommen bekommt, ist das professionelle Schreibmaschineschreiben blind im Zehnfingersystem eine notwendige Fähigkeit.

Eine Stellenauswertung der Zeitungen Computerwoche, FAZ, SZ und Die Welt im 1. Quartal 1989 hat die in Abb. 4.1-1 aufgeführten fachübergreifenden Qualifikationen ergeben (/Maisberger 89, S. 20/, Quelle: Control Data Institut).

Abb. 4.1-1: Gewünschte fachübergreifende Qualifikationen



4.1.2 Spezialisierung

In der Software-Entwicklung überwiegt heute noch der Generalist, der von der Analyse bis zur Programmierung an einem Software-Produkt mitarbeitet. In /Weber 92, S. 78f/ wird das Problem der Spezialisierung anschaulich mit anderen Branchen verglichen:

Zitat »Es ist jedoch hinlänglich akzeptiert, daß die Softwaresysteme, die wir konstruieren, von einer überragenden Größe und Komplexität sind und insoweit durchaus vergleichbar sind mit anderen großtechnischen Systemen wie Flugzeugen, Staudämmen, kompletten Fabrikanlagen etc.

Schon ein flüchtiger Blick auf andere technische Großsysteme erlaubt die Feststellung, daß sie nur in Arbeitsteilung erstellt werden können und daß diese Arbeitsteilung im wesentlichen darin begründet ist, daß die Vielzahl der bei der Entwicklung dieser Systeme zu lösenden Probleme die Problemlösungskompetenz einer Berufsgruppe bei weitem übersteigt. So ist zum Beispiel bei der Konstruktion eines Wasserkraftwerkes, zur Konstruktion des dazu notwendigen Staudamms und zur Konstruktion des gesamten Kraftwerkes der Einsatz

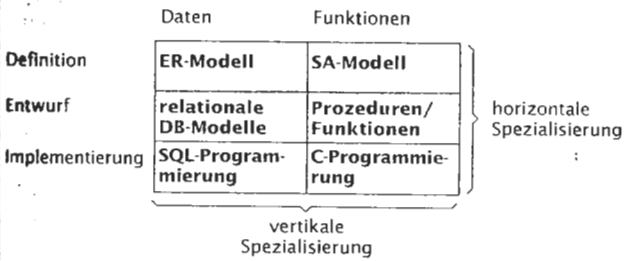
von Spezialisten mit etwa 200 verschiedenen beruflichen Spezialisierungen notwendig. Es sind für den Bau einer solchen Gesamtanlage und für deren Betrieb auch etwa 200 in ihrer Art sehr verschiedene Softwaresysteme nötig.

Dies ist einerseits ein Zeichen dafür, daß Software für nahezu jede der zum Einsatz kommenden Technologien Verwendung findet. Es ist aber auch ein Zeichen dafür, daß die insgesamt erzeugte Software 200 verschiedenen beruflichen Spezialisierungen zuzuordnen ist. Es kann selbstverständlich nicht davon ausgegangen werden, daß Software-Entwickler dazu qualifiziert sind, Software für etwa 200 verschiedene Ingenieuranwendungen in gleicher Weise gut produzieren zu können. Vielmehr erfordert eine solche komplexe Software-Aufgabe eine Spezialisierung der Softwareentwickler. Zufriedenstellende Ergebnisse in der Softwareentwicklung können nur erreicht werden, wenn die Spezialisierung der Softwareentwickler den Erwerb der notwendigen Anwendungsfachkompetenz einschließt.«

Eine Spezialisierung kann in der Software-Technik vertikal und horizontal erfolgen.

Beispiel Ein Beispiel zeigt Abb. 4.1-2. Bei der vertikalen Spezialisierung ist ein Datenspezialist zuständig für die *Entity-Relationship*-Modellierung, anschließend für die Umsetzung dieses Modells in ein relationales Datenbankmodell und letztendlich für die Programmierung des DB-Modells in SQL. Parallel dazu kümmert sich ein Funktionsspezialist um die Modellierung, den Entwurf und die Implementierung der Funktionen.

Abb. 4.1-2: Horizontale vs. vertikale Spezialisierung



Bei der horizontalen Spezialisierung modelliert ein Systemanalytiker die Anforderungen, ein Software-Entwerfer konzipiert die Systemarchitektur und ein Programmierer implementiert das System.

Bewertung

Die **vertikale Spezialisierung** ist heute vorherrschend. Sie besitzt den Vorteil, daß es bei ihr weniger Spezialisierungsgebiete gibt als bei der horizontalen Spezialisierung. Vorteil

Dem stehen folgende Nachteile gegenüber: Nachteile

- Nicht bei jeder Software-Methode einsetzbar, z.B. bei der objektorientierten Software-Entwicklung.

- Verlangt vom Spezialisten sehr unterschiedliche Qualifikationen.
- Jede Tätigkeit wird nur selten durchgeführt, eine Analyse z.B. nur alle zwei Jahre.
- Gefahr, daß auf der jeweiligen Ebene die Produktteile nicht zusammenpassen.

Vorteile

Die **horizontale Spezialisierung** besitzt folgende Vorteile:

- Volle Nutzung der Qualifikationen.
- Wiederholung der gleichen Tätigkeiten in kurzen Abständen, z.B. jedes halbe Jahr eine neue Systemanalyse.
- Höhere Chancen für die Wiederverwendung vorhandener Komponenten, da besserer Überblick wegen Wiederholung der Tätigkeiten in kurzen Abständen.
- Chance, ständig den »Stand der Technik« auf dem Spezialisierungsgebiet zu halten.

Nachteile

Dem stehen folgende Nachteile gegenüber:

- Gefahr, daß zwischen den jeweiligen Ebenen das Produkt nicht zusammenpaßt.
- Mehr Spezialisierungsgebiete als bei der vertikalen Spezialisierung.

Probleme

Jede **Spezialisierung** bringt folgende Probleme mit sich:

- Für das Management wird es schwieriger, eine gleichmäßige Auslastung über längere Zeiträume zu planen.
- Die Einsatzmöglichkeiten für einzelne Mitarbeiter sind eingeschränkter.

Wegen der notwendigen fachlichen Qualifikationen und der hohen Innovationsgeschwindigkeit ist eine Spezialisierung in der Software-Technik aber unvermeidlich. Die Vielzahl der Vorteile spricht für eine horizontale Spezialisierung. Die Nachteile müssen durch das Software-Management vermieden werden.

Abschnitt 3.3.2

In der Software-Technik wird der Begriff **Rolle** verwendet, um die Erfahrungen, Kenntnisse und Fähigkeiten zu beschreiben, die nötig sind, um einzelne Aufgaben zu erledigen.

Für wichtige Rollen werden im folgenden die notwendigen Qualifikationen beschrieben:

Systemanalytiker *(Requirements Engineer)*

- Abstraktes Denken (Vom Konkreten zum Abstrakten)
- Hohe Kommunikationsbereitschaft
- Hineindenken in andere Begriffs- und Vorstellungswelten
- Fachwissen aus den Anwendungsgebieten
- Flexibilität

Software-Entwerfer / Software-Architekt

- Abstraktes Denken (Vom fachlich Abstrakten zum software-technisch Konkreten)
- Konzeptionelles Denken (nicht im Detail versinken)

Implementierer / Programmierer / Algorithmenkonstrukteur

- Mathematisches Denkvermögen (Verifikation, Aufwand)
- Abstraktionsvermögen
- Kreativität
- Präzision

Qualitätssicherer

- Geduld
- Hartnäckigkeit
- Mathematisches Denkvermögen

Software-Ergonom

- Interdisziplinäres Wissen aus kognitiver Psychologie, Arbeitswissenschaft und Software-Technik
- Fähigkeit zu konzeptioneller, experimenteller und evaluatorischer Tätigkeit
- Hohe Kommunikationsbereitschaft

Anwendungsspezialist

- Breites Modellierungswissen über das Anwendungsgebiet
- Hohe Kommunikationsbereitschaft
- Abstraktes Denken (Vom Konkreten zum Abstrakten)
- Fähigkeit, das Anwendungsgebiet aufgabengerecht zu strukturieren
- Kenntnis vorhandener Software-Systeme für das Anwendungsgebiet
- Automatisierungsmöglichkeiten des Anwendungsgebiets einschätzen können unter Berücksichtigung von wirtschaftlichen und ergonomischen Gesichtspunkten
- Ganzheitliches Denken, z.B. in Geschäftsprozessen

Software-Manager

- Kooperativer und mitarbeiterorientierter Führungsstil
- Fähigkeit zum strategischen Denken
- Mitarbeiter zum selbständigen und eigenverantwortlichen Handeln anleiten
- Kenntnisse in Problemlösung und Konfliktmanagement
- Initiieren und Fördern von Innovationen
- Fachübergreifendes Denken
- Entwickler und Förderer seiner Mitarbeiter
- Fähigkeit und Bereitschaft zum Delegieren
- Kreativität

In seinem Buch »Nieten in Nadelstreifen« beschreibt Günter Ogger die neuen Manager folgendermaßen /Ogger 92, S. 24ff./:

Zitat

»Für die künftigen Bosse ist deshalb ein Lebenslauf, der sich durch eine Reihe abwechslungsreicher Tätigkeiten und frühzeitiger Übernahme von Verantwortung auszeichnet, wahrscheinlich hilfreicher als das bisher so hoch bewertete Streben nach guten Noten und Examina.

Ein Betriebswirt zum Beispiel, der nach dem Studium eine Weile in Australien jobbt, in Hongkong eine Einkaufsorganisation für deutsche Einzelhändler gründet und schließlich in Leipzig einen Gebrauchtwagenhandel aufmacht, taugt als Chef einer »business unit« wahrscheinlich mehr als der Klassenprimus, der als Trainee bei Daimler-Benz anfängt und sich in der Finanzabteilung Schritt für Schritt vorarbeitet, ohne jemals ein Auto verkauft zu haben (...)

Manager-Eigenschaften

Der Manager der Zukunft ist ein Mann mit drei hervorstechenden Eigenschaften: Er hat eine Witterung für profitable Geschäfte, kann strategisch denken und mit Menschen umgehen. Fachwissen, das sich ohnehin schnell überholt, eignet er sich bei Bedarf an; er legt keinen Wert auf Status- und Abgrenzungssymbole, sondern er überzeugt durch Aufrichtigkeit und Kompetenz (...)

Erheblich an Bedeutung gewinnt demnach auch die Auswahl und Entwicklung des Personals sowie die Fähigkeit zur Verhandlungsführung und Konfliktlösung. Die Topleute des 21. Jahrhunderts werden weniger am Schreibtisch sitzen als ihre Kollegen von heute, dafür aber viel mehr mit ihren Mitarbeitern, Kunden und Partnern kommunizieren. Sie brauchen, darin waren sich die Befragten einig, vor allem eine klare Vorstellung von der Zukunft ihrer Firma, und die müssen sie nach drinnen wie draußen vermitteln können.

Deshalb müssen Spitzenmanager in Zukunft vor allem enthusiastisch sein – das fordern 92 Prozent aller Befragten. Sie sollen inspirieren (91 Prozent), Mut machen (89 Prozent). aufgeschlossen und kreativ (88 Prozent) und ein Beispiel an ethischem Verhalten geben (88 Prozent). Interessant auch, daß konservatives Verhalten künftig weniger wichtig sein wird als heute, und daß differenzierte, auch widersprüchliche Charaktere mehr gefragt sein werden. Der ideale Boß des Jahres 2000 ist demnach ein Kosmopolit von hervorragender Allgemeinbildung mit einem Verständnis für unterschiedliche Kulturen, er ist ein erstklassiger Teamarbeiter, aber in seinem Denken unabhängig (...)

Die uralten Tugenden der Betriebswirtschaft, nämlich Sparsamkeit, Einfachheit und Rentabilität sind jetzt wieder gefragt.«

Abschnitt 3.3.2 Projektmanagement-Rollen

Im V-Modell werden 25 verschiedene Rollen vorgeschlagen (Tab. 3.3-1) /V-Modell 97, S. R-2/. Die Projektmanagement-Rollen werden dort folgendermaßen charakterisiert:

Projektmanager (Manager)
Aufgaben
- Festlegung der Rahmenbedingungen für die Projektorganisation
- Initialisierung und Koordination des Projekts, gegebenenfalls Koordination mehrerer Projekte
- Initialisierung und Mitgestaltung der Vergabe
- Kontrolle und Einhaltung der vertraglichen Abmachungen
- Kommunikation mit dem Auftragnehmer
- Beauftragung der Kosten-Nutzenanalyse und ihre Bewertung

- Problem- und Konfliktlösung bei der Projektplanung, bei der Projektabwicklung und beim Projektabschluß
- Kooperation mit dem Qualitätssicherungs- und Konfigurationsmanagement-Manager
- Mitarbeit bei Durchführungsentscheidungen und Umsetzung der getroffenen Entscheidungen

Geforderte Kenntnisse und Fähigkeiten
- Kenntnisse auf betriebswirtschaftlichem Gebiet, aber auch technisches Verständnis
- Erfahrung in der Projektorganisation
- Kenntnis über Anwendung und Einsatzgebiet des Systems
- Führungsqualitäten
- Fähigkeit zu Organisation und Delegation
- Fähigkeit zu positiver Sichtweise

Projektleiter (Verantwortlicher)
Aufgaben
- Planung, Steuerung und Kontrolle des Projekts
- Erstellung des Projekthandbuchs und -plans (Aufwand, Termine und Einsatzmittel) basierend auf den Projektzielen und den organisatorischen Rahmenbedingungen
- Vorbereitung und Durchführung von Ausschreibungen
- Überwachung der Vertragserfüllung
- Durchführung von Kosten-Nutzenanalyse
- Vorbereitung und Begleitung von Durchführungsentscheidungen
- Erkennen möglicher Risiken im Projekt und Einleitung geeigneter Maßnahmen
- Beauftragung der Projektmitarbeiter
- Berichterstattung gegenüber dem Projektmanager und gegebenenfalls weiteren Lenkungsgremien

Geforderte Kenntnisse und Fähigkeiten
- Erfahrung in der Projektabwicklung
- Verständnis von betriebswirtschaftlichen Zusammenhängen
- Kenntnis über Anwendung und Einsatzgebiete des Systems
- Kenntnis über die Entwicklungsumgebung
- Kenntnis über Methoden und Werkzeuge
- Durchsetzungsvermögen und Akzeptanz bei den Projektmitarbeitern
- Fähigkeit zur Führung, Motivation und Moderation
- Fähigkeit zu Organisation und Kommunikation

Rechtsverantwortlicher (Verantwortlicher)
Aufgaben
- Betreuung des Projektmanagers und Projektleiters in juristischen Angelegenheiten

- Mitwirkung bei der Vergabe von Aufträgen
- Führung von Vertragsverhandlungen bis hin zum Vertragsabschluß
- Überwachung der Vertragserfüllung

Geforderte Kenntnisse und Fähigkeiten
- Juristische Kenntnis in der Vergabe und Haftung
- Verhandlungsgeschick

Projektadministrator (Durchführender)
Aufgaben
- Unterstützung des Projektleiters
- Koordination und Verwaltung von Arbeitsaufträgen
- Erstellung und Verwaltung von Berichtsdokumenten

Geforderte Kenntnisse und Fähigkeiten
- Kenntnis und Erfahrung in Methoden und Werkzeugen des Projektmanagements
- Fähigkeit zu Organisation und Kommunikation

4.1.3 Führungs- und Fachlaufbahn

Abb. 4.1-3: ifikationen der Fa. IKOSS

Um Mitarbeiter über lange Zeit hin zu motivieren, muß man ihnen Perspektiven und Aufstiegschancen eröffnen. Da nicht alle Mitarbeiter Manager werden können und wollen, ist es wichtig, daß neben

Qualif.-Gruppe	Führungslaufbahn			Funktionen	Fachlaufbahn		
	Technik	Vertrieb	Kaufm.		Technik	Vertrieb	Kaufm.
I	GF GL	GF GL	GF GL	Geschäftsführer Geschäftsleiter Senior Chefberater	SCB	---	---
II	BL AL	BL AL	BL AL	Bereichsleiter/Abteilungsleiter Technik, Vertriebsleiter, Kfm. Leiter, Personalleiter, Chefberater, Senior-Vertriebsbeauftragter, Chefcontroller	CB SPL SB	SVB	CR/CC
III	Keine definierten Funktionen			Systemanalytiker, Projektleiter, Vertriebsbeauftragter, Referent, Seniorcontroller, Assistent in I	SS SI PL SA	VB	R/SC ASS I
IV				Systemprogrammierer, Anwendungsprogrammierer, Vertriebsbeauftragter, Sachbearbeiter, Controller, Assistent in II	SP ST AOP		SB/C ASS II
V				Programmierer, Juniorcontroller, Juniorsachbearbeiter, Vertriebsassistent, Assistent in III	P	VA Berufseinstiegsfunktion	JC JSB ASS III

einer Führungslaufbahn a
handen ist.

Während es die **Führungslaufbahn** dem Mitarbeiter ermöglicht, zunehmend Personalverantwortung zu übernehmen, erlaubt es die **Fachlaufbahn**, zunehmend größere fachliche Verantwortung zu übernehmen.

Führungslaufbahn

Fachlaufbahn

Ein Beispiel für eine Aufteilung in Führungs- und Fachlaufbahn zeigt die Abb. 4.1-3 der Firma IKOSS /IKOSS 94/. Abb. 4.1-4 stellt die Aufstiegsmöglichkeiten in der Fachlaufbahn Technik und Vertrieb dar, einschließlich möglicher Übergänge zwischen Technik und Vertrieb /IKOSS 94/.

Beispiel

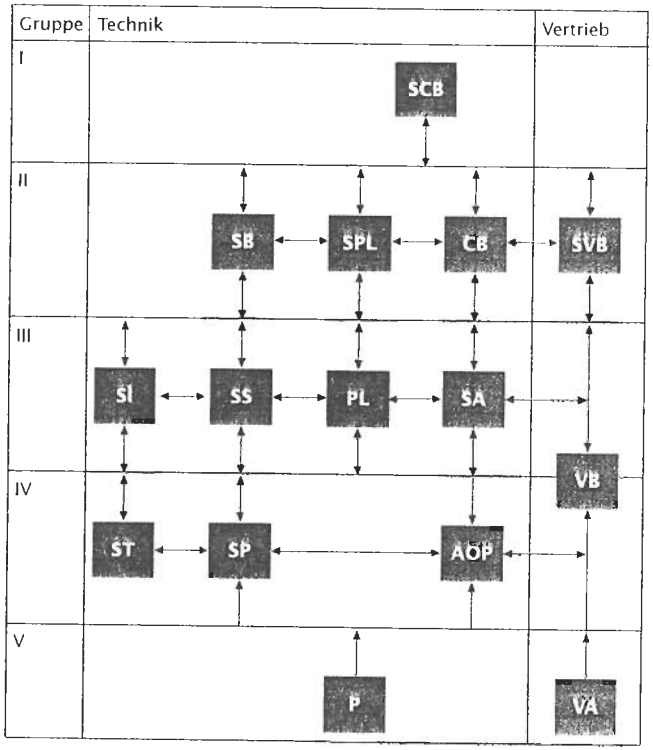


Abb. 4.1-4: Qualifikationsschema für die Fachlaufbahn Technik und Vertrieb der Fa. IKOSS

Legende:

Qualifikationsprofil	Schwerpunktmäßiger Einsatz als...	Qualifikationsprofil	Schwerpunktmäßiger Einsatz als...
P:	Programmierer	PL:	Projektleiter (Junior-PL)
ST:	Systemtechniker	SA:	Systemanalytiker
SP:	Systemprogrammierer	VB:	Vertriebsbeauftragter
AOP:	Anwendungs-/Organisationsprogrammierer	SB:	Systemberater
		SPL:	Senior-Projektleiter
SI:	Systemingenieur	CB:	Chefberater
VA:	Vertriebsassistent	SVB:	Senior-Vertriebsbeauftragter
SS:	Systemspezialist	SCB:	Senior-Chefberater

Die Tab. 4.1-1 zeigt, welche Qualifikationsmerkmale in welcher Qualifizierungsgruppe in welcher Ausprägung vorhanden sein sollen. Dieser Tabelle ist der für ein Mitarbeitergespräch erforderliche Maßstab für die Bewertung der derzeitigen Soll-Qualifikation zu entnehmen /IKOSS 94/.

Tab. 4.1-1: Qualifikationsprofile der Fachlaufbahn Technik in der Fa. IKOSS

V	IV			III				II				I	Gruppe
P	ST	AOP	SP	SI	SA	PL	SS	SB	SPL	CB	SCB		Qualifikationsmerkmale
1	2	2	2	3	3	4	3	4	4	5	5		**1** Methodik
2	2	2	2	3	3	3	4	5	4	5	5		**2** Analytisches Vorgehen
1	2	2	3	3	3	3	4	5	4	4	4		**3** Abstraktionsvermögen
1	2	2	2	3	4	4	4	5	5	5	5		**4** Gesamthafte Betrachtungsweise
2	3	3	3	3	4	4	4	4	4	4	4		**5** Dokumentation
1	2	2	2	3	3	3	3	3	5	4	4		**6** Planungsfähigkeit
1	2	1	1	2	3	3	3	3	5	4	4		**7** Organisationsfähigkeit
1	1	2	1	2	3	4	4	5	4	5	5		**8** Moderationsfähigkeit
1	1	2	1	2	3	4	4	5	4	5	5		**9** Präsentationsfähigkeit
1	1	1	1	3	4	2	1	3	3	4	5		**10** Marktkenntisse
1	1	2	2	2	3	4	3	5	4	5	5		**11** Kenntnisse über Themen, Ziele und Inhalte der IKOSS-Aktivitäten
1	2	1	1	2	3	2	2	3	3	4	4		**12** Fremdsprachen
3	3	3	3	3	3	4	3	4	4	4	5		**13** Integrationsfähigkeit
3	3	3	3	4	4	4	4	4	5	5	5		**14** Loyalität und Integrität
1	2	2	2	3	4	5	4	5	5	5	5		**15** Persönliches Auftreten
1	2	1	2	3	3	4	2	4	5	5	5		**16** Eigeninitiative und Begeisterungsfähigkeit
2	2	2	2	3	3	3	3	3	4	4	4		**17** Flexibilität
3	3	3	3	3	3	4	3	4	4	4	5		**18** Engagement und Fleiß
2	2	2	2	3	3	3	3	3	4	4	4		**19** Belastbarkeit
3	3	3	3	4	4	4	3	4	4	4	5		**20** Qualitätsbewußtsein
2	3	3	3	4	4	4	3	4	4	4	4		**21** Termintreue
1	3	3	3	3	3	4	3	4	4	4	4		**22** Kostenbewußtsein
1	2	3	3	3	4	3	4	5	3	5	5		**23** Kreativität
1	1	1	1	2	3	4	3	4	4	4	5		**24** Unternehmerisches Denken
3	3	3	3	3	4	4	4	4	4	4	4		**25** Weiterbildungsbereitschaft
2	2	3	3	3	3	4	3	5	4	5	5		**26** Kontaktfähigkeit
2	2	2	2	3	3	3	2	3	4	4	4		**27** Einfühlungsvermögen
1	2	3	3	3	4	4	4	4	5	5	5		**28** Problembewußtsein
1	1	2	2	2	3	4	3	4	4	4	4		**29** Konfliktfähigkeit
1	2	1	1	3	4	3	2	4	4	4	5		**30** Überzeugungsfähigkeit
1	2	1	1	3	3	3	2	4	4	4	5		**31** Durchsetzungsvermögen
1	1	1	1	2	3	3	3	5	4	5	5		**32** Erfolgsorientiertes Verhandeln
3	3	3	3	3	3	4	2	3	4	3	3		**33** Teamfähigkeit
1	1	1	1	3	2	3	1	3	4	3	4		**34** Führungsfähigkeit

Legende:
Wichtigkeit der Qualifikationsmerkmale von »sehr bedeutend« (5) bis »weniger bedeutend« (1)

4.2 Aufgaben und

Personalaufgaben beinhalten alle Aktivitäten, die mit dem Besetzen und dem Besetzt halten von Stellen zu tun haben, die durch eine Organisationsstruktur gegeben oder entstanden sind.

Dazu gehört die Auswahl von Bewerbern für die verfügbaren Stellen und die Aus- und Weiterbildung sowohl der neuen Mitarbeiter als auch der Stelleninhaber, damit sie ihre Aufgaben effektiv ausüben können. Die Mitarbeiterbeurteilung, -bezahlung und -entwicklung gehören ebenso zu den Personalaufgaben wie das Versetzen und Entlassen von Mitarbeitern. Das primäre Ziel besteht darin, die definierten Rollen mit Mitarbeitern zu besetzen, die fähig und motiviert sind, diese Rollen auszufüllen.

Im folgenden werden Aufgaben und Aktivitäten näher betrachtet, die softwarespezifische Eigenarten aufweisen.

4.2.1 Stellen besetzen

Ein Software-Manager ist für die Auswahl und Einstellung qualifizierter Mitarbeiter verantwortlich. Bei der Auswahl muß er sich von den Qualifikationen und den Erfahrungen der potentiellen Mitarbeiter überzeugen.

Neben der Überprüfung, ob die Qualifikationen mit dem geforderten Qualifikationsprofil der zu besetzenden Position übereinstimmen, sollten folgende Faktoren bei der Besetzung einer Stelle beachtet werden /Thayer 87, S. 35/:

Faktoren

- Ausbildung
  Hat der Bewerber die notwendige Ausbildung für die Stelle? Hat er die geeignete Ausbildung für seine zukünftigen Aufgaben in dem Unternehmen?
- Erfahrung
  Hat der Bewerber ausreichende Erfahrungen? Sind sie von der richtigen Art und von der notwendigen Vielfalt?
- Training
  Ist der Bewerber in den Programmiersprachen und den Methoden, die eingesetzt werden, ausreichend trainiert? Kennt er die benutzte Hardware und System-Software sowie den Anwendungsbereich des zu entwickelnden Software-Systems?
- Motivation
  Ist der Bewerber für die Stelle, die Arbeit im Projekt und die Arbeit für das Unternehmen motiviert?
- Engagement
  Zeigt der Bewerber Engagement und Loyalität zum Projekt, zum Unternehmen und zu den getroffenen Entscheidungen?

- Selbständigkeit
  Ist der Bewerber ein *»self-starter«* mit dem Willen, ein Projekt bis zum Ende – ohne übermäßige Personalführung – durchzustehen?
- Gruppenaffinität
  Paßt der Bewerber in die vorhandene Mannschaft? Gibt es potentielle Konflikte, die gelöst werden müssen?
- Intelligenz
  Besitzt der Bewerber die Fähigkeit zum Lernen, zur Übernahme schwieriger Aufgaben und zur Anpassung an wechselnde Umgebungen?
- Persönlichkeit
  Ist der Bewerber eine reife und ausgeglichene Persönlichkeit mit richtiger Einschätzung von sich selbst und der Umwelt?

Schwächen auf einigen dieser Gebiete können durch Stärken auf anderen Gebieten ausgeglichen werden. Ausbildungsschwächen können z.B. durch umfangreiche Erfahrung, spezielle Weiterbildungen oder durch Engagement für die Tätigkeit kompensiert werden. Gravierende Schwächen sollten zu einer Ablehnung führen.

Qualifizierte Mitarbeiter können durch Neueinstellungen aber auch durch Versetzungen innerhalb des Unternehmens gewonnen werden.

Einführung Innovationen Kap. 5.5

Neben den oben aufgeführten Faktoren ist zusätzlich noch darauf zu achten, daß innovative Mitarbeiter eingestellt werden, da dadurch die Einführung von Innovationen erleichtert wird. Es ist darauf zu achten, Problemlöser zu gewinnen und keine Problemmacher oder »Bedenkenträger«. Das Problem ist, im Bewerbungsgespräch herauszufinden, in welche Kategorie ein Bewerber gehört.

Generell gilt die Leitlinie: Lieber weniger gute Mitarbeiter, als viele durchschnittliche Mitarbeiter.

Um das Risiko von Fehleinstellungen von Berufsanfängern zu vermeiden, sollte ein frühzeitiger Kontakt zu potentiellen zukünftigen Mitarbeitern hergestellt werden.

winnung von ufsanfängern

Dies kann durch folgende Maßnahmen erreicht werden:

- Aktiven Kontakt zu Universitäten und Fachhochschulen herstellen oder halten.
- Durchführung von Studien- und Diplomarbeiten im eigenen Unternehmen.
- Anbieten von Praktikanten- und Werkstudentenplätzen.
- Anbieten von Werksbesichtigungen.
- Aushang neuer Stellen am schwarzen Brett von Universitäten und Fachhochschulen.

Hat der Student seine Studien- oder Diplomarbeit im eigenen Unternehmen durchgeführt, dann kennt sowohl der Software-Manager den Studenten als auch der Student das Unternehmen. Dadurch wird für beide Seiten das Risiko minimiert, eine falsche Entscheidung zu treffen. Bewirbt sich dagegen ein potentieller Mitarbeiter auf eine Stellenausschreibung, dann erfolgt das »Kennenlernen« in einem mehrstündigen Bewerbungsgespräch. Das Risiko für Fehleinschätzungen ist dabei viel größer.

Generell ist es erforderlich, eine mittelfristige Personalplanung vorzunehmen, um gezielt und ohne Termindruck geeignete Mitarbeiter für zukünftige Aufgaben zu finden.

Arbeitskonditionen

Um gute Mitarbeiter zu gewinnen und zu halten, ist es notwendig, attraktive Konditionen in der richtigen Kombination zu bieten:

- Interessante Arbeitsaufgaben entsprechend der Qualifikationen, Eignungen und Neigungen
- Flexible Arbeitszeiten (Gleitzeit)
- Gehalt (auch nicht-finanzielle Anreize)
- Arbeitsumgebung (Einzelzimmer)
- Arbeitsausstattung (CASE)
- Weiterbildungsangebot
- Teamarbeit
- Betriebsklima
- Unternehmenskultur

Teambildung Abschnitt 5.3

Aufgabe eines Software-Managers ist es nicht nur, geeignete Personen auszuwählen, er muß insbesondere darauf achten, daß ein gutes Team entsteht.

4.2.2 Integration neuer Mitarbeiter

Neue Mitarbeiter müssen in die Organisation integriert werden. Sie müssen mit den Entwicklungsrichtlinien, -methoden und -werkzeugen vertraut gemacht werden.

Der Manager ist für die Einführung neuer Mitarbeiter ins Unternehmen und die Vorstellung des Unternehmens gegenüber dem Mitarbeiter verantwortlich. Große Unternehmen haben oft ein mehrtägiges Einführungsprogramm für neue Mitarbeiter.

Orientierungsprogramme umfassen die wichtigsten Aspekte eines Unternehmens einschließlich ihrer Geschichte, ihrer Produkte und Dienstleistungen, ihrer Unternehmenskultur, ihrer Organisation und ihrer Leistungen für die Mitarbeiter usw.

4.2.3 Weiterbildung und Training von Mitarbeitern

»›Auf keiner Gehaltsliste eines europäischen, amerikanischen oder japanischen Unternehmens ist heute noch Platz für Mitarbeiter, die sich nicht dem lebenslangen Lernen verpflichten‹, behauptet der amerikanische Unternehmensberater Tom Peters. ›Lernen muß als normaler Bestandteil des Berufslebens akzeptiert werden‹« /Schwertfeger 93, S. 6/.

Diese Aussage gilt ganz besonders für die Software-Technik. Durch die hohe Innovationsgeschwindigkeit, die zu einer Verdoppelung des

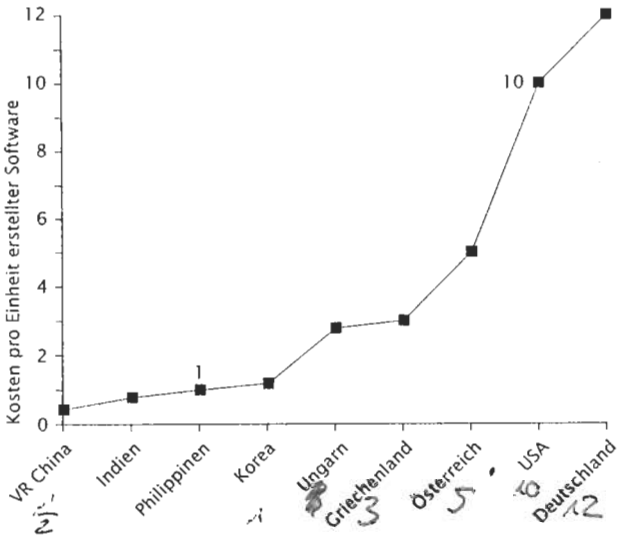
Wissens alle vier Jahre führt, ist eine ständige Weiterbildung erforderlich.

Abschnitt 1.3

Da die Qualifikation der Mitarbeiter stark die Produktivität beeinflußt, kann sich mangelnde Qualifikation schnell zum Engpaßfaktor eines Unternehmens entwickeln. Das Halten und Steigern der Mitarbeiterqualifikation wird immer stärker zur Existenzfrage. Da Deutschland niemals mit den Arbeitslöhnen der Entwicklungsländer konkurrieren kann, lassen sich entscheidende Produktivitätssteigerungen nur durch Wissen und hochgradige Automation erreichen.

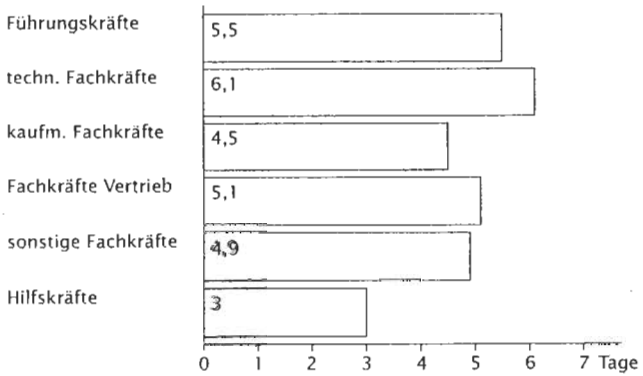
4.2-1: Kosten die Software-Produktion ber 92, S. 64/

Abb. 4.2-1 zeigt die Kosten für die Software-Produktion in verschiedenen Ländern im Vergleich zu Deutschland.



Die durchschnittlichen Weiterbildungstage nach Mitarbeitergruppen zeigt Abb. 4.2-2.

Abb. 4.2-2: chschnittliche rbildungstage h Mitarbeiter-gruppen laisberger 93, S. 15/



Für den Bereich der Software-Technik sind sechs Tage Weiterbildung im Jahr zu wenig. Nach meiner Erfahrung benötigen Mitarbeiter in der Software-Technik pro Jahr zwei bis drei Wochen Vollzeit-Training, um den Stand der Technik auf dem jeweiligen Arbeitsgebiet zu halten. Werden neue Methoden und Werkzeuge eingeführt, dann ist dafür zusätzliche Zeit nötig.

Defizite

Heute wird in der Software-Technik zu wenig Zeit für die Weiterbildung aufgewendet. Außerdem erfolgt eine Trainingsmaßnahme oft ad hoc. Meist werden nur unumgängliche Produktschulungen vorgenommen, z.B. die Bedienung von neuen Compilern oder CASE-Systemen. Die Ausbildung in Methoden und Konzepten unterbleibt häufig oder kommt zu kurz.

Kapitel III 3.1

Die Bedeutung der Schulung spiegelt sich auch in der ISO 9000, Teil 3: »Qualitätsmanagement- und Qualitätssicherungsnormen.«

Dort heißt es zum Thema Schulung:

»Der Lieferant sollte Verfahren zur Ermittlung des Schulungsbedarfs einführen und aufrechterhalten und für die Schulung aller Mitarbeiter sorgen, die mit qualitätsrelevanten Tätigkeiten betraut sind. Personal, welches eine ihm speziell zugeordnete Aufgabe ausführt, muß auf der Basis einer angemessenen Ausbildung, Schulung und/oder Erfahrung entsprechend den gestellten Forderungen qualifiziert sein.

Die zu vermittelnden Themen sollten unter Berücksichtigung der verwendeten spezifischen Werkzeuge, Techniken, Methoden und Rechnerhilfsmittel für die Entwicklung und das Management des Softwareprodukts festgelegt werden. Es kann auch erforderlich sein, die Schulung von Fertigkeiten und Vermittlung von Kenntnissen auf dem betreffenden speziellen Softwaregebiet einzuschließen. Zweckentsprechende Aufzeichnungen über die Schulung und Erfahrung sollten aufbewahrt werden.« /ISO 9000, Teil 3, 92, S. 30f./.

Um ein Qualitätsmanagementsystem zertifiziert zu bekommen, werden zum Schulungsbereich folgende Fragen gestellt /DGQ 93, S. 135f./:

ISO 9000-Zertifizierung

- Sind Zuständigkeiten und Verfahren für die Personalschulung schriftlich festgelegt und wird eine angemessene Fortbildung durchgeführt? (Gibt es ein differenziertes Schulungsprogramm?)
- Wie und durch wen wird der Schulungsbedarf aufgabenbezogen ermittelt und gibt es darüber schriftliche Unterlagen?
- Welches Programm für die durchzuführenden Schulungen und Qualifikationen gibt es?
- Welche Aufzeichnungen über durchgeführte Schulungen und Qualifikationen gibt es?
- Wird durch ein Verfahren sichergestellt, daß Personen mit besonderen Aufgaben durch angemessene Ausbildung, Schulung und/oder Erfahrung, soweit zutreffend, für die Aufgaben qualifiziert sind?

- Gibt es für Mitarbeiter mit qualitätsbezogenen Tätigkeiten schriftlich festgelegte Forderungen an:
  - **a** die Ausbildung?
  - **b** die praktische Erfahrung (Training)?
  - **c** die formellen Qualifikationen für Qualitätsmanagement-Tätigkeiten
- Sind die Unternehmensleitung und alle Führungskräfte in das Qualitätsmanagement-Fortbildungsprogramm mit einbezogen?
- Wie erfolgt nachweislich eine arbeitsplatzbezogene Unterweisung durch Vorgesetzte in qualitätsbezogene Aufgaben?
- Welche Maßnahmen zur Motivation und Förderung des Qualitätsbewußtseins gibt es?
- Wie erfolgt nachweislich eine qualitätsbezogene Unterweisung des Personals bei Einführung neuer Verfahren?
- Gibt es qualitätsbezogene Einweisungs- und Unterweisungsprogramme für Mitarbeiter bei Neueinstellung oder Umbesetzung und bei Einführung neuer oder geänderter Verfahren oder Abläufe?

blembereiche

Generell kann ein Software-Manager nicht davon ausgehen, daß alle Mitarbeiter selbst etwas für ihre Weiterbildung tun:

»Die Statistik über das Lesen ist besonders enttäuschend: Der durchschnittliche Software-Entwickler besitzt kein einziges Buch zu dem Thema seiner Arbeit und hat nicht einmal eines gelesen.« /DeMarco, Lister 91, S. 14/.

Management-sbildung von ufsanfängern

Ein spezielles, oft nicht beachtetes Problem ist die Managementqualifikation von Berufsanfängern. Berufsanfänger übernehmen oft bereits nach kurzer Zeit Projektleiteraufgaben oder werden Gruppenleiter. Von der Universität oder der Fachhochschule sind sie auf solche Aufgaben nicht vorbereitet. Umgekehrt beginnt in vielen Unternehmen die Managementausbildung erst, wenn man Abteilungsleiter ist. Ziel muß es aber sein, die Mitarbeiter frühzeitig auf diese Aufgaben durch Schulung vorzubereiten.

Fachliche ıalifikationen der Manager

Umgekehrt hat sich gezeigt, daß Software-Manager immer weniger fachliche Weiterbildung betreiben, je höher sie in der Hierarchie aufsteigen. Ziel eines Software-Managers kann es nicht sein, im Detail genauso viel zu wissen wie seine Mitarbeiter. Er muß sich vielmehr auf die strategischen Gesichtspunkte konzentrieren, er muß wissen, welche Herausforderungen auf ihn zukommen und wie sie nach rechtzeitigem Erkennen bewältigt werden können.

Häufig fehlt Software-Managern aber dieses strategische, fachliche Wissen, so daß sie in ihrem Urteil auf Berater angewiesen sind.

mpfehlungen

Für die Weiterbildung und das Training seiner Mitarbeiter sollte ein Software-Manager folgendes tun:

- Weiterbildung als Investition in die Gegenwartsicherung des Unternehmens ansehen.
- Mittelfristiges Weiterbildungskonzept erarbeiten und durchführen.

- Mitarbeiter in der Softwai
  weiterbilden.
- Moderne Lerntechniken, wie *Computer Based Training* (CBT), einsetzen, um das individuelle Lernen sowie das Lernen am Arbeitsplatz zu ermöglichen.
- Berufsanfänger rechtzeitig auf Managementaufgaben vorbereiten.
- Software-Manager auf geeignete fachliche Seminare schicken.
- Mitarbeiter mit verschiedenen Schwerpunktgebieten einstellen.
- Attraktive Fachlaufbahn bieten.
- Nicht darauf vertrauen, daß Mitarbeiter sich selbst weiterbilden.
- Eigeninitiative bei der Weiterbildung unterstützen.
- Weiterbildung als Teil der Personalentwicklung betrachten.

4.2.4 Personalentwicklung

Ziel der Personalentwicklung ist es, für die Durchführung der aktuellen und der zukünftigen Entwicklungsaufgaben in ausreichender Qualität und Quantität Personal zu wirtschaftlichen Kosten zur Verfügung zu stellen. In der Software-Technik gibt es für die Personalentwicklung zwei besondere Problembereiche, denen besondere Aufmerksamkeit zu widmen ist:

- Fluktuation von Mitarbeitern und
- Qualifikationsprobleme bei *nicht* fortbildungsfähigen Mitarbeitern.

Fluktuation

Die Software-Technik-Branche hat – zumindest in Hochkonjunkturzeiten – mit einer hohen Fluktuation zu kämpfen. In den USA bleiben Mitarbeiter von großen Software-Projekten im Durchschnitt 12 bis 30 Monate im Entwicklungsteam. Da große Software-Projekte oft länger als drei Jahre dauern, muß ein solches Projekt ein oder mehrere komplette Personalwechsel überleben /Scacchi 89, S. 62/. Folgende Lösungsmöglichkeiten bieten sich für dieses Problem an:

- Gute personenunabhängige Dokumentation (standardisiert, abgenommen, computergestützt).
- Unabhängige Qualitätssicherung.
- Attraktive Arbeitsbedingungen.
- Realistische Erwartungen bei der Personalplanung. Ein Berufsanfänger verläßt in der Regel nach zwei bis drei Jahren seinen ersten Arbeitgeber.
- Gute Personalförderung einschließlich Führungs- und Fachlaufbahnmöglichkeiten.
- Mittelfristige Personalplanung.

nicht fortbildungsfähige Mitarbeiter

Durch die hohe Innovationsgeschwindigkeit in der Software-Technik gibt es Qualifikationsprobleme bei nicht fortbildungsfähigen Mitarbeitern.

Manche Mitarbeiter kommen aus fachfremden Berufen und wurden dann zum Assembler-Programmierer umgeschult. Es erfolgte dann oft eine Weiterqualifikation zum COBOL-Programmierer. Der Versuch

einer weiteren Ausbildung in Richtung Datenbanken, moderne Sprachen und Methoden führte meistens zu einer Überforderung.
Das führt zu zwei Managementproblemen:

- Diese Mitarbeiter können keine Systeme mehr entsprechend dem Stand der Technik entwickeln.
- Es treten Akzeptanzprobleme auf, wenn neue, hochqualifizierte Mitarbeiter in ein solches Team, u.U. sogar mit Leitungsfunktion, integriert werden sollen.

Ist es bereits soweit gekommen, dann hat die Personalentwicklung in der Vergangenheit versagt und die Software-Entwicklung in eine Sackgasse geführt. Für diese Sackgasse gibt es zwei Lösungsalternativen:

1 Die alten Sprachen und Methoden werden beibehalten. Man ist sich darüber im klaren, daß ein Produktivitäts- und Qualitätsverlust eintritt und neue Mitarbeiter abgeschreckt werden.
2 Nicht fortbildungsfähige Mitarbeiter pflegen die vorhandene Software oder werden für andere Aufgaben eingesetzt. Neue Mitarbeiter entwickeln neue Systeme entsprechend dem Stand der Technik.

Die zweite Alternative kann nur dann sozialverträglich umgesetzt werden, wenn dem Management rechtzeitig das Problem bekannt ist und frühzeitig gegengesteuert werden kann.

Fachlaufbahn Aufstiegshierarchie für Fachkräfte bzw. Experten, meist beginnend ab Abteilungsreferent. Mit jeder Position innerhalb der Fachlaufbahn ist fachliche Verantwortung verbunden, im Umfang abhängig von der Höhe der Hierarchie. Im Gegensatz zur →Führungslaufbahn ist mit der Fachlaufbahn keine Personalverantwortung verknüpft. Oft entspricht jeder Position in der Führungslaufbahn eine Position in der Fachlaufbahn.

Führungslaufbahn Aufstiegshierarchie für Führungskräfte bzw. Manager, meist beginnend ab Gruppen- oder Abteilungsleiter. Mit jeder Position innerhalb der Führungslaufbahn sind Personalverantwortung einschließlich disziplinarischer Befugnisse verbunden, im Umfang abhängig von der Höhe der Hierarchie (→ Fachlaufbahn).

Die Qualifikation, die Motivation und das Engagement der Mitarbeiter haben den entscheidenden Einfluß auf die Produktivität einer Software-Entwicklung. Jeder Software-Manager muß daher das Ziel haben, gute Mitarbeiter einzustellen, sie in die vorhandene Organisationsstruktur zu integrieren, sie entsprechend den Erfordernissen weiterzubilden und zu trainieren, sie zu fördern und zu entwickeln.

Bei der Auswahl von Mitarbeitern ist darauf zu achten, daß sie über die allgemeinen Qualifikationen verfügen, die für eine Tätigkeit in der Software-Technik erforderlich sind. Je nach vorgesehenem Einsatzgebiet sollten sie auch über die notwendigen speziellen Qualifikationen verfügen. Da nicht alle Mitarbeiter Führungskräfte mit Personalverantwortung sein können bzw. wollen, ist es für die Motivation der Mitarbeiter wichtig, neben einer Führungslaufbahn auch

eine attraktive Fachlaufbahn mit Aufstiegsmöglichkeiten anzubieten.
Im Personalbereich muß ein Software-Manager auf folgende kritische Gebiete besonders achten:

- Horizontale Spezialisierung anstreben
- Fachlaufbahn etablieren
- Innovative Problemlöser einstellen
- Weniger, aber gute Mitarbeiter auswählen, statt viele mittlere
- Attraktive Arbeitskonditionen bieten
- Frühzeitigen Kontakt zu potentiellen zukünftigen Mitarbeitern herstellen (Studien- und Diplomarbeiten)
- Mittelfristige Personalplanung vornehmen
- Neue Mitarbeiter integrieren
- Für permanente und ausreichende Weiterbildung sorgen
- Berufsanfänger rechtzeitig auf Managementaufgaben vorbereiten
- Manager auch weiterhin fachlich qualifizieren
- Förder- und Entwicklungsplan für jeden Mitarbeiter zusammen mit dem Mitarbeiter aufstellen
- Personalfluktuationen antizipieren
- Qualifikationsprobleme von Mitarbeitern rechtzeitig erkennen und durch geeignete Maßnahmen mittelfristig sozialverträglich lösen.

/DeMarco, Lister 91/
DeMarco T., Lister T., *Wien wartet auf Dich! Der Faktor Mensch im DV-Management*, München: Carl Hanser Verlag 1991, 216 Seiten, Originalausgabe: *Peopleware*, New York: Dorset House Publishing Co. 1987, 188 Seiten
Die Autoren beschreiben die menschlichen Einflußfaktoren auf die Software-Entwicklung. Ein Muß für jeden Software-Manager!

Zitierte Literatur

/DGQ 93/
Deutsche Gesellschaft für Qualität (Hrsg.), *Audits zur Zertifizierung von Qualitätsmanagementsystemen*, Berlin: Beuth-Verlag, 1993
/IKOSS 94/
IKOSS, Aachen, persönliche Mitteilung
/ISO 9000, Teil 3, 92/
Qualitätsmanagement- und Qualitätssicherungsnormen, Leitfaden für die Anwendung von ISO 9001 auf die Entwicklung, Lieferung und Wartung von Software, Beuth-Verlag, Berlin, Juni 1992
/Maisberger 89/
Maisberger P., *Computeranwender gefragt wie nie zuvor*, in: Office Management, 11/1989, S. 20
/Maisberger 93/
Maisberger P., *Das lernende Unternehmen*, in: Digital Magazin, München, Nr. 3, Okt. 1993, S. 14–15
/Naisbitt 90/
Naisbitt J., *Megatrends 2000*, Düsseldorf: Econ-Verlag, 1990
/Office Management 89/
Office Management, 11/1989
/Ogger 92/
Ogger G., *Nieten in Nadelstreifen*, München: Droemersche Verlagsanstalt, 1992
/Scacchi 89/
Scacchi W., *The USC System Factory Project*, in: ACM SIGSOFT, Jan. 1989, p. 61ff.

/Schwertfeger 93/
Schwertfeger B., *Mittelpunkt Mensch*, in: Digital Magazin, München, Nr. 3, Okt. 1993, S. 6–8
/Thayer 87/
Thayer R.H., *Software-Engineering Project Management – A Top-Down View*, in: Software Engineering Project Management, Los Alamitos: IEEE Computer Society Press 1990, pp. 15–53
/V-Modell 97/
Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell, Teil 3: Handbuchsammlung, Allgemeiner Umdruck Nr. 250/1, Juni 1997, BWB IT 15, Koblenz
/Weber 92/
Weber H., *Die Software-Krise und ihre Macher*, Berlin: Springer-Verlag 1992

Muß-Aufgabe *15 Minuten*

1 *Lernziel: Angeben können, welche Qualifikationen für die beschriebenen Rollen erforderlich sind.*
In Ihrem Unternehmen sind folgende Positionen neu zu besetzen: Systemanalytiker, Software-Entwerfer, Programmierer, Qualitätssicherer und Software-Ergonom. Sie haben fünf qualifizierte Kandidaten. Durch Personalgespräche können Sie die Personen folgendermaßen charakterisieren:

a Der erste Kandidat hat ein hohes Abstraktionsvermögen. Aufgrund seines Mathematikstudiums ist er gewohnt, exakte Ergebnisse zu liefern. Weiterhin ist er ein sehr einfallsreicher Mensch.

b Die Stärken des zweiten Kandidaten liegen im Umgang mit anderen Menschen. Neben seiner Ausbildung in der Software-Technik hat er sich mit Arbeitsabläufen und der Mensch-Maschine-Kommunikation beschäftigt.

c Dieser Kandidat ist dafür bekannt, daß er versucht, den Dingen auf den Grund zu gehen. Dafür nimmt er sich Zeit und gibt nicht auf, bevor er eine Lösung gefunden hat.

d Die Person kommt ursprünglich aus dem Anwendungsgebiet, für das sie Software entwickelt. Sie hatte kein Problem, sich in anderen Bereichen schnell zurechtzufinden und deren Terminologie zu erlernen. Neben der Fähigkeit zu abstrahieren, ist das befragende Gespräch mit anderen Menschen eine weitere Stärke.

e Die Fähigkeit, die Übersicht zu behalten und global zu denken, sind die Pluspunkte des letzten Kandidaten. Er ist ferner in der Lage, allgemeine Situationen in konkrete umzuformen.

Besetzen Sie die Positionen!

Muß-Aufgabe *5 Minuten*

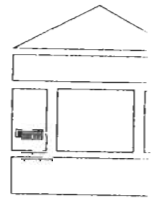
2 *Lernziel: Wissen, wie die Kosten der deutschen Software-Produktion im Verhältnis zu anderen Ländern aussehen.*
Die Erstellung eines speziellen Programms kostete in Griechenland umgerechnet 25 000 DM. Mit welchen Kosten ist für die Erstellung des gleichen Programms in Österreich, China, Deutschland und den USA zu rechnen?

Muß-Aufgabe *5 Minuten*

3 *Lernziel: Die Bedeutung von Personalfragen anhand von Beispielen aufzählen können.*
Zwei Teams entwickeln je ein Programm. Ein Projekt-Team war in der Lage, in kürzerer Zeit ein besseres Produkt zu liefern. Nennen Sie zwei Gründe in der Personalplanung, an denen das schlechte Abschneiden des einen Teams gelegen haben könnte.

Weitere Aufgaben befinden sich auf der beiliegenden CD-ROM.

5 Leitung



wissen

- Angeben können, über welche Eigenschaften ein Software-Manager verfügen soll und welchen Herausforderungen er sich stellen muß.
- Die Charakteristika von Teams aufzählen, ihre Stärken und Schwächen nennen sowie angeben können, wodurch die Teambildung gefördert bzw. verhindert wird.
- Eigenschaften teamorientierter Manager und teamfähiger Mitarbeiter aufführen können.
- Wissen, was man unter Kreativität, Kreativitätstechniken, Metaplan-Technik, Pinnwand und Flip-Chart versteht.
- Leitungsaktivitäten eines Software-Managers aufzählen und erläutern können.
- Führung, Führungsstil und *Management-by-...* erklären und die beschriebenen *Management-by*-Methoden erläutern können.
- Die Kreativitätstechniken *Brainstorming* und *Brainwriting* erläutern können.
- Kreativitätstechniken klassifizieren können.
- Die sechs Schritte des Risikomanagements angeben und erläutern sowie zugeordnete Techniken aufzählen können.
- Die *Top ten*-Risiken einer Software-Entwicklung und zugeordnete Risikomanagement-Techniken nennen und erläutern können.
- Für vorgegebene Szenarios geeignete Leitungsaktivitäten eines Software-Managers identifizieren, begründen und anwenden können.
- Risikomanagement anhand von Beispielen durchführen können.

Prof. Dr. Barry Boehm
Pionier auf dem Gebiet des Softw Managements, Erfinder des COCOMO-Kosten schätzmodells u des Spiralmodell grundlegende Arbeiten zur Soft ware-Produktivita und zum Risikomanage ment; Promotion in Mathematik, Universität von Kalifornien, Los Angeles, von 1973–1989 bei TRW, von 1989–1992 beim US-Verteidigungs ministerium, heut TRW *Professor of Software Enginee ing, University of Southern Californ Fellow of the IEEE.*

5.1 Grundlagen

Mitarbeiter anzuleiten und zu führen ist eine der anspruchsvollsten und zeitlich aufwendigsten Aufgaben eines Managers. Neben der formalen Autorität trägt die natürliche Autorität eines Managers wesentlich dazu bei, seine Mitarbeiter so zu motivieren, daß sie ihr Bestes geben, um die angestrebten Ziele zu erreichen.

Aktivitäten

Zu den Leitungsaktivitäten gehören neben der Führung und Beaufsichtigung von Mitarbeitern auch das Delegieren von Kompetenzen, das Koordinieren von Aktivitäten, das Unterstützen der Kommunikation, das Lösen von Konflikten und die Einführung von Innovationen.

Kapitel 5.5

Zur Personalführung gibt es eine umfangreiche Managementliteratur und eine Vielzahl von Managementphilosophien, auf die hier nicht eingegangen wird (siehe z.B. /Raidt 85/, /Kolb 95/). In den folgenden Abschnitten wird sich weitgehend auf software-spezifische Fragestellungen konzentriert.

qualifiziertes Personal

Eine Besonderheit des Software-Managements besteht darin, hochqualifiziertes Personal zu führen.

Teambildung

Dieses Personal muß in einem oder mehreren Teams konstruktiv und kreativ zusammenarbeiten. Die Zusammenstellung geeigneter Teams stellt daher einen wichtigen Erfolgsfaktor dar.

Kreativität

Da selbst bei routinemäßig ablaufenden Software-Enwicklungen neue Ideen und Lösungsmöglichkeiten erforderlich sind, spielen Kreativitätstechniken, ihre Vermittlung und Anwendung eine große Rolle.

Risiko-Management

Jeder Software-Manager hat seine »Lieblingsthemen«, die er mit seinen Mitarbeitern besonders gern bis ins letzte Detail diskutiert. In diese Themen steckt er einen großen Teil seiner Management-Aktivitäten, obwohl diese Themen für den Entwicklungserfolg meist nicht ausschlaggebend sind. Er übersieht dabei oft die wirklich riskanten Gebiete. Um dies zu vermeiden, gibt es das Risiko-Management, dessen Ziel es ist, dem Manager Risiken systematisch bewußt zu machen.

ftware-Manager

Personalführung kann nicht nur darin bestehen, alles so fortzuführen, wie es bisher war. Ein Software-Manager muß in hohem Maße strategisch denken und die Fähigkeit besitzen, Visionen zu entwikkeln. Jeder Software-Manager muß in der Lage sein, seine eigene Arbeit so zu organisieren, daß er sich genügend Zeit und Freiraum nimmt, um die technischen Trends zu analysieren und in Strategien für die eigene Firma umzusetzen.

Ein Manager wird dafür bezahlt, daß er Chancen nutzt und Risiken vermeidet. Chancen kann man aber nur nutzen, wenn man bereit ist, die damit verbundenen Risiken einzugehen.

Meiner Erfahrung nach nutzt gerade das deutsche mittlere Management zu wenig seinen Spielraum, um Chancen wahrzunehmen.

Anders ausgedrückt: Das Mittelmanagement ist oft risikoscheu und versucht, Entscheidungen an die nächsthöhere Managementebene zurückzudelegieren. Das ist der Tod jeder Innovation.

Beispiel

Kunden reklamieren die schlechte Software-Qualität eines Software-Hauses. Der Leiter der Software-Entwicklung erhält von der Geschäftsleitung den Auftrag, eine Qualitätssicherung zu etablieren. Obwohl dem Leiter der Software-Entwicklung hundert Mitarbeiter unterstehen, sagt er der Geschäftsleitung, er könne nur dann eine Qualitätssicherung aufbauen, wenn er fünf neue Stellen genehmigt bekomme. Solange dies nicht der Fall sei, trage die Geschäftsleitung die Verantwortung für die mangelhafte Qualität (Rückdelegation von Verantwortung).

Ein Software-Manager darf in einer so innovativen Branche wie der Software-Technik auf neue Herausforderungen nicht nur reagieren, sondern er muß selbst rechtzeitig agieren.

fachliche Herausforderungen

Folgende fachliche Herausforderungen stellen sich heute für einen Software-Manager:

- Umstieg auf die objektorientierte Software-Entwicklung,
- Umstellung auf *Client-Server*-Architekturen,
- Wiederverwendung technisch und organisatorisch ermöglichen,
- Metriken einführen, auswerten und zur Prozeßsteuerung verwenden,
- CASE-Umgebungen einführen bzw. auf dem neuesten Stand halten.

Mangement-Herausforderungen

Folgende Management-Herausforderungen stellen sich heute:

- ISO 9000-Zertifizierung erreichen,
- Kontinuierliche Prozeß- und Qualitätsverbesserung anstreben,
- Liberale, innovationsfreundliche Firmenkultur entwickeln,
- Innovationen initiieren und fördern,
- Mit flachen Hierarchien auskommen,
- Kundenorientiertes Denken und Handeln bewirken.

Den meisten dieser Herausforderungen sind in diesem Buch eigene Kapitel gewidmet.

Um heute im Markt Erfolg zu haben, ist eine kundenorientierte Sicht erforderlich. Abb. 5.1-1 zeigt die Leitbilder von drei System-Häusern.

5.2 Hochqualifizierte Mitarbeiter führen

Ein besonderes Kennzeichen der Software-Entwicklung ist es, daß die Mitarbeiter in der Regel hoch qualifiziert sind und oft über einen Hochschulabschluß verfügen. Diese Besonderheit muß sich auf den Führungsstil auswirken.

Abb. 5.1-1: Kundenorientierte Leitbilder von drei System-Häusern

Firma IKOSS

Unser Kunde

- Unser Kunde ist die wichtigste Person für unser Unternehmen, gleich, ob er uns schreibt oder mit uns spricht.
- Unser Kunde findet leichter einen neuen Auftragnehmer als wir einen neuen Kunden.
- Unser Kunde stört uns nicht bei der Arbeit, er gibt sie uns.
- Unser Kunde stellt uns Aufgaben. Unser Ziel ist es, sie gewinnbringend für ihn und für uns zu lösen.
- Unser Kunde möchte von uns beraten werden. Wir messen weder unseren Intellekt noch streiten wir mit ihm. Niemand hat je einen Streit mit einem Kunden gewonnen.
- Unser Kunde ist kein Außenstehender, sondern der Mittelpunkt unserer Arbeit. Wir tun ihm keinen Gefallen, indem wir ihn bedienen, sondern er uns einen, wenn er uns Gelegenheit dazu gibt.

Firma Schleupen

- Unsere Kunden erleben unsere kompetenten Mitarbeiter als zuverlässige und zielorientierte Partner.
- Unser Handeln gegenüber Kunden und Mitarbeitern ist durch Fairness bestimmt.
- Unser Handeln konzentrieren wir auf Kunden in unseren Zielmärkten.
- Wir geben Impulse, verändern Märkte und setzen Standards.
- Unsere Lösungen orientieren sich an den Anforderungen unserer Kunden.
- Die Anforderungen unserer Kunden nach einer umfassenden Unterstützung ihrer organisatorischen Abläufe erfüllen wir durch ein Angebot von der Analyse bis zum Service.
- Die kontinuierliche Qualität unserer Arbeit messen wir an unseren langjährigen Kundenbeziehungen.
- Engagierten und kreativen Mitarbeitern bieten wir einen zukunftssicheren Arbeitsplatz mit Herausforderung und Kompetenz.

Firma Hauni

- Unser wichtigstes Ziel, die Zufriedenheit unserer Kunden, wollen wir durch Produkte und Dienstleistungen von höchstem Nutzen erreichen.
- Wir wollen im Markt für Tabaktechnologie das führende Unternehmen und Schrittmacher des technischen Fortschritts sein.
- Nur mit motivierten Mitarbeiterinnen und Mitarbeitern erreichen wir unseren Unternehmenserfolg. Dafür wollen wir das erforderliche Umfeld schaffen und ihnen in allen Funktionen die Auswirkungen ihrer Tätigkeit auf die Kundenzufriedenheit sichtbar machen.
- Wir wollen die Zukunft unseres Unternehmens langfristig sichern. Die dazu erforderliche unternehmerische und finanzielle Unabhängigkeit muß durch nachhaltige Erträge gewährleistet werden.
- Von unseren Lieferanten, zu denen wir faire und vertrauensvolle Beziehungen pflegen, erwarten wir erstklassige Produkte und Dienstleistungen.
- Wir sind Bestandteil einer freien sozialen Marktwirtschaft in einem ökologischen System und fühlen uns der Gesellschaft und der Umwelt gegenüber verpflichtet.

Führung

Unter **Führung** versteht man die Einwirkung auf Mitarbeiter, so daß vorgegebene Ziele erreicht werden. Wie die Form dieser Einwirkung aussieht, wird durch den **Führungsstil** festgelegt. Im Laufe der letzten 30 bis 40 Jahre bildeten sich verschiedene Führungsstile bzw. Führungsphilosophien heraus. Man bezeichnet sie als ***Management-by***-Methoden. Eine Auswahl dieser Methoden, die für hochqualifizierte Mitarbeiter geeignet sind, wird im folgenden kurz skizziert.

Führungsstil

*Mangement-by-*Methoden

Management by Objectives (MbO)

Literatur: /Raidt 85/

Führung durch Zielsetzung erfordert zuerst eine Festlegung der Unternehmensziele. Aus diesen werden dann die Ziele der einzelnen Bereiche und Abteilungen abgeleitet. Führungskräfte geben ihren Mitarbeitern in regelmäßigen Abständen, z.B. jährlich, operationale Ziele vor oder vereinbaren sie mit ihnen. Alle Entscheidungen, die

Ziele

zur Erreichung des Ziels nötig sind, liegen in der Verantwortung des jeweiligen Mitarbeiters. Die Beurteilung der Mitarbeiterleistung erfolgt am Periodenende durch Vergleich der abgesprochenen Ziele mit den tatsächlich erreichten Zielen.

Führung durch Zielsetzung ist von folgenden Voraussetzungen abhängig:

- Mitarbeitern wird ein bestimmter Aufgabenbereich delegiert.
- Aufgaben und Kompetenzen jeder Stelle sind in Stellenbeschreibungen festgelegt.
- Mitarbeitern wird erläutert, wie ihre Ziele in übergeordnete Ziele eingebettet sind, was von der nächsten Periode erwartet wird und welche Unterstützung die Führungskraft bereitstellt.
- Festlegung, woran bzw. wie die Leistung gemessen wird.

Führung durch Zielsetzung ist einer der am meisten verbreiteten Führungsstile.

Management by Results

Ergebnisse

Dezentrale Führungsorganisation, bei der die Ergebnisse vorgegeben, gemessen und kontrolliert werden. Die delegierten Führungsaufgaben werden über die Ergebnisse kontrolliert. Folgende Grundsätze sind zu beachten:

- Die Abteilungen konzentrieren sich auf wenige, möglichst quantitative Entscheidungsmaximen.
- Die Ziele sollen motivieren.
- Die Führungskräfte werden auf allen Hierarchieebenen ausreichend über die von ihnen erwarteten Verhaltensweisen informiert.

Management by Delegation

Delegation

Aufgaben und die entsprechenden Befugnisse werden soweit wie möglich an die Mitarbeiter und auf untere Hierarchieebenen übertragen. Mit der Delegation sind immer geeignete Kontrollen verbunden. Voraussetzungen sind klare Aufgabendefinitionen und Kompetenzabgrenzungen.

Management by Participation

Partizipation

Führungsstil mit starker Betonung der Mitarbeiterbeteiligung an den sie betreffenden Zielentscheidungen. Es wird davon ausgegangen, daß die Identifikation der Mitarbeiter mit den Unternehmenszielen – und damit ihre Leistung – wächst, je stärker sie an der Formulierung dieser Ziele mitwirken.

Management by Alternatives

Alternativen

Für jedes wichtige Problem sind Alternativlösungen zu entwickeln. Erst nach der Bewertung der Alternativen wird eine Entscheidung gefällt.

Diese Methode wird durch das Spiral-Modell formalisiert.

Abschnitt 3.3.7

Ausnahmen

Management by Exception

Normal- und Routinefälle werden von der mittleren und unteren Führungsebene völlig selbständig bearbeitet und entschieden. Vorgesetzte werden nur dann zu Entscheidungen hinzugezogen, wenn Ausnahmefälle vorliegen. Dieser Führungsstil erfordert folgende Voraussetzungen:

- Klare Definition der übertragenen Aufgaben.
- Umfassende Richtlinien für die Entscheidungen der einzelnen Stellen.
- Übertragung von Vollmacht und Verantwortung.

Das Management kann sich bei diesem Führungsstil auf die kritischen Entwicklungen konzentrieren.

Abschnitt 5.5

In der Software-Technik wird dieser Führungsstil durch das Risikomanagement unterstützt.

Motivation

Management by Motivation

Die Aufgabe des Managers besteht darin, die Bedürfnisse, Interessen, Einstellungen und persönlichen Ziele der Mitarbeiter zu erkennen und sie mit den Unternehmenszielen und betrieblichen Erfordernissen zu verbinden, so daß die Mitarbeiter Spaß an der Arbeit haben.

Wie sich überdurchschnittlich erfolgreiche Führungskräfte verhalten, zeigt Abb. 5.2-1.

Abb. 5.2-1: Wie Spitzenmanager führen

Die 19 spezifischen Verhaltensweisen überdurchschnittlich erfolgreicher Führungskräfte.

Ziele und Aufgaben definieren	Beraten und unterstützen	Leistungen beurteilen	Organisationsentwicklung
1 Der erfolgreiche Manager entwickelt herausfordernde und erreichbare Ziele für seine Mitarbeiter.	**6** Äußert mehr Anerkennung als negative Kritik.	**12** Belohnt und fördert Mitarbeiter im Hinblick auf Innovations- und Risikofreudigkeit.	**16** Entwickelt Strategien und Ziele für die Organisation (Hauptabteilung, Abteilung).
2 Legt klare, spezifische Hauptaufgaben und Fähigkeiten für Mitarbeiterpositionen oder Stellen fest.	**7** Bietet Mitarbeitern Hilfestellung und Unterstützung an.	**13** Bespricht regelmäßig mit Mitarbeitern ihren Leistungsfortschritt und Zielerreichungsgrad.	**17** Führt Meetings so durch, daß Zusammenarbeit gefördert wird.
3 Erklärt Aufgaben und Projekte verständlich und gründlich.	**8** Vermittelt hohe persönliche Erwartungen auf informelle Art und Weise.	**14** Verstärkt ausgezeichnete Leistungen seiner Mitarbeiter durch finanzielle und nichtfinanzielle Anreize.	**18** Ermutigt Mitarbeiter, Aufgaben und Projekte, die sie für wichtig halten, zu entwickeln und zu übernehmen.
4 Bestimmt meßbare, beziehungsweise überprüfbare Kriterien für erforderliche Leistungen.	**9** Legt Wert auf positive zwischenmenschliche Beziehungen zu seinen Mitarbeitern.	**15** Bezieht das Gesamtbeurteilungssystem (Belohnung, Beförderung, Anerkennung) nur auf das tatsächliche Leistungsverhalten, nicht auf andere Faktoren (z.B. Dienstalter).	**19** Zeigt persönliches Engagement in der Verfolgung übergeordneter strategischer Ziele.
5 Klärt Probleme und ihre Ursachen vollständig ab, so daß Mitarbeiter sie korrigieren können.	**10** Gibt Mitarbeitern die Möglichkeit, ihre Fehler selbst herauszufinden und zu korrigieren, anstatt Probleme für sie zu lösen.		
	11 Bezieht Mitarbeiter in Zielfindungs- und Entscheidungsprozesse ein.		

Quelle: /Derschka, S.10/

Abb. 5.2-2: Mit acht Merkmalen zum Unternehmenserfolg

Drang zur Tat
Aufgaben werden zügig angepackt, ohne lange Analysen. Es wird ständig experimentiert, auch auf die Gefahr hin, Fehler zu machen.

Dicht am Kunden
Ehrgeiz, dem Kunden gute Qualität, guten Service und Verläßlichkeit zu bieten; permanenter Kundenkontakt.

Eigenständigkeit und Unternehmertum
Es gibt – unabhängig von der Unternehmensgröße – kleine operative Einheiten, die überschaubar sind und unternehmerisch agieren. Viel Entscheidungsfreiheit und Wettbewerb auf unteren Hierarchieebenen.

Produktivität durch Mitarbeiter
Den Fähigkeiten der Mitarbeiter wird vertraut. Sie werden an der Verbesserung von Arbeitsabläufen und Produkten beteiligt. Dadurch wird ihre Motivation erhöht, und aus durchschnittlichen Mitarbeitern werden gute Mitarbeiter.

Von Werten geleitet
Alle Aktivitäten werden von Unternehmenswerten wie Qualität, Zuverlässigkeit, Kundenpflege durchdrungen. Sie bestimmen die Unternehmensstrategien.

»Schuster bleib bei deinen Leisten«
Nur dort, wo eigenes Know-how erfolgreich eingesetzt werden kann, erfolgen geschäftliche Aktivitäten und Firmenkäufe.

Einfache Organisationsstrukturen, kleine Stäbe
Organisationsstrukturen werden nicht perfektioniert, Stäbe sind mager ausgestattet. Das Berichtswesen konzentriert sich auf das Notwendigste. Es findet eine breite, informelle Kommunikation statt.

Führung zugleich locker und fest
Ausgewogene Mischung zentraler und dezentraler Strukturen. Viele Freiräume für Initiative und eigene Lösungswege, wenn sie im Rahmen der klar definierten Firmenziele und der strikt beachteten Firmenwerte zu Ergebnissen führen.

Quellen: /Peters, Waterman 82/, /Rußmann 83/

Ziel vieler Unternehmen ist es, einen einheitlichen Führungsstil im gesamten Unternehmen zu praktizieren. Es entsteht dann eine Firmenkultur.

Peters und Waterman haben 43 als exzellent eingestufte Unternehmen in den USA untersucht und acht Merkmale isoliert, die exzellente Firmen von durchschnittlichen unterscheiden /Peters, Waterman 82/ (Abb. 5.2-2). Diese acht Merkmale sind bei den herausragenden Firmen weitgehend komplett anzutreffen, während andere Firmen nur einige davon aufzuweisen haben – wenn überhaupt. Die Merkmale betreffen vor allem Zielsetzung, Selbstverständnis, Struktur, innere Dynamik und Firmenkultur eines Unternehmens.

Neu ist an diesen Merkmalen nichts. Es scheint aber so, daß nur die ausgezeichneten Firmen konsequent diese Merkmale befolgen. Besonders auffällig war die starke wert- und mitarbeiter-orientierte Firmenkultur.

Das Merkmal »Drang zur Tat« setzt eine offene Kommunikation voraus. Gespräche finden auf allen Ebenen und zwischen allen Ebenen statt. Höhere Linienmanager gehen häufig in die Frontbereiche, erleben mit, was geschieht, wo Probleme entstehen und vermitteln den Mitarbeitern nebenbei die Unternehmensziele *(Management by walking around).*

Bürokratische Tendenzen werden oft durch einfache Regeln bekämpft. In manchen Firmen darf keine innerbetriebliche Mitteilung länger als eine DIN A4-Seite sein.

Die Innovationskraft vieler Unternehmen entsteht durch ihre Kundennähe. Viele Anregungen für Produktverbesserungen kommen von Kunden.

5.3 Teams bilden und führen

Eine Software-Entwicklung kann nicht durch einen einzelnen Mitarbeiter durchgeführt werden, da er zuviel Zeit benötigen würde, um ein Software-System fertigzustellen. Daher muß eine Gruppe von Mitarbeitern gemeinsam an einer Aufgabe arbeiten.

Team

Ein Team stellt die ausgeprägteste Form der Gruppenarbeit dar. In einem **Team** arbeiten Mitarbeiter unterschiedlicher Qualifikationen miteinander, um eine gemeinsame Aufgabe zu erledigen.

Charakteristika der Teamarbeit

Teamarbeit ist durch folgende Charakteristika gekennzeichnet:

- Regelmäßige und kontinuierliche Kommunikation untereinander.
- Von Fall zu Fall gegenseitige Abstimmung.
- Gleichberechtigte Mitbestimmung aller Teammitglieder bei der Diskussion von Methoden, Inhalten und Zielen der Arbeit und ihrer Durchführung.
- Alle Teammitglieder sind gleichrangig und agieren auch gleichrangig.
- Verschiedene Teammitglieder übernehmen zeitweise die Führungsrolle, jeweils auf dem Gebiet, auf dem sie ihre Stärken haben.

Die Struktur eines Teams ist ein Netzwerk und keine Hierarchie. Manager sind normalerweise nicht Teil des Teams, das sie leiten.

Abb. 5.3-1: Teams – ihre Stärken und ihre Schwächen

- :: Hoher Problemlösungsgrad bei schwierigen Problemen.
- :: Verschiedene Standpunkte und Meinungen kommen zur Sprache.
- :: Das unterschiedliche »*know-how*« der Teammitglieder wird genutzt.
- :: Viele Dinge lassen sich im Team besser erledigen als alleine.
- :: Für manche Aufgaben sind Teams immer besser als ein einzelner.
- :: Die Gefahr, in eine Sackgasse zu geraten, ist geringer als bei Einzelarbeit.
- :: Teams sorgen dafür, daß alle an einem Strang ziehen.
- :: Hohe Arbeitszufriedenheit.
- :: Hohe Risikobereitschaft.
- :: Gegenseitige Anregung und Verstärkung.
- :: Vielfältige Informationen werden in relativ kurzer Zeit vermittelt.
- :: Zwischenmenschliche Bedürfnisse werden in hohem Maße befriedigt.

- = Hoher Zeit- und Kommunikationsaufwand.
- = Hoher Konformitäts- und Normierungsdruck.
- = Informationsfülle erfordert lange Diskussionen und zögert Entscheidungsfindungen hinaus.
- = Konkurrenzdenken und individuelle Profilierung können Leistung verringern.
- = Es laufen gruppendynamische Prozesse ab, die nur schwer zu kontrollieren und zu beeinflussen sind.
- = Gruppendruck kann Teammitglieder zu leistungsabträglichem Verhalten veranlassen.

Quellen: /DeMarco, Lister 91, S.141ff./, /Mantei 81/, /Berkel 84/, eigene Erfahrungen

Teams haben Stärken, aber auch Schwächen (Abb. 5.3-1). Um die Schwächen zu vermeiden, sollte meiner Erfahrung nach die Teamgröße in der Software-Technik bei drei oder vier Mitgliedern liegen.

Teamgröße 3 – 4

Besonders bewährt hat sich Teamarbeit zu Beginn der Systemanalyse und des Software-Entwurfs, da zu diesem Zeitpunkt wichtige Entscheidungen fallen.

Einsatzgebiete

Für den Software-Manager stellt sich die Frage, welche Faktoren die Teambildung fördern bzw. verhindern (Abb. 5.3-2).

- :: Team auf gemeinsames Ziel ausrichten.
- :: Team zu Erfolgen verhelfen.
- :: Elitegefühl stärken.
- :: Qualität zum Kult machen.
- :: Vielfalt ins Team bringen.
- :: Strategische Richtlinien vorgeben, keine taktischen.
- :: *Never change a winning team.*

- = Kontrolle statt Vertrauen und Autonomie.
- = Bürokratie.
- = Räumliche Trennung statt räumlicher Nähe.
- = Gleichzeitige Mitarbeit in mehreren Teams, statt nur in einem.
- = Scheintermine statt Vertrauen.

Q.: /DeMarco, Lister 91, S.141ff./

Abb. 5.3-2: Förderung/ Verhinderung der Teambildun

Teams brauchen ein gemeinsames Ziel, damit sie motiviert arbeiten. Firmenziele sind viel zu abstrakt. Daher ist es wichtig, konkrete Ziele vorzugeben, die vom Team akzeptiert werden.

gemeinsames Ziel

Teams benötigen, insbesondere in der Anfangsphase, gemeinsame Erfolge und Anerkennung. Sie müssen darin bestätigt werden, daß sie auf dem richtigen Weg sind. Als Manager sollte man daher die zu erledigende Arbeit so aufteilen, daß genügend oft Erfolgserlebnisse möglich sind.

Erfolge

Um mit sich selbst zufrieden zu sein, brauchen Mitarbeiter das Gefühl, einzigartig zu sein. Irgendwann fängt ein Team an, sich als etwas Besonderes anzusehen. Alle verspüren dasselbe Elitegefühl, man hebt sich von allen anderen ab. Egal, worin sich die Einzigartigkeit ausdrückt, sie bildet die Grundlage für die Identität des Teams.

Elite-Team

Jedes Team braucht eine Herausforderung: »Nur das Beste ist gut genug für uns«. Bekommt ein Team die Aufgabe, nur ein mittelmäßiges Produkt zu entwickeln, weil die Zeit oder das Geld fehlt, dann hat das Team keinen Anreiz, eine herausragende Leistung zu erbringen. Jeder schämt sich, an einem »Schundprodukt« mitzuarbeiten.

Qualitätskult

Es ist aber auch darauf zu achten, daß kein »*Overengineering*« betrieben wird oder Funktionen »vergoldet« werden, nur weil es dem Team Spaß macht.

Kapitel 5.5, Tab. 5.5-1

Auf der anderen Seite wird es immer wichtiger, Qualitätsbewußtsein bei jedem Mitarbeiter zu etablieren. Daher ist das Teammotto »Nur die beste Qualität ist gut genug für uns« mit besten Kräften vom Management her zu fördern.

Kapitel III 4.2

Vielfalt

Die Erfolgschancen für eine gute Zusammenarbeit im Team werden größer, wenn das Team vielfältig zusammengesetzt ist, z.B. aus Mitarbeitern und Mitarbeiterinnen, aus Endbenutzern und Entwicklern usw.

Strategie vorgeben

Der Manager sollte sich darauf beschränken, Strategien vorzugeben. Er sollte sich aber nicht in die Taktik des Teams einmischen, wie es gedenkt, die Strategie zu erfüllen.

Erfolgreiche Teams erhalten

Ein harmonisches, erfolgreiches Team sollte die Chance bekommen, auch das nächste Projekt gemeinsam zu bearbeiten, wenn es dies wünscht. Das neue Projekt startet dann bereits mit einem großen Motivationsvorsprung.

Kontrolle statt Vertrauen

Ein gut kooperierendes Team kann nicht entstehen, wenn der Manager dem Team kein Vertrauen entgegenbringt, wenn jede Entscheidung durch ihn abgesegnet werden muß, wenn er sich in technische Fragen einmischt. Teams müssen autonom arbeiten können. Dies schließt ein, Fehler zu machen und z.B. einen anderen Weg einzuschlagen als den, den der Manager gewählt hätte.

Bürokratie

Abschnitt 3.3.2

In manchen Prozeßmodellen versucht man das Risiko einer Entwicklung dadurch zu reduzieren, daß man eine Vielzahl von Dokumenten von den Software-Entwicklern fordert (siehe z.B. das V-Modell). Diese Software-Bürokratie führt dazu, daß die Entwickler meinen, wenn sie nur alle Formulare vollständig ausfüllen, dann würden sie ein gutes System erhalten. Es soll hier nicht gegen die Dokumentation argumentiert werden, sie ist im richtigen Umfang erforderlich. Beim Einsatz von CASE-Umgebungen entsteht ein großer Teil der Dokumentation sowieso automatisch. Aber: Software-Bürokratie behindert die Teamarbeit. Wichtig ist es, das Team auf ein Ziel einzuschwören. An dieses Ziel muß es glauben, und es muß spüren, daß das Management daran glaubt.

Räumliche Trennung

Sind die Teammitglieder räumlich entfernt untergebracht, dann entfallen zwanglose Gespräche, man sieht sich selten, es kann sich kein Zusammengehörigkeitsgefühl entwickeln.

Teams sollten daher räumlich nahe untergebracht sein, was nicht gleichbedeutend mit einem Raum sein muß (siehe dazu auch den Faktor Einzelzimmer im Kapitel 1.4). Sie sollten einen gemeinsamen Besprechungsraum und eine gemeinsame Teeküche haben oder die Möglichkeit haben, sich in virtuellen Räumen zu treffen.

Mitarbeit in mehreren Teams

Muß ein Mitarbeiter gleichzeitig in mehreren Teams mitarbeiten, dann ist dies für die Teambildung und die Effizienz schlecht. Er muß zu allen Teams seine Kontakte aufrecht erhalten. Folglich muß er ständig umdenken. Eingeschworene Teams entstehen nur, wenn ihre Mitglieder den größten Teil ihrer Zeit darin verbringen.

Scheintermine

Werden vom Manager Termine vorgegeben, die absolut nicht einzuhalten sind, dann sind sie unglaubwürdig. Das Team engagiert sich nicht, da solche Termine nur als Druck »von oben« betrachtet werden, die die Vertrauensbasis zerstören.

Damit Teams erfolgreich sein können, müssen sie gut geführt werden. Dazu benötigt man teamorientierte Manager. Auf der anderen Seite benötigt man teamfähige Mitarbeiter. Die Eigenschaften dieser Manager und Mitarbeiter sind in Abb. 5.3-3 zusammengestellt.

teamorientierte Manager

teamfähige Mitarbeiter

Eigenschaften...

...teamorientierter Manager

- Kompetenz bei Mitarbeitern anerkennen.
- Gewisses Maß an Freiheit und Verantwortung für bestimmte Aufgaben an Mitarbeiter übertragen.
- Vertrauensvorschuß gewähren.
- Teams sich selbst bilden lassen oder Mitspracherecht bei der Zusammensetzung einräumen.
- Administrative und organisatorische Hürden für das Team aus dem Weg räumen.
- Teams zeitweise völlig autonom arbeiten lassen.
- Teams zeitweise in Isolation »verbannen« (Hotel, abgelegenes Büro, Ferienhaus).

...teamfähiger Mitarbeiter

- Positive Einstellung zur Teamarbeit.
- Kritik- und Konfliktoleranz.
- Gegenseitige Anerkennung und Respektierung der fachlichen Qualifikation und persönlichen Integrität.
- Partnerschaftliches Verhalten.
- Fähigkeit, widersprüchliche und voneinander abweichende Informationen zu verarbeiten.
- Bereitschaft, sich voll im Team zu engagieren.
- Mit sich selbst zufrieden sein.

Q.: /DeMarco, Lister 91, S.141ff./, /Berkel 84/, eigene Erfahr.

Abb. 5.3-3: Eigenschaften teamorientierter Manager und teamfähiger Mitarbeiter

Ein eingeschworenes Team erkennt man an folgenden Merkmalen:

- Niedrige Fluktuationsrate.
- Ausgeprägtes Identifikationsbewußtsein.
- Freude an der Arbeit.
- Bewußtsein einer Elitemannschaft.

5.4 Kreativität fördern

Viele Aktivitäten in der Software-Entwicklung erfordern ein hohes Maß an Kreativität, z.B. das Modellieren des Fachkonzepts, das Entwerfen einer geeigneten Systemarchitektur, das Finden eines neuen Algorithmus. Gerade Software-Ingenieure denken oft zu rational, zu sehr in eingefahrenen Denkschablonen. Kreativitätstechniken helfen, neue Lösungswege, neue Ideen zu finden.

Kreativität

Unter **Kreativität** versteht man die Fähigkeit, Wissens- und Erfahrungselemente aus verschiedenen Bereichen unter Überwindung verfestigter Strukturen und Denkmuster zu neuen Problemlösungsansätzen bzw. zu neuen Ideen zu verschmelzen.

Originelle Ideen zeichnen sich vielfach dadurch aus, daß sie Prinzipien oder Erfahrungen aus Bereichen nutzen, die vom bearbeiteten Problemfeld weit entfernt liegen.

Kreative Menschen wenden – offenbar unbewußt – bestimmte Prinzipien an, um zu neuen Ideen zu kommen. Diese heuristischen Prinzipien, wie Assoziieren, Abstrahieren, Strukturen aus anderen Bereichen übertragen, Kombinieren, Variieren usw., helfen, eine Brücke vom Problem zu problemfremden Wissenselementen zu schlagen. Durch Anwendung dieser Prinzipien können festgefügte Denkstrukturen überwunden werden.

Kreativitäts-techniken

Kreativitätstechniken wenden diese heuristischen Prinzipien in formalisierter Form an.

Die Prinzipien »Assoziieren« und »Strukturen übertragen« fördern das intuitive Hervorbringen von Ideen, während die Prinzipien »Variieren«, »Kombinieren« und »Abstrahieren« eher in systematisch-analytischer Weise zu neuen Ansätzen hinführen.

Gruppe

Die meisten Kreativitätstechniken nutzen die Gruppe, da mehrere Personen mehr Wissen und Erfahrungen in den Ideengenerierungsprozeß einbringen als ein einzelner. Voraussetzung ist, daß eine offene Kommunikation in der Gruppe stattfindet, damit das unterschiedliche Wissen auch ausgetauscht wird. Dies ist nur in einer kleinen Gruppe mit maximal fünf bis sieben Teilnehmern möglich. Gegen größere Gruppen spricht außerdem, daß der Wissenszuwachs mit zunehmender Größe abnimmt.

Klassifizierung

Kreativitätstechniken lassen sich nach zwei Einflußfaktoren klassifizieren (Abb. 5.4-1).

Abb. 5.4-1: Klassifizierung der Kreativitätstechniken

Vorgehensprinzip zur Kreativitätsförderung	Ideenauslösendes Prinzip: Assoziation /Abwandlung	Ideenauslösendes Prinzip: Konfrontation
Verstärkung der Intuition	**Methoden der intuitiven Assoziation** *Brainstorming*-Methoden ■ Klassisches *Brainstorming* ■ Schwachstellen-*Brainstorming* ■ Parallel-*Brainstorming* *Brainwriting*-Methoden ■ Kartenumlauftechnik ■ Methode 635 ■ Ringtauschtechnik ■ *Brainwriting*-Pool ■ Galerie-Methode ■ Ideen-Delphi ■ Ideen-Notizbuch-Austausch	**Methoden der intuitiven Konfrontation** ■ Reizwortanalyse ■ Exkursionssynetik ■ Bildmappen-*Brainwriting* ■ Visuelle Konfrontation in der Gruppe ■ Semantische Intuition
Systematisches analytisches Vorgehen	**Methoden der systematischen Abwandlung** ■ Morphologisches Tableau ■ Sequentielle Morphologie ■ Modifizierende Morphologie *(Attribute Listing)* ■ Progressive Abstraktion	**Methoden der systematischen Konfrontation** ■ Morphologische Matrix ■ TILMAG ■ Systematische Reizobjekt-ermittlung

Quelle: /Geschka 85/

Legende: Die blau geschriebenen Techniken sind in Abb.5.4-2 näher beschrieben

Das Finden von Ideen kann durch Förderung der Intuition oder durch systematisch-analytisches Vorgehen unterstützt werden.

Die Ideen werden durch Abwandlung vorliegender Lösungsansätze – eine Idee entwickelt sich aus der anderen, insbesondere in Form von Assoziationsketten – oder aus der Konfrontation mit problemfremden Wahrnehmungen generiert.

Diese Klassifizierung führt zu vier Methodengruppen, in die sich die bekannten Kreativitätstechniken einordnen lassen.

In Abb. 5.4-2 werden zwei dieser Techniken näher dargestellt,

- das klassische *Brainstorming* und
- die Kartenumlauftechnik.

Mit beiden Techniken habe ich in der Praxis gute Erfahrungen gemacht.

Abb. 5.4-2: Ausgewählte Kreativitätstechniken

Klassisches *Brainstorming*

Funktionsweise: Spezielle Form einer verbalen Gruppensitzung.

Vier Regeln:
1. Freies und ungehemmtes Aussprechen von Gedanken; auch sinnlos erscheinende und phantastische Einfälle sind erwünscht, da sie andere Teilnehmer inspirieren können. Alle Vorschläge an Pinnwand oder *Flipchart* schreiben.
2. Die gemachten Vorschläge sind als Anregungen aufzunehmen und assoziativ weiterzuentwicklen. Voraussetzung: Zuhören und innerlich offen sein.
3. Kritik und Bewertung ist während der Sitzung verboten. Keine Killerphrasen wie »Das haben wir noch nie gemacht«, »Das hat noch keiner geschafft«.
4. Quantität geht vor Qualität. Vernunft und Logik sind nicht gefragt.

Voraussetzungen: Erfahrener Moderator, disziplinierte Teilnehmer, vier bis sieben Teilnehmer, nicht länger als 30 Minuten.

Problemklassen: Suchprobleme (viele Lösungsmöglichkeiten) und genau definierte Probleme.

Vorteile: Mimik, Gestik und Rhetorik führen zu spontanen Dialogen. Mehrere Teilnehmer können sich an den Diskussionen beteiligen. Der Funke kann von Teilnehmer zu Teilnehmer »überspringen«.

Nachteil: Rhetorisch begabte Teilnehmer können die Gruppe dominieren.

***Brainwriting:* Kartenumlauftechnik**

Funktionsweise: Spezielle Form einer nichtverbalen Gruppensitzung.

Vier Regeln:
1. Jeder Teilnehmer erhält Karten (z.B. Metaplankarten) und einen Filzstift. Jede Idee schreibt er auf eine Karte.
2. Jeder Teilnehmer legt seine beschriebenen Karten links von sich ab, griffbereit für seinen Nachbarn.
3. Gehen die eigenen Ideen aus, sichtet man den rechts von sich entstandenen Kartenstapel seines Nachbarn und läßt sich dadurch anregen.
4. Weiterentwickelte Ideen werden auf neue Karten geschrieben und alle Karten, einschließlich der durchgegangenen Karten des Nachbarn, werden links abgelegt.

Voraussetzungen: Vier bis sieben Teilnehmer, nicht länger als 30 Minuten.

Problemklassen: Suchprobleme und genau definierte Probleme, bei denen mehr Nachdenken erforderlich ist.

Vorteile: Nach Ende des *Brainwriting* kann schnell eine Strukturierung und Bewertung der Ideen vorgenommen werden. Karten werden thematisch gebündelt, mit Überschriften versehen und auf eine Pinnwand geheftet. Rhetorik spielt keine Rolle.

Nachteil: Spontaneität geht etwas verloren.

Quellen: /Geschka 85/, eigene Erfahrungen

intuitive Assoziation

Beide Techniken unterstützen die intuitive Assoziation. Das Ziel besteht darin, die Intuition in der Gruppe zu verstärken. Die Teilnehmer werden dazu angeregt, die Ideen der anderen aufzugreifen und assoziativ weiterzuentwickeln. Das kann sowohl verbal *(Brainstorming)* als auch schriftlich *(Brainwriting)* erfolgen.

Brainstorming

Brainstorming ist die bekannteste und am häufigsten angewandte Methode. Sie will die negativen Erscheinungen von Konferenzen wie destruktive Kritik, Rivalität der Teilnehmer, Verzettelung in Nebensächlichkeiten vermeiden.

Brainwriting

Brainwriting basiert ebenfalls auf dem Prinzip der wechselseitigen Assoziation. Die mündliche Kommunikation wird jedoch durch den Austausch schriftlicher Notizen ersetzt. Dadurch können rhetorisch begabte Teilnehmer eine Gruppe nicht dominieren. Außerdem gibt sie jedem Teilnehmer mehr Zeit zum Nachdenken und Ausgestalten.

Beide Methoden eignen sich gut, um Suchprobleme zu lösen. Bei Suchproblemen wird nach bestimmten Kriterien eine Auswahl getroffen.

intuitive Konfrontationen

Die Methoden der intuitiven Konfrontation versuchen den natürlichen kreativen Prozeß nachzuahmen. Originelle Ideen entstehen oft als Reaktion auf die Wahrnehmung völlig problemfremder Ereignisse, Gegenstände, Vorgänge und Gedanken ganz plötzlich – als Eingebung, als »Geistesblitz«.

Beispiel

Der Erfinder des Kugelschreibers ging in einem Park spazieren. Auf einem feuchten Rasen spielten Kinder Ball. Als der Ball über einen Kiesweg rollte, hinterließ er eine nasse Spur. Das brachte den Erfinder Biro auf das Wirkungsprinzip des Kugelschreibers.

Den Problemlösern legt man bei der intuitiven Konfrontation problemfremde Objekte vor. Die Auseinandersetzung mit diesen Objekten und ihren Bau- und Wirkungsprinzipien führt zu neuen Lösungsansätzen.

Werden durch die intuitive Assoziation keine neuen Impulse mehr erzielt, dann sollte man die intuitive Konfrontation einsetzen. Sie ist besonders für Gestaltungs- und Konstellationsprobleme geeignet. Bei diesen Problemen sind unter Beachtung von Randbedingungen komplexe Lösungsansätze zu finden.

Ist eine Kreativitätssitzung abgeschlossen, dann sind anschließend die Ideen zu sichten, zu ordnen, zu gruppieren, zu bewerten und zu gewichten.

Im Interesse jeden Software-Managers muß es liegen, die Problemlösungsfähigkeit seiner Mitarbeiter zu steigern. Über gezielte Kreativitätsförderung sollen neue Ideen und Lösungen angeregt werden. Es gilt die ablehnende Haltung aufgrund einer überwiegend nüchternen Berufsmentalität, insbesondere von Ingenieuren, zu überwinden.

Das gemeinsame Nachdenken über problematische Einzelfragen ist heute eher die Ausnahme. Daher sollten die Teams zum kooperativen Problemlösen unter Einsatz von Kreativitätstechniken angeregt werden.

Kreativität fördern

Führung zu Kreativität heißt /Schlicksupp 85, S. 96/:

- Freiräume für Experimente gewähren,
- Initiativen anerkennen,
- Ungewöhnliches positiv diskutieren,
- Erfolgsziele eindeutig definieren, aber die Wege zu den Erfolgen weitgehend offenlassen.

Als wichtigstes Arbeitsmittel für Gruppen- und Teambesprechungen hat sich die Metaplan-Technik durchgesetzt.

Metaplan-Technik

Bei der **Metaplan-Technik** werden Pinnwände, *Flip-Charts*, verschiedenartig geformte, farbige Karten (Rechtecke, Ovale, Kreise, Wolken), Stecknadeln, Klebepunkte und Filzstifte eingesetzt, um Ideen zu visualisieren, zu strukturieren, zu gewichten usw. (Abb. 5.4-3).

Eine **Pinnwand** ist eine Tafel, auf die mit Stecknadeln Karten geheftet werden können. Oft wird an die Tafel zunächst Packpapier geheftet. Darauf werden dann Karten gesteckt. Strukturierungen, Verbindungen usw. können dann durch Filzstifte auf dem Packpapier markiert und gezeichnet werden. Durch Wahl verschiedenartiger Karten können Ideen geeignet visualisiert werden.

Klebepunkte können dazu verwendet werden, um Gewichtungen durch die Teilnehmer vornehmen zu lassen. Beispielsweise erhält jeder fünf Punkte und kann diese beliebig auf zehn Ideen verteilen.

Ein ***Flip-Chart*** ist ein großer Papierblock, der auf einem Gestell befestigt ist. Seine Blätter können nach oben umgeschlagen werden. Auf *Flip-Chart*-Blätter kann man Problemlösungen skizzieren, die Blät-

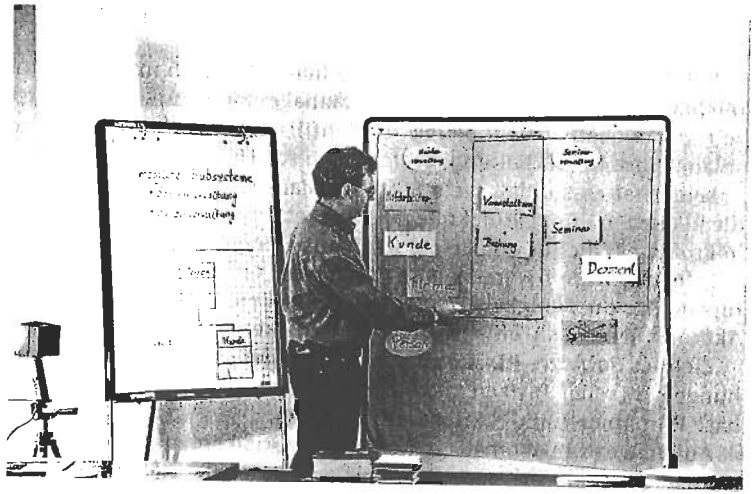


Abb. 5.4-3: Metaplan-Technik

ter anschließend vom Block abreißen und mit Kreppstreifen an die Wand heften. Verschiedene Lösungen kann man dann im Team gut diskutieren.

Die Metaplan-Technik sollte zur Grundausstattung jedes Besprechungsraums gehören. Inzwischen gibt es auch Software zur Sitzungsunterstützung, z.B. durch *Groupware*.

5.5 Risiken managen

Untersuchungen haben gezeigt, daß die Überarbeitung *(rework)* von fehlerhafter Software sehr kostenintensiv ist. Abb. 5.5-1 zeigt, daß ungefähr 80 Prozent aller Überarbeitungskosten benötigt werden, um 20 Prozent der Fehler zu beseitigen.

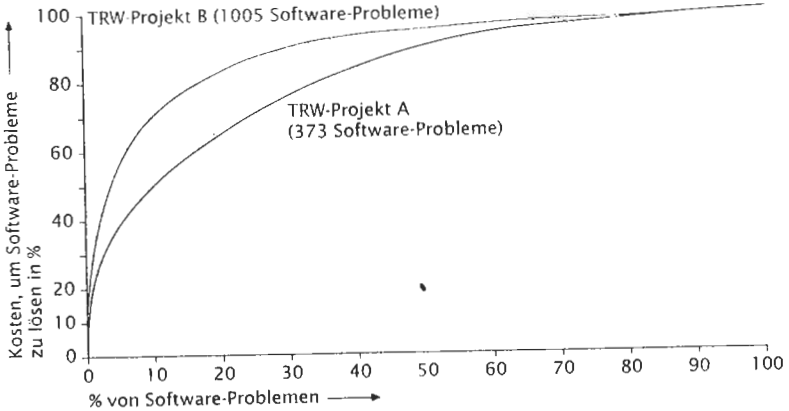


Abb. 5.5-1: Überarbeitungskosten /Boehm 89, S. 3/

Bei diesen 20 Prozent der Probleme handelt es sich um die risikoreichen Probleme. Ziel des Software-Managements muß es sein, diese risikoreichen 20%-Probleme zu identifizieren und zu beseitigen, solange ihre Überarbeitungskosten noch relativ gering sind.

Leider ist es so, daß viele Software-Manager nicht erkennen, wo die Risiken in ihrem Projekt liegen. Sie stecken ihre Energie daher oft in Probleme, die für den Erfolg nicht ausschlaggebend sind.

Erfolgreiche Software-Manager sind gute Risiko-Manager. Sie verfügen über die Fähigkeit, Risiken aufzudecken. Ihre Prioritäten und Aktionen lenken sie entsprechend den Risiken.

Risikomanagement

Ziel des Software-**Risikomanagements** ist es, die Wechselbeziehungen zwischen Risiken und Erfolg zu formalisieren und in anwendbare Prinzipien und Praktiken umzusetzen.

Aufgabe des Risikomanagements ist es, Risiken zu identifizieren, anzusprechen und zu beseitigen, bevor sie zu einer Gefahr für einen

erfolgreichen Software-I
arbeitungen darstellen.

Ein **Risiko** beschreibt die Möglichkeit, daß eine Aktivität einen körperlichen oder materiellen Verlust oder Schaden zur Folge hat. Von Risiko spricht man nur dann, wenn die Folgen ungewiß sind.

Risiko

Anders ausgedrückt: Ein Risiko ist ein potentielles Problem. Ein Problem ist ein Risiko, das eingetreten ist.

Vorgehensweise

Risikomanagement besteht aus sechs Schritten (Abb. 5.5-2). Jedem Schritt können mehrere Techniken zugeordnet werden, die helfen, die jeweiligen Aufgaben durchzuführen. Einige dieser Techniken werden im folgenden näher betrachtet. Als Beispiel zur Veranschauli-

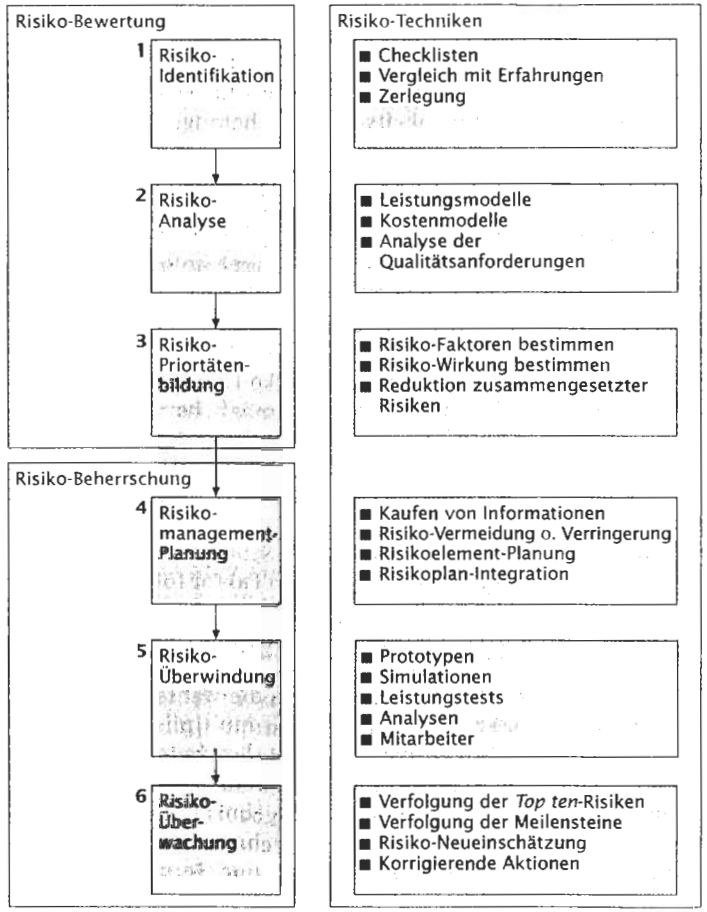


Abb. 5.5-2: Die sechs Schritt des Risiko-managements (/Boehm 91, S. 34/, modifizie

chung wird eine Software betrachtet, die ein Satelliten-Experiment steuern soll. Dieses Beispiel ist von /Boehm 91/ übernommen. Ein anderes Beispiel (Kernkraftwerks-Software) enthält /Boehm 89/. Anhand einer Software für ein Telekommunikationsprotokoll zeigt /Fairley 94/ das Risikomanagement.

1. Schritt: Risiko-Identifikation

Das Ergebnis einer Risiko-Identifikation ist eine Liste der projektspezifischen Risikoelemente, die den Projekterfolg gefährden. Projektspezifische Risiken sind Risiken, die nicht allgemein auf alle Projekte zutreffen, wie z.B. das Risiko, ein fehlerhaftes Produkt zu erstellen.

Checklisten

Bei der Risikoidentifikation kann man sich an Checklisten orientieren, um die projektspezifischen Risiken zu finden. Eine solche Checkliste zeigt Tab. 5.5-1. Sie enthält die zehn wichtigsten Quellen für Risiken in Software-Projekten. Jedem Risiko sind Techniken zugeordnet, mit denen man die Risiken vermeidet oder überwindet.

Um zu einer genaueren Einschätzung eines Risikos zu gelangen, können Risiko-Wahrscheinlichkeits-Tabellen herangezogen werden. Tab. 5.5-2 zeigt eine solche Tabelle. Sie hilft, die Wahrscheinlichkeit einzuschätzen, daß ein Projekt seinen Kostenrahmen überschreitet.

2. Schritt: Risiko-Analyse

Bei der Risiko-Analyse wird die Schadenswahrscheinlichkeit und das Schadensausmaß für jedes identifizierte Risikoelement geschätzt. Außerdem werden zusammengesetzte Risiken abgeschätzt.

Um zu einer quantitativen Bewertung eines Risikos zu gelangen, berechnet man einen Risiko-Faktor. Der Risiko-Faktor ergibt sich aus dem erwarteten Verlust, d.h. der Höhe der möglichen Verluste oder Schäden (Schadensausmaß), multipliziert mit den Wahrscheinlichkeiten ihres Eintretens (Schadenswahrscheinlichkeiten):

Risiko-Faktor

Risiko-Faktor = Schadenswahrscheinlichkeit x Schadensausmaß

Ein Risiko ist also um so größer, je höher die Eintrittswahrscheinlichkeit und das Ausmaß des potentiellen Schadens sind. Bezogen auf die Software-Technik läßt sich der Risiko-Faktor folgendermaßen definieren:

Risiko-Faktor = Wahrscheinlichkeit (unbefriedigendes Ergebnis) x Schadensausmaß

Der Risiko-Faktor wird also bestimmt durch die Wahrscheinlichkeit, ein unbefriedigendes Ergebnis zu erhalten multipliziert mit dem Schadensausmaß, das dieses unbefriedigende Ergebnis zur Folge hat.

Ein unbefriedigendes Ergebnis liegt vor, wenn die Hauptbeteiligten an einem Software-Projekt durch das Ergebnis einen Schaden erleiden. Ein unbefriedigendes Ergebnis ist mehrdimensional:

- Für Kunden und Entwickler sind Kosten- und Terminüberschreitungen unbefriedigend.

Risikoelement	Risikomanagement-Techniken
1 Personelle Defizite	■ Hochtalentierte Mitarbeiter einstellen ■ Teams zusammenstellen
2 Unrealistische Termin- und Kostenvorgaben	■ Detaillierte Kosten- und Zeitschätzung mit mehreren Methoden ■ Produkt an Kostenvorgaben orientieren ■ Inkrementelle Entwicklung ■ Wiederverwendung von Software ■ Anforderungen streichen
3 Entwicklung von falschen Funktionen und Eigenschaften	■ Benutzerbeteiligung ■ Prototypen ■ Frühzeitiges Benutzerhandbuch
4 Entwicklung der falschen Benutzungsschnittstelle	■ Prototypen ■ Aufgabenanalyse ■ Benutzerbeteiligung
5 Vergolden (über das Ziel hinausschießen)	■ Anforderungen streichen ■ Prototypen ■ Kosten/Nutzen-Analyse ■ Entwicklung an den Kosten orientieren
6 Kontinuierliche Anforderungsänderungen	■ Hohe Änderungsschwelle ■ Inkrementelle Entwicklung (Änderungen auf spätere Erweiterungen verschieben)
7 Defizite bei extern gelieferten Komponenten	■ Leistungstest ■ Inspektionen ■ Kompatibilitätsanalyse
8 Defizite bei extern erledigten Aufträgen	■ Prototypen ■ Frühzeitige Überprüfung ■ Verträge auf Erfolgsbasis
9 Defizite in der Echtzeitleistung	■ Simulation ■ Prototypen ■ Leistungstest ■ Instrumentierung ■ Modellierung ■ Tuning
10 Überfordern der Software–Technik	■ Technische Analyse ■ Kosten/Nutzen-Analyse ■ Prototypen

Quelle: /Boehm 91, S.35/. Diese Liste basiert auf einer Umfrage bei mehreren erfahrenen Projektleitern.

Tab. 5.5-1: Die Top ten-Risiken einer Software-Entwicklung

- Für Benutzer sind Produkte mit der falschen Funktionalität, mit Defiziten der Benutzungsoberfläche, der Leistung oder Zuverlässigkeit unbefriedigend.
- Für Wartungsingenieure ist schlechte Qualität unbefriedigend.

Beispiel

Tab. 5.5-3 zeigt die Risikofaktoren für die Software zur Steuerung eines Satellitenexperiments. Die linke Spalte listet die Risikoelemente auf, beschrieben als mögliche unbefriedigende Ergebnisse. Die zweite Spalte gibt an, wie hoch die Eintrittswahrscheinlichkeit für das jeweilige Ergebnis geschätzt wird (Skala 0 bis 10; 0 gleich »nicht vorhanden«, 10 gleich »hoch«). Spalte 3 führt das Schadensausmaß auf, das ein solches Ergebnis zur Folge hätte. Die rechte Spalte zeigt den Risiko-Faktor.

Kostenverursacher	Wahrscheinlichkeit		
	Unwahrscheinlich(0–0,3)	Wahrscheinlich(0,4–0,6)	Häufig(0,7–1)
Anforderungen			
Umfang	Gering, einfach oder leicht zerlegbar	Mittlere Komplexität, zerlegbar	Umfangreich, sehr komplex oder nicht zerlegbar
Hardwarebedingte Beschränkungen	Geringe oder keine	Einige	Signifikante
Anwendung	Keine Echtzeit, geringe Systemabhängigkeit	Eingebettet, einige Systemabhängigkeit	Echtzeit, eingebettet, starke Systemabhängigkeit
Technologie	Reif, vorhanden, eigene Erfahrungen	Vorhanden, einige eigene Erfahrungen	Neu oder neue Anwendung, wenig Erfahrung
Anforderungsänderungen	Keine oder sehr gering	Gering	Hoch
Personal			
Verfügbarkeit	Vorhanden, geringe Fluktuation zu erwarten	Verfügbar, etwas Fluktuation zu erwarten	Nicht verfügbar, hohe Fluktuation zu erwarten
Mischung der Software-Disziplinen	Gute Mischung	Einige Disziplinen unterrepräsentiert	Einige Disziplinen nicht vorhanden
Erfahrungen	Hoch	Mittel	Gering
Personalführung	Sehr gut	Gut	Schwach
Wiederverwendbare Software			
Verfügbarkeit	Kompatibel	Fraglich	Inkompatibel
Modifikationen	Gering oder keine Änderung	Einige Änderungen	Extensive Änderungen
Sprache	Kompatibel	Teilweise kompatibel	Inkompatibel
Lizenzrechte	Kompatibel mit Wartungs- und Wettbewerbsanforderungen	Teilweise kompatibel	Inkompatibel mit Wartungskonzept
Zertifizierung	Verifizierte Leistung, Anwendungskompatibel	Einige anwendungskomp. Testdaten verfügbar	Nicht zertifiziert, wenig Testdaten verfügbar
Werkzeuge und Umgebung			
Einrichtungen	Geringe oder keine Modifikation	Einige Modifikationen, vorhanden	Größere Modifikationen, nicht vorhanden
Verfugbarkeit	Vorhanden	Bedingt kompatibel	Nicht vorhanden
Lizenzrechte	Kompatibel	Partiell kompatibel	Inkompatibel
Konfigurationsmanagement	Volle Kontrolle	Etwas Kontrolle	Keine Kontrolle
	Auswirkungen		
	Ausreichende finanzielle Ressourcen	Defizite bei finanziellen Ressourcen, mögliche Überziehung	Signifikante finanz. Defizite, Kostenüberschreitungen wahrscheinlich

Quelle: /Boehm 91, S.38/. Diese Checkliste ist eine von mehreren, die in einem Handbuch der US-Luftwaffe über Risikoverminderung enthalten sind.

Tab. 5.5-2: Quantifizierung der Wahrscheinlichkeit von Kostenüberschreitungen

Unbefriedigendes Ergebnis	Wahrscheinlichkeit für unbef. Ergebnis	Schäden verursacht durch unbefr. Ergebnis	Risikofaktor
A Ein Software-Fehler tötet das Experiment	3-5	10	30-50
B Ein Software-Fehler verursacht den Verlust von Schlüsseldaten	3-5	8	24-40
C Fehlertolerante Eigenschaften führen zu einer nicht annehmbaren Leistung	4-8	7	28-56
D Überwachung der Software ergibt, daß unsichere Bedingungen als sicher gemeldet werden	5	9	45
E Überwachung der Software ergibt, daß sichere Bedingungen als unsicher gemeldet werden	5	3	15
F Verzögerungen bei der Hardwarelieferung verursachen Zeitüberschreitungen	6	4	24
G Software-Fehler bei der Datenreduktion verursachen zusätzl. Arbeit	8	1	8
H Schlechte Benutzungsoberfläche führt zu ineffizienter Bedienung	6	5	30
I Prozessorspeicher nicht ausreichend	1	7	7
J Datenbankmanagement-Software verliert hergeleitete Daten	2	2	4

Quelle: /Boehm 91, S.37/

Legende: 0 = nicht vorhanden, 10 = hoch, Risikofaktor = Spalte 2 * Spalte 3

Tab. 5.5-3: Risikofaktore bei der Softw für ein Satelli experiment

3. Schritt: Risiko-Prioritätenbildung

Um zu verhindern, daß man vor lauter identifizierten und analysierten Risikoelementen die wirklich relevanten Risiken nicht übersieht, müssen die Risiken nach Prioritäten geordnet werden.

Eine Möglichkeit dazu besteht in der Berechnung der Risiko-Faktoren. Oft konzentriert man sich auf die Eintretenswahrscheinlichkeit oder das Schadensausmaß. Wie Tab. 5.5-3 zeigt, gibt jedoch der Risiko-Faktor die höchsten Risiken an. Die Tabelle zeigt aber auch, daß es oft sehr schwierig ist, Eintrittswahrscheinlichkeiten genau genug

zu schätzen (z.B. A, B, C). Checklisten wie Tab. 5.5-2 können helfen, diese Wahrscheinlichkeiten zu schätzen. Eine vollständige Risiko-Analyse würde Prototypen, Leistungsmessungen und Simulationen erfordern, die aber teuer und zeitaufwendig sind.

4. Schritt: Risikomanagement-Planung

Nachdem die Hauptrisikoelemente und ihre relativen Prioritäten bestimmt sind, müssen Risikokontroll-Aktivitäten etabliert werden, um die Risikoelemente unter Kontrolle zu bringen. Der erste Schritt dazu besteht darin, Risikomanagement-Pläne zu entwickeln, die die notwendigen Aktivitäten festlegen.

Eine Hilfe dazu ist Tab. 5.5-1, die erfolgreiche Risikomanagement-Techniken für die wichtigsten Risikoelemente angibt. Der nächste Schritt besteht darin, für jedes Risikoelement einen Risikomanagement-Plan zu entwickeln.

Beispiel Ein hohes Risiko bei dem Satellitenexperiment besteht darin, daß fehlertolerante Eigenschaften zu einer nicht annehmbaren Leistung führen (Tab. 5.5-3, C).
In Tab. 5.5-1 wird bei Defiziten in der Echtzeitleistung als eine Technik die Erstellung von Prototypen vorgeschlagen.
Eine Risikomanagement-Planung kann nun darin bestehen, den Plan für die Erstellung eines Prototyps aufzustellen. Ziel des Prototyps soll es sein, die Abhängigkeit der Leistung von den fehlertoleranten Eigenschaften aufzuzeigen. Der Plan sollte folgende Fragen beantworten: Warum, Was, Wann, Wer, Wo, Wie und Wieviel.

Der letzte Planungsschritt besteht darin, die Risikomanagement-Pläne in den übergeordneten Projektplan zu integrieren.

5. Schritt: Risiko-Überwindung

Nach Abschluß der Risikomanagement-Planung werden die dort festgelegten Aktivitäten ausgeführt. Beispielsweise wird ein Prototyp erstellt, oder es werden Anforderungen gelockert.

6. Schritt: Risiko-Überwachung

Die Fortschritte bei der Risiko-Minimierung werden überwacht. Bei Abweichungen werden korrigierende Aktionen vorgenommen.

Eine bewährte Technik zur Risiko-Überwachung stellt die Verfolgung der *Top ten*-Risiken dar. Sie ermöglicht es dem Manager, sich effektiv auf die hohen Risiken und die kritischen Erfolgsfaktoren zu konzentrieren. Es wird vermieden, daß der Manager mit Details überschwemmt wird, die nur geringe Risiken beinhalten.
Die Verfolgung der *Top ten*-Risiken beinhaltet folgende Schritte:

a Die Risikoelemente in eine Rangfolge bringen.
b Festlegung regelmäßiger Überprüfungstermine durch das höhere Management.

c Jede Sitzung beginnt mit einem Bericht über den Fortschritt bei den *Top ten*-Risikoelementen. Die Übersicht sollte die Rangordnung jedes Risikoelements angeben, den Rang bei der letzten Sitzung und wie oft das Element schon auf der *Top ten*-Liste stand. Außerdem sollte angegeben werden, wie sich das Risikoelement seit der letzten Sitzung entwickelt hat.
d Die Sitzung soll sich darauf konzentrieren, die Risikoelemente zu beseitigen.

Beispiel Tab. 5.5-4 zeigt, wie sich die *Top ten*-Liste bei dem Satellitenexperiment im dritten Monat entwickelt hat. Die Tabelle zeigt, daß im dritten Monat ein Personalproblem am kritischsten ist. Eine Diskussion kann zu folgenden Alternativen führen:
- Die nicht verfügbare Schlüsselperson verfügbar machen.
- Projektpersonal umstellen.
- Neue Mitarbeiter innerhalb oder außerhalb der Organisation suchen.

Tab. 5.5-4: Projektbezog Top ten-Risik elementliste f das Satellite experiment

Risikoelement	Monatsrang			Fortschritt bei der Risiko-überwindung
	Dieser Monat	Letzter Monat	Anzahl Monate	
Ersetzen des Entwicklers für die Sensorkontroll-software	1	4	2	Gewünschter Ersatz-kandidat nicht verfügbar
Auslieferung der Ziel-hardware verzögert	2	5	2	Verzögerungen beim Beschaffungsverfahren
Datenformat für die Sensoren undefiniert	3	3	3	Aktionen des Software- u. Sensorteams nötig; fällig nächsten Monat
Personal für die Qualitätssicherung	4	2	3	Schlüsselperson verpflichtet, Fehlertoleranzprüfer benötigt
Fehlertoleranz gefährdet Leistung	5	1	3	Fehlertoleranzprototyp war erfolgreich
Datenbusänderungen berücksichtigen	6	—	1	Treffen der Datenbusentwerfer terminiert
Schnittstellendefinitionen für die Testumgebung	7	8	3	Einige Verzögerung bei den Aktionen; Treffen terminiert
Unsicherheiten in der Benutzungsoberfläche	8	6	3	Prototyp erfolgreich
Betriebskonzept erstellen	—	7	3	erledigt
Unsicherheiten in der wiederverwendeten Überwachungssoftware	—	9	3	geforderte Entwurfsänderungen erfolgreich durchgeführt

Quelle: /Boehm 91, S.39/

In Abhängigkeit von den gewählten Optionen sollten entsprechende Aktionen festgelegt werden.
In der Tabelle sieht man außerdem, daß einige Risikoelemente sich hin zu niedrigerer Priorität bewegen oder aus der Liste verschwinden, während andere höhere Prioritäten erhalten oder in die Liste aufgenommen werden.
Diejenigen, die in der Liste nach unten gehen, müssen noch überwacht werden, benötigen aber keine speziellen Managementaktionen mehr. Aufsteigende oder neue Risikoelemente benötigen höhere Managementaufmerksamkeit, um sie möglichst schnell zu lösen.

Abschnitt 3.3.7

Die volle Integration von Risikomanagement erfordert ein risikogetriebenes Prozeßmodell wie das Spiralmodell. Ein guter Weg, um Risikomanagement einzuführen, ist ein *Top ten*-Risiko-Verfolgungsprozeß.

Brainstorming →Kreativitätstechnik, um durch Sammeln und wechselseitiges Assoziieren von spontanen, verbal vorgetragenen Einfällen von Mitarbeitern in einer Gruppensitzung die beste Lösung eines Problems zu finden.
Brainwriting →Kreativitätstechnik, um durch Sammeln und Assoziieren von spontanen, schriftlich formulierten Einfällen von Mitarbeitern in einer Gruppensitzung die beste Lösung eines Problems zu finden.
Flip-Chart Auf einem Gestell befestigter, großer Papierblock, dessen Blätter nach oben umgeschlagen werden können (→Metaplan-Technik).
Führung Einwirkung auf Mitarbeiter, so daß vorgegebene Ziele erreicht werden.
Führungsstil Art und Weise, wie sich Manager gegenüber Mitarbeitern verhalten; hängt gegebenenfalls von einer besonderen Führungsphilosophie ab (→ *Management by ...*).
Kreativität Neue Ideen und Lösungen durch schöpferische Übertragung von Wissen und Erfahrungen aus anderen Bereichen finden, wobei traditionelle Denkmuster überwunden werden.
Kreativitätstechniken Formalisierte Anwendung der heuristischen Prinzipien Assoziation, Strukturtransformation, Abstraktion, Kombination, Variation zur Erzeugung von →Kreativität.
Management-by-... Managementmethoden bzw. Führungsphilosophien. Sie sind meistens durch Zielvorgaben für alle Stellen im Unternehmen, mehr oder weniger kooperativen Führungsstil und Delegation von Verantwortung gekennzeichnet. Das jeweils im Vordergrund stehende Konzept wird als Schlagwort hinter *Management-by-...* angegeben, wie *Management by Objectives, Management by Results* usw.
Metaplan-Technik Ideen werden durch Einsatz von →Pinnwänden, →Flip-Charts, Karten, Stecknadeln, Klebepunkten und Filzstiften visualisiert, strukturiert und gewichtet.
Pinnwand Tafel, auf die mit Stecknadeln Karten und Papier geheftet werden können (→Metaplan-Technik).
Risiko Möglichkeit eines körperlichen oder materiellen Schadens oder Verlustes durch eine Aktivität.
Risikomanagement →Führungsstil, bei dem sich der Manager auf die Risiko-Bewertung und die Risiko-Beherrschung konzentriert (→Risiko).
Team Zusammenarbeit verschiedener, qualifizierter Mitarbeiter in einer Gruppe, um eine gemeinsame Aufgabe zu erledigen.

Zu den wichtigsten, schwierigsten und zeitintensivsten Aufgaben eines Managers gehört die Führung von Mitarbeitern. Der Führungsstil eines Software-Managers muß berücksichtigen, daß er in der Regel hochqualifiziertes Personal anzuleiten und zu motivieren hat. Verschiedene *Management-by*-Methoden wurden entwickelt, um bestimmte Führungsphilosophien umzusetzen.

Weitverbreitet ist die Führung durch Zielsetzungsvereinbarung *(Management by Objectives).* Der Führungsstil *Management by Exception* wird in der Software-Technik durch das Risikomanagement unterstützt, mit dem Ziel, Risiken frühzeitig zu identifizieren und gezielt zu beseitigen oder zu reduzieren.

Zur Aufgabenbearbeitung haben sich in der Software-Technik kleine Teams (3 bis 4 Mitarbeiter) bewährt. Damit Teams erfolgreich sind, müssen sie richtig zusammengesetzt werden. Der Software-Manager muß die Teambildung fördern, die Mitarbeiter müssen teamfähig sein.

Schöpferische neue Ideen werden in einer Software-Entwicklung häufig benötigt. Kreativität kann durch Kreativitätstechniken geeignet unterstützt werden.

Brainstorming und *Brainwriting* sind bewährte Kreativitätstechniken, die sich auch in der Software-Technik bewährt haben.

Zur Visualisierung, Strukturierung und Bewertung von Ideen sind bei der Metaplan-Technik Pinnwand und *Flip-Chart* unentbehrliche Hilfsmittel.

Personalführung in der Software-Technik bedeutet aber nicht nur, den »Betrieb aufrecht zu erhalten«, sondern verlangt vom Software-Manager, seine Mitarbeiter auf die kommenden fachlichen Herausforderungen vorzubereiten und »einzuschwören«. Nur innovationsfreudige, kreative, teamorientierte und risikobewußte Manager können analoge Eigenschaften von ihren Mitarbeitern erwarten.

/Boehm 89/
Boehm B.W., *Software Risk Management*, Washington: IEEE Computer Society Press 1989, 496 Seiten
Sammelband, der Beiträge verschiedener Autoren zum Thema Risikomanagement enthält. Es werden alle wichtigen Aspekte des Risikomanagements behandelt.
/Boehm 91/
Boehm B.W., *Software Risk Management: Principles and Practices*, in: IEEE Software, Jan. 1991, pp. 32-41
Guter einführender Artikel über Risikomanagement
/DeMarco, Lister 91/
DeMarco T., Lister, T., *Wien wartet auf Dich! Der Faktor Mensch im DV-Management*, München: Carl Hanser Verlag 1991, 216 Seiten, Originalausgabe: *Peopleware*, New York: Dorset House Publishing Co. 1987, 188 Seiten
Die Autoren beschreiben die menschlichen Einflußfaktoren auf die Software-Entwicklung. Ein Muß für jeden Software-Manager!

Zitierte Literatur

/Berkel 84/
Berkel K., *Stichwort Gruppenarbeit*, in: Management Wissen 9/84, S. 29f
/Beyer 85/
Beyer, *Retten Sie sich vor dem Feuer!*, in: Management Wissen 11/85, S. 87
/Boehm 89/
Boehm B.W., *Software Risk Management*, in: ESEC (European Software Engineering Conference) 1989, S. 1–19
/Derschka/
Derschka P., *Wie Spitzenmanager führen*, in: Management Wissen, S. 10
/Fairley 94/
Fairley R., *Risk Management for Software Projects*, in: IEEE Software, May 1984, S. 57–67
/Geschka 85/
Geschka H., *Auch Kreativität kann man lernen*, in: Blick durch die Wirtschaft, 8.7.85
/Hauni 96/
Hauni, Hamburg, persönliche Mitteilung
/IKOSS 94/
IKOSS, Aachen, persönliche Mitteilung
/Kolb 95/
Kolb M., *Personalmanagement*, Berlin: Verlag A. Spitz 1995
/Mantei 81/
Mantei M., *The Effect of Programming, Team Structures on Programming Tasks*, in: Communications of the ACM, March 1981, S. 106–113
/Peters, Waterman 82/
Peters T.J., Waterman R.H., *In Search of Excellence. Lessons from America's Best-Run-Companies*, New York: Harper & Row 1982, 360 Seiten, deutsche Ausgabe: *Auf der Suche nach Spitzenleistungen*, Landsberg: Verlag Moderne Industrie, 1983
/Raidt 85/
Raidt F., *Die Konstruktion der Wirklichkeit*, in: Management Wissen 2/85, S. 72–82
/Rüßmann 83/
Rüßmann K.H., *Acht Regeln für Erfolg*, in: manager magazin 4/83, S. 144–155
/Schleupen 96/
Schleupen Computersysteme GmbH, Ettlingen 1996, persönliche Mitteilung
/Schlicksupp 85/
Schlicksupp H., *Jedem macht es Spaß zu denken*, in: Management Wissen 11/85, S. 92–96

Muß-Aufgabe
10 Minuten

1 *Lernziel: Die Charakteristika von Teams aufzählen, ihre Stärken und Schwächen nennen sowie angeben können, wodurch die Teambildung gefördert bzw. verhindert wird.*
Ein Team soll ein Projekt bearbeiten. Das Management beschließt folgende Maßnahmen für die Zusammenarbeit:

- **a** Das Projekt wird in kleine Teile mit definierten Ergebnissen gegliedert.
- **b** Jedes Teilergebnis wird genau kontrolliert und bewertet.
- **c** Der Ablauf des Projekts wird durch das Management bestimmt.
- **d** Alle Mitarbeiter des Projekts bekommen Büros auf einem Flur.
- **e** Tagesberichte über den Projektfortschritt werden eingeführt.
- **f** Die Vorgabe für das Team ist, das beste Produkt im Markt zu entwickeln, auch wenn dies etwas länger dauern sollte.
- **g** Ist das Team erfolgreich, werden die Mitarbeiter auf andere Projekte verteilt, um diese zu fördern.

- **h** Einige Topleute, die in anderen Projekten zur Zeit gute Arbeit liefern, werden in das Projektteam eingebaut. Sie sollen 30 Prozent ihrer Arbeitszeit in dem neuen Projekt mitarbeiten.

Welche Maßnahmen fördern und welche verhindern die Bildung eines guten Teams?

Muß-Aufgabe
5 Minuten

2 *Lernziel: Angeben können, über welche Eigenschaften ein Software-Manager verfügen soll und welchen Herausforderungen er sich stellen muß.*
Ein Software-Manager hat beschlossen, für sein Team die Objektorientierung einzuführen. Sein kurzfristiges Ziel ist, Software objektorientiert zu entwikkeln. Sein langfristiges Ziel ist, die Entwicklung durch CASE, Generatorsysteme und Wiederverwendung zu optimieren. Welche Eigenschaften eines Software-Managers sind an diesem Beispiel erkennbar?

Muß-Aufgabe
15 Minuten

3 *Lernziele: Wissen, was man unter Kreativität, Kreativitätstechniken, Metaplan-Technik, Pinnwand und Flip-Chart versteht. Die Kreativitätstechniken Brainstorming und Brainwriting erläutern können. Kreativitätstechniken klassifizieren können.*
Sie erhalten den Auftrag, ein *Brainstorming* für Ihr Team zu organisieren. Für einen Teil der Mitarbeiter ist dies das erste *Brainstorming*. Schildern Sie Ihre Aktivitäten. Erklären Sie alle Maßnahmen und klassifizieren Sie die Kreativitätstechnik *Brainstorming*.

Muß-Aufgabe
20 Minuten

4 *Lernziel: Führung, Führungsstil und Management-by-... erklären und die beschriebenen Management-by-Methoden erläutern können.*
Sie sind Manager eines Teams. Beschreiben Sie das Zusammenspiel mit Ihrem Team, wenn Sie den Führungsstil

- **a** *Management by Objectives (MbO)*
- **b** *Management by Results*
- **c** *Management by Delegation*
- **d** *Management by Participation*
- **e** *Management by Alternatives*
- **f** *Management by Exception*
- **g** *Management by Motivation*

verwenden.

Muß-Aufgabe
10 Minuten

5 *Lernziele: Für vorgegebene Szenarios geeignete Leitungsaktivitäten eines Software-Managers identifizieren, begründen und anwenden können. Leitungsaktivitäten eines Software-Managers aufzählen und erläutern können.*
Benennen Sie die jeweilige Leitungsaktivität des Software-Managers für folgende Aktivitäten:

- **a** Der Manager führt eine neue CASE-Umgebung ein, mit der langfristig die Produktivität gesteigert werden soll.
- **b** Das Zusammenspiel des Analyse- und des Entwurfs-Teams wird geplant.
- **c** Der Arbeitsfortschritt einzelner Mitarbeiter wird überprüft.
- **d** Ein Mitarbeiter des Teams wird als Projektleiter ausgewählt.
- **e** *Reviews* über den Projektverlauf werden angesetzt.
- **f** Der Manager wählt aus zwei möglichen Alternativen der Realisierung eine aus.

Muß-Aufgabe
10 Minuten

6 *Lernziel: Die Top ten-Risiken einer Software-Entwicklung und zugeordnete Risikomanagment-Techniken nennen und erläutern können.*
Welche der *Top ten*-Risiken einer Software-Entwicklung können Sie durch die Entwicklung eines Prototypen ausschließen? Begründen Sie Ihre Entscheidung.

Muß-Aufgabe
20 Minuten

7 *Lernziele: Die sechs Schritte des Risikomanagements angeben und erläutern sowie zugeordnete Techniken aufzählen können. Risikomanagement anhand von Beispielen durchführen können.*

Während der Entstehung eines Lehrbuches soll ein Student ein Fallbeispiel des Buches implementieren. Der Student erledigt diese Arbeit als Studienarbeit, also neben seinen normalen Vorlesungen. Zur Entwicklung der Software soll er eine neue Datenbank und eine neue Klassen-Bibliothek für die Oberfläche einsetzen. Das fertige Programm soll dem Buch auf einem Datenträger beigefügt werden. Führen Sie ein Risikomanagement anhand der vorgegebenen sechs Schritte durch.

Muß-Aufgabe
10 Minuten

8 *Lernziel: Eigenschaften teamorientierter Manager und teamfähiger Mitarbeiter aufführen können.*

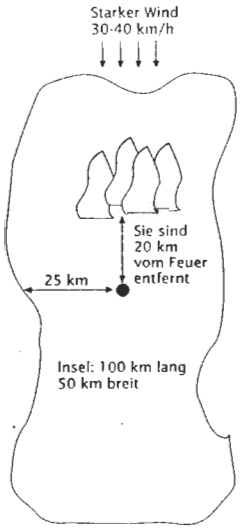
Welche der folgenden Maßnahmen zeichnen einen *teamorientierten* Manager aus, welche nicht?

a Der Manager bestimmt die Arbeitsabläufe eines Teams, weil diese bei anderen Teams, die er betreut hat, zum Erfolg führten.
b Das Team soll erweitert werden. Der Manager stellt dem Team drei Kandidaten vor. Das Team soll beurteilen, ob diese zu ihm passen.
c Der Manager vertraut auf den fachlichen Rat seiner Mitarbeiter.
d Um den Projektfortschritt zu überwachen, werden jede Woche *Reviews* angesetzt.

Kann-Aufgabe
30 Minuten

9 *Lernziel: Wissen, was man unter Kreativität, Kreativitätstechniken, Metaplan-Technik, Pinnwand und Flip-Chart versteht.*

Der Psychologe Beyer /Beyer 85/ veröffentlichte folgende anspruchsvolle Kreativitätsaufgabe mit dem Titel »Retten Sie sich vor dem Feuer!«, die im folgenden im Original wiedergegeben wird:



»Bitte schauen Sie sich einmal die Zeichnung an! Stellen Sie sich einmal vor, Sie befinden sich auf dieser Insel, die 100 Kilometer lang und 50 Kilometer breit ist. Hoch oben im Norden ist ein Feuer ausgebrochen, das mit einer Geschwindigkeit von etwa 40 Kilometern pro Stunde in Richtung Süden auf Sie zu treibt. Sie befinden sich etwa in der Mitte der Insel, die Entfernung zum Ufer beträgt etwa 25 Kilometer, das Feuer selbst ist etwa 20 Kilometer von Ihnen entfernt. Demnach wird Sie das Feuer in etwa 30 Minuten erreicht haben. Da das Gelände darüber hinaus auch noch zerklüftet und unwegsam ist, können Sie es niemals schaffen, innerhalb von 30 Minuten bis zum rettenden Wasser zu kommen. Außerdem macht Ihnen ja auch schon der Qualm zu schaffen, der bereits sichtbar auf Sie zutreibt. Der Boden besteht aus sehr trockenem Sand, bewachsen ist er mit sehr gut brennbarem, ausgedörrten Gras.

So wie das Ganze im Moment aussieht, ist es eine ausweglose Situation. Und nun meine Frage an Sie: Wie schaffen Sie es, daß Sie dem Feuer entkommen, daß Ihnen also kein ›Haar gekrümmt wird‹?

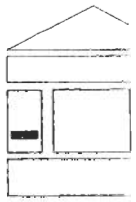
Um Ihnen die Sache nicht allzu schwer zu machen, haben Sie noch folgende Gegenstände bei sich: eine Schaufel, einen Kochtopf, einen Eimer, ein Feuerzeug, ein Radiogerät, ein Boot, ein Gewehr und ganz in Ihrer Nähe einen Brunnen, in dem es frisches Süßwasser gibt.

Bitte versuchen Sie nun, eine passable Lösung zu finden. Hier gibt es sicher mehrere Lösungen, aber eine Lösung gilt als optimal.

Im übrigen können Sie folgende Lösung bereits ausklammern: Sie können nicht dem Feuer entgegenlaufen, und darauf hoffen, am Feuer vorbeizukommen, dafür ist die Feuerwand zu breit.

Sie haben nun maximal 30 Minuten Zeit für eine optimale Lösung, dann hat Sie das Feuer erreicht. Ich wünsche Ihnen viel Erfolg.«

5 Leitung – Innovationen einführen



- Die bestimmenden Faktoren des Technologie-Transfers kennen und an Beispielen erläutern können.
- Die fünf Charakteristika einer Innovation aufzählen und ihre Auswirkungen auf den Diffusionsprozeß erklären können. — verstehen
- Die Personenkategorien, bezogen auf eine Innovationseinführung, nennen und charakterisieren sowie die S-Kurve erläutern können.
- Die Charakteristika des sozialen Systems aufzählen und ihre Auswirkungen auf den Innovationsprozeß beschreiben können.
- Den Kommunikationsprozeß charakterisieren und seinen Einfluß auf den Innovationsprozeß darstellen können.
- Darlegen können, welche Personengruppen auf welche Weise eine CASE-Einführung erleichtern können.
- Die Eigenschaften eines Methodenberaters skizzieren können.
- Relevante Gesichtspunkte einer Migrationsstrategie beschreiben können.
- Die Eigenschaften von Lernkurven erklären und auf Problemstellungen der Software-Technik anwenden können. — anwenden
- Anhand von vorgegebenen Szenarien prüfen können, inwieweit eine Innovation anhand der fünf Charakteristika leicht oder schwer einzuführen ist. — beurteilen
- Für vorgegebene Szenarien eine Innovationseinführung planen und begründen können.
- Beurteilen können, ob ein Software-Entwicklungsprojekt für die Einführung einer Innovation geeignet ist.

5.6 Einführung von Innovationen

Eines der größten Probleme der Software-Technik – vielleicht sogar das größte Problem – ist der Technologie-Transfer. Die Forschung auf dem Gebiet der Software-Technik hat in den letzten 25 Jahren eine ungeheuere Menge an Erkenntnissen gewonnen. In der täglichen Praxis der Software-Entwicklung werden jedoch oft nur wenige dieser Erkenntnisse angewandt. Lange Technologie-Transferzeiten von 15 bis 20 Jahren verzögern den technischen Fortschritt erheblich.

Lesehinweis: /Redwine, Riddle 85/

Technologie-Transfer-Probleme gibt es auch auf anderen Gebieten. Die Software-Technik zeichnet sich jedoch dadurch aus, daß sie eine hohe Innovationsgeschwindigkeit besitzt.

Eine permanente Aufgabe des Software-Managers besteht daher darin, Innovationen zu initiieren und einzuführen. Beispiele für notwendige Innovationseinführungen im Bereich der Software-Technik sind (wenn noch nicht erfolgt):

Hauptkapitel IV 2
uptkapitel I 2 bis 4
Kapitel II 3.3
Kapitel 6.2

- Einführung von CASE,
- Einführung der objektorientierten Software-Entwicklung,
- Einführung eines definierten Entwicklungsprozesses,
- Einführung von Metriken.

Ironischerweise ist die »Mechanisierung der Mechanisierer« /Bouldin 89, S. 1/, d.h. in diesem Fall der Software-Entwickler, oft schwieriger als die Einführung von Systemen (die die »Mechanisierer« entwickelt haben) bei Endbenutzern, von denen selbstverständlich die Bereitschaft zu Änderungen erwartet und vorausgesetzt wird.

Im folgenden werden die wesentlichen Faktoren beschrieben, die den Technologie-Transfer beeinflussen.

Vorgehensweise

Die oben aufgeführten Beispiele für Innovationseinführungen in der Software-Technik sind in den übergeordneten Rahmen »Technologie-Transfer« einzuordnen. Es werden im folgenden daher jeweils die Erkenntnisse der Technologie-Transfer-Forschung sowie anderer Forschungsgebiete dargestellt. Ihre Relevanz für die Software-Technik wird am Beispiel einer CASE-Einführung aufgezeigt.

Lesehinweis: E.M. Rogers

Die grundlegenden Arbeiten zum Technologie-Transfer stammen von E.M. Rogers /Rogers 83/. Seine Erkenntnisse gewann er ursprünglich aus Untersuchungen über die Verbreitung von Innovationen im landwirtschaftlichen Bereich. Die Ergebnisse wurden dann durch empirische Studien in anderen Bereichen evaluiert. Es wurden Faktoren isoliert, die die Ausbreitung *(diffusion)* einer Innovation in einer Zielgruppe bestimmen.

Diffusion

Unter **Diffusion** versteht man in diesem Zusammenhang den Prozeß des Transfers von Technologie von denen, die sie entwickelt haben, zu denen, die sie einsetzen.

Innovation

Eine **Innovation** ist eine Idee, ein Verfahren oder ein Objekt, das für die Personengruppe neu ist, die das Ziel der Einführung ist.

Im folgenden wird zunächst ein möglicher Lebenszyklus für Innovationseinführungen behandelt. Unabhängig von einem Lebenszyklus werden dann bestimmende Faktoren des Technologie-Transfers behandelt. Diese Faktoren sind in Abb. 5.6-1 im Überblick dargestellt.

Vorgehensweise

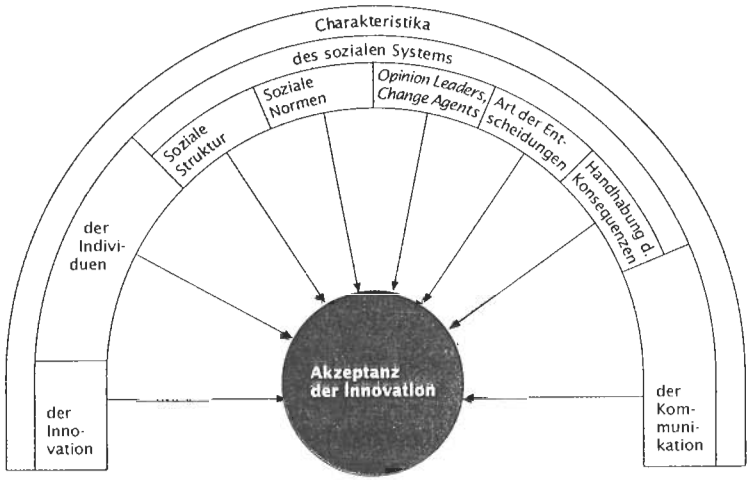


Abb. 5.6-1: Bestimmende Faktoren des Technologie-Transfers

5.6.1 Der Lebenszyklus von Innovationseinführungen

Ähnlich wie es einen Lebenszyklus für die Entwicklung von Software gibt, so gibt es auch einen Lebenszyklus für die Einführung von Innovationen bzw. für die Einführung von Veränderungen (Abb. 5.6-2). Ähnliche Lebenszyklen beschreiben auch /Raghavan, Chand 89, S. 84/ und /Spinas, Troy, Ulich 83, S. 88 f./.

Aus psychologischer Sicht kann die Dynamik von Veränderungsprozessen durch ein Drei-Schritte-Modell beschrieben werden /Spinas, Troy, Ulich 83, S. 89 f./ (Abb. 5.6-3).

Soziale Zustände können als ein Gleichgewicht zwischen treibenden und hemmenden Kräften um das Gleichgewichtsniveau herum angesehen werden. Veränderungen erfordern die Auflösung des bestehenden Gleichgewichts und die Überführung in ein neues Gleichgewicht.

»Auftauen« bedeutet, daß das Gleichgewicht zugunsten der treibenden Kräfte ins Wanken gebracht wird. Dies kann durch Verstärkung der treibenden Kräfte oder durch Schwächung der hemmenden Kräfte geschehen. Das bestehende Gleichgewicht kann jedoch von den Mitgliedern eines sozialen Systems beharrlich verteidigt werden, was sich dann als Widerstand bemerkbar macht.

In dem Veränderungsschritt werden die Kräfte auf das geplante Ziel hin gesteuert. Die Erwartungen der Betroffenen spielen dabei

Abb. 5.6-2: Lebenszyklus von Innovationseinführungen

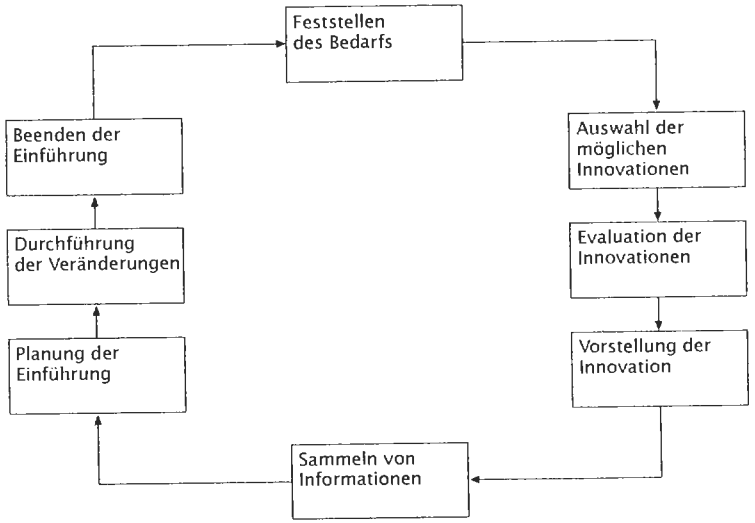
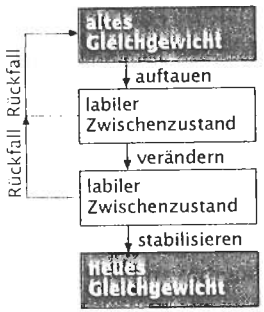


Abb. 5.6-3: Drei-Schritte-Modell der Dynamik von Veränderungsprozessen



eine große Rolle. Wichtig ist, daß sich die aufgebaute Erwartungshaltung im Rahmen des technisch und organisatorisch Möglichen bewegt. Sind die Erwartungshaltungen zu euphorisch, dann können die Veränderungsprozesse eine nicht mehr steuerbare Eigendynamik entwickeln.

Bewährt hat sich eine Unterteilung des Veränderungsprozesses in überschaubare Teilschritte, die Erfolgserlebnisse vermitteln und die Vorteile der Innovation sichtbar machen. Außerdem können überzogene und unrealistische Erwartungen laufend korrigiert werden.

Diese Art des Veränderungsprozesses erlaubt den Mitarbeitern einen stetigen Gewöhnungs- und Lernprozeß. Dies ist einer schlagartigen Umstellung vorzuziehen.

Der Stabilisierungsschritt sichert die Veränderung durch ihre Institutionalisierung und Integration in die Organisation. Solange der neue Zustand als Ausnahmesituation betrachtet wird, besteht die Gefahr des Rückfalls in die Ausgangssituation.

5.6.2 Charakteristika einer Innovation

Fünf Charakteristika einer Innovation beeinflussen direkt die Verbreitungsrate der Innovation /Rogers 83/:

1 **Relativer Vorteil** der Innovation gegenüber vorhandenen Alternativen — relativer Vorteil

Je größer der relative Vorteil einer Innovation durch die Zielgruppe wahrgenommen wird, desto schneller vollzieht sich die Verbreitung der Innovation. Der relative Vorteil kann in ökonomischen Kriterien gemessen werden, aber soziale Faktoren wie Prestige, Bequemlichkeit und Zufriedenheit sind oft ebenfalls wichtige Komponenten.

2 **Kompatibilität** mit gegenwärtigen Verfahren — Kompatibilität

Die Kompatibilität gibt an, wie eine Innovation wahrgenommen wird im Vergleich zu vorhandenen Werten, Erfahrungen und den Bedürfnissen der Zielgruppe. Erfordert ein neues Produkt eine signifikante Änderung des Verhaltens, der Einstellung oder des Glaubens, dann verliert es an Kompatibilität.

Die Einführung einer inkompatiblen Innovation kann die Einführung eines neuen Wertesystems erfordern.

3 **Einfachheit** der Innovation — Einfachheit

Die Einfachheit gibt an, wie leicht die Innovation für die Zielgruppe zu erlernen und zu benutzen ist. Neue Ideen, die leicht zu verstehen sind, werden schneller angenommen, als Innovationen, die neues Wissen und neue Fertigkeiten erfordern.

4 Möglichkeit zum **Ausprobieren** — Ausprobieren

Das Ausprobieren einer Innovation ermöglicht der Zielgruppe anhand von kleinen Problemen zu überprüfen, wie gut die Innovation zu erlernen, zu verstehen und anzuwenden ist. Die Unsicherheit über die Nützlichkeit nimmt ab und die Wahrscheinlichkeit einer schnellen Übernahme nimmt zu. Wesentlich ist außerdem, daß die Innovation einen inkrementellen Einsatz beginnend bei kleinen Problemen und hinführend zu großen Problemen erlaubt.

5 **Sichtbarkeit** der Ergebnisse — Sichtbarkeit

Die Sichtbarkeit gibt an, wie sichtbar die Ergebnisse der Innovation gegenüber anderen Personengruppen sind. Je leichter es ist, die Ergebnisse der Innovation zu erkennen, desto schneller vollzieht sich die Übernahme der Innovation.

Prüft man anhand dieser Charakteristika CASE-Umgebungen, dann läßt sich folgendes feststellen: — Anwendung auf CASE

zu 1 Relativer Vorteil

Wie stark der relative Vorteil in der Zielgruppe wahrgenommen wird, hängt wesentlich von der bisherigen Arbeitsweise ab. Zur Beurteilung des relativen Vorteils ist insbesondere die Situation der einzelnen Mitarbeiter zu betrachten. Deren Beurteilung kann völlig anders aussehen als eine Beurteilung aus Unternehmenssicht.

Beispiel Hat ein Systemanalytiker bisher nur verbale Pflichtenhefte erstellt, dann stellt sich für ihn die Situation folgendermaßen dar:
Die verbalen Pflichtenhefte hat er handschriftlich erstellt. Eine Sekretärin hat sie dann mit einem Textsystem erfaßt. Wurden im Entwurf oder in der Implementierung Unklarheiten im Pflichtenheft entdeckt, dann konnte der Systemanalytiker auf eine fehlerhafte Interpretation seines Textes verweisen, d.h. er war für seine fachliche Arbeit nur schwer verantwortlich zu machen.
Durch die Einführung der objektorientierten Software-Entwicklung verbunden mit einem entsprechenden CASE-System verschlechtert sich für den Systemanalytiker die Situation:
Zusätzlich zu verbalen Pflichtenheften muß er nun auch noch OOA-Modelle erstellen und beides mit dem CASE-System erfassen und verwalten. Neben dem Umgang mit CASE-Systemen muß er noch eine neue Methode erlernen. Außerdem sind Fehler jetzt leichter nachzuweisen, da OOA-Modelle formale Spezifikationen sind, deren Eindeutigkeit und Konsistenz besser zu überprüfen sind als verbale Beschreibungen.
Aus der subjektiven Sicht eines »konservativen« Systemanalytikers bringt die CASE-Einführung nur Nachteile mit sich und auf jeden Fall zusätzliche Arbeit.

Beispiel Ein Systemanalytiker, der bisher bereits verbale Pflichtenhefte und OOA-Modelle – allerdings ohne CASE-Werkzeuge – erstellt hat, sieht eine CASE-Einführung völlig anders. Bisher hat er seine OOA-Modelle mühselig mit einem Zeichenwerkzeug erstellt. Ein OO-CASE-Werkzeug erleichtert ihm wesentlich die Erstellungs-, Verwaltungs- und Überprüfungsarbeit seiner OOA-Modelle.

Beispiel Aus Unternehmenssicht verbessert eine CASE-Umgebung vor allem die Qualität und die Wartungsproduktivität durch die einheitliche, rechnergestützte Dokumentation. Das Software-Management muß die Mitarbeiter davon überzeugen, daß diese Ziele auch in ihrem eigenen Interesse liegen, weil dadurch die Wettbewerbsfähigkeit und die Arbeitsplätze gesichert werden.

zu 2 Kompatibilität
In der Regel ist die CASE-Einführung mit der Anwendung neuer Methoden verknüpft. Sind diese Methoden für die Zielgruppe neu und wurden bisher keine oder wenige Methoden eingesetzt, dann ist eine Verhaltensänderung eines jeden Einzelnen bezogen auf die Entwicklung von Software erforderlich. Die Kompatibilität von CASE mit der bisherigen Arbeitsweise ist in den meisten Fällen dann nicht gegeben.

zu 3 Einfachheit
Wird nicht nur ein einzelnes CASE-Werkzeug, sondern eine umfangreiche CASE-Umgebung eingeführt, dann handelt es sich um keine

einfache, sondern eine komplexe Innovation, die neues Wissen und neue Fertigkeiten erfordert.

zu 4 Ausprobieren
Inwieweit sich eine CASE-Umgebung mit kleinen Anwendungen ausprobieren läßt, hängt wesentlich vom CASE-Hersteller ab. Durch geeignete Voreinstellungen des Systems, durch mitgelieferte Fallbeispiele und Szenarien kann er einen inkrementellen Einsatz wesentlich fördern.

Voraussetzung auf der anderen Seite ist natürlich, daß die Zielgruppe die durch die CASE-Umgebung unterstützten Methoden beherrscht.

zu 5 Sichtbarkeit
Da viele CASE-Werkzeuge insbesondere grafische Methoden unterstützen, sind die Ergebnisse anschaulich sichtbar. Führt das CASE-Werkzeug außerdem umfangreiche Qualitätsüberprüfungen durch und meldet Fehler in verständlicher Form, dann sind diese Vorteile gut erkennbar. Ein Problem ist jedoch die Sichtbarkeit der Wirtschaftlichkeit bezüglich des Verhältnisses von Kosten und Nutzen. Hier ist ein Nachweis der Wirtschaftlichkeit schwierig zu führen.

Kapitel IV 2.1

Die folgende Tabelle zeigt die gewonnenen Erkenntnisse nochmals im Zusammenhang:

Charakteristika	Anwendung auf CASE	Kommentar
1 Relativer Vorteil	von – bis +	Hängt stark
2 Kompatibilität	–	von der
3 Einfachheit	–	Zielgruppe ab.
4 Ausprobieren	von – bis +	Hängt vom Produkt ab.
5 Sichtbarkeit	+	Hängt vom Produkt und den Methoden ab.

Im Normalfall sind die ersten drei Charakteristika als negativ zu bewerten, im Optimalfall sind das erste und die beiden letzten Charakteristika als positiv anzusehen. Aufgrund dieser CASE-Charakteristika läßt sich für den Normalfall voraussagen, daß eine CASE-Einführung nicht die idealen Voraussetzungen mit sich bringt, um schnell eingeführt zu werden und sich in der Zielgruppe zu verbreiten. Daher muß eine CASE-Einführung sehr sorgfältig geplant und durchgeführt werden.

Vor einem Fehlschluß muß jedoch gewarnt werden. Um die Probleme einer CASE-Einführung zu umgehen, könnte man zu der Erkenntnis gelangen, nur einfache Werkzeuge auszuwählen, die kompatibel mit den bisherigen Vorgehensweisen der Entwickler sind. Eine solche Sichtweise verhindert jedoch Innovationen, »zementiert« bisherige Vorgehensweisen und verhindert eine Produktivitäts- und Qualitätsverbesserung.

Empfehlunge

Anwendung auf OO

Prüft man anhand der oben aufgeführten Charakteristika die Einführung der objektorientierten Software-Entwicklung, dann gelangt man zu folgenden Ergebnissen:

zu 1 Relativer Vorteil

Eine objektorientierte Software-Entwicklung besitzt gegenüber einer strukturierten Software-Entwicklung im wesentlichen folgende eindeutige relative Vorteile: Einheitliche Kernkonzepte in allen Entwicklungsphasen, kein Strukturbruch zwischen Definition und Entwurf, leichte Erweiterbarkeit und Änderbarkeit, Unterstützung der Wiederverwendbarkeit, leichtere Modellierung der realen Welt.

zu 2 Kompatibilität

Eine objektorientierte Entwicklung ist *nicht* kompatibel mit einer strukturierten Entwicklung. Wurde in der Definitionsphase aber bereits das *Entity Relationship*-Modell verwendet, dann erleichtert dies den Übergang zur objektorientierten Analyse, da die objektorientierte Analyse als ein Teilkonzept das *Entity Relationship*-Modell enthält.

zu 3 Einfachheit

Abschnitt IV 1.1.1

Im Vergleich zur strukturierten Software-Entwicklung erfordert die Objektorientierung ein höheres Abstraktionsvermögen. Vom Umfang der Konzepte her sind beide Methoden ungefähr vergleichbar.

zu 4 Ausprobieren

Viele Vorteile der Objektorientierung können bereits an kleinen Beispielen gezeigt werden.

zu 5 Sichtbarkeit

Da OOA- und OOD-Modelle grafisch dargestellt werden, können objektorientierte Software-Entwicklungen gut sichtbar gemacht werden.

Zusammengefaßt ergibt sich folgende Bewertung:

Charakteristika	Anwendung auf OO	Kommentar
1 Relativer Vorteil	+	Hängt stark
2 Kompatibilität	–	von den bisherigen
3 Einfachheit	von – bis +	Methoden ab.
4 Ausprobieren	+	
5 Sichtbarkeit	+	

Im Vergleich zu einer CASE-Einführung ist die Einführung der Objektorientierung leichter, da die Charakteristika der Objektorientierung insgesamt für eine schnelle Verbreitung sprechen.

5.6.3 Charakteristika der Zielgruppe

Untersuchungen /Rogers 83/ haben gezeigt, daß die Mitglieder einer Zielgruppe in unterschiedlichem Maße bereit sind, eine Innovation anzunehmen. Es lassen sich fünf verschiedene Personenkategorien unterscheiden:

- Innovatoren,
- Frühe Anwender,
- Frühe Majorität,
- Späte Majorität,
- Nachzügler.

Den prozentualen Anteil jeder Personenkategorie in der Zielgruppe zeigt Abb. 5.6-4 unten.

Diese Kategorien spiegeln auch die zeitliche Reihenfolge wider, in der Innovationen benutzt werden. Daher dehnt sich die Verbreitung einer Innovation auch über die Zeit.

Die Verbreitung einer Innovation in einer Zielgruppe folgt einer **S-Kurve**, wie sie Abb. 5.6-4 oben zeigt.

Literaturh
Die S-Kurv
ausführlic
behandelt
/Asthana

Die Anzahl der Innovatoren und frühen Anwender ist klein. Daher ist die Anzahl der Mitarbeiter, die eine Innovation übernehmen, am Anfang klein. Jeder Verbreitungsprozeß einer Innovation beginnt daher langsam, beschleunigt sich und verlangsamt sich anschließend wieder, da dann die Anzahl der Nichtanwender immer kleiner wird.

Am Anfang einer Innovationseinführung ist es daher empfehlenswert, sich auf die Teilmenge der Innovatoren und frühen Anwender zu konzentrieren. Ihre Erfahrungen mit der Innovation können dann benutzt werden, um die Verbreitungsgeschwindigkeit bei der restlichen Zielgruppe zu beschleunigen.

Empfehlun

Es spricht daher einiges dafür, die Einführung zunächst in einer »Pilotgruppe« oder »Pilotabteilung« zu beginnen und erst später auf die gesamte Organisation auszudehnen. Die »Pilotgruppe« sollte die Charakteristika von Innovatoren und frühen Anwendern aufweisen.

»Schneeba
system«

5.6.4 Charakteristika des sozialen Systems

Das soziale System ist die Umgebung, in der der Einführungsprozeß stattfindet. Fünf verschiedene Aspekte beeinflussen den Einführungsprozeß:

- Soziale Struktur,
- Soziale Normen,
- Rollen der Meinungsbildner und Innovationsförderer,
- Art der Innovationsentscheidungen,
- Konsequenzen der Innovationseinführung.

Die **soziale Struktur** gibt an, wie die Mitglieder eines sozialen Systems miteinander kommunizieren. *Formale Strukturen* geben ei-

Soziale Str

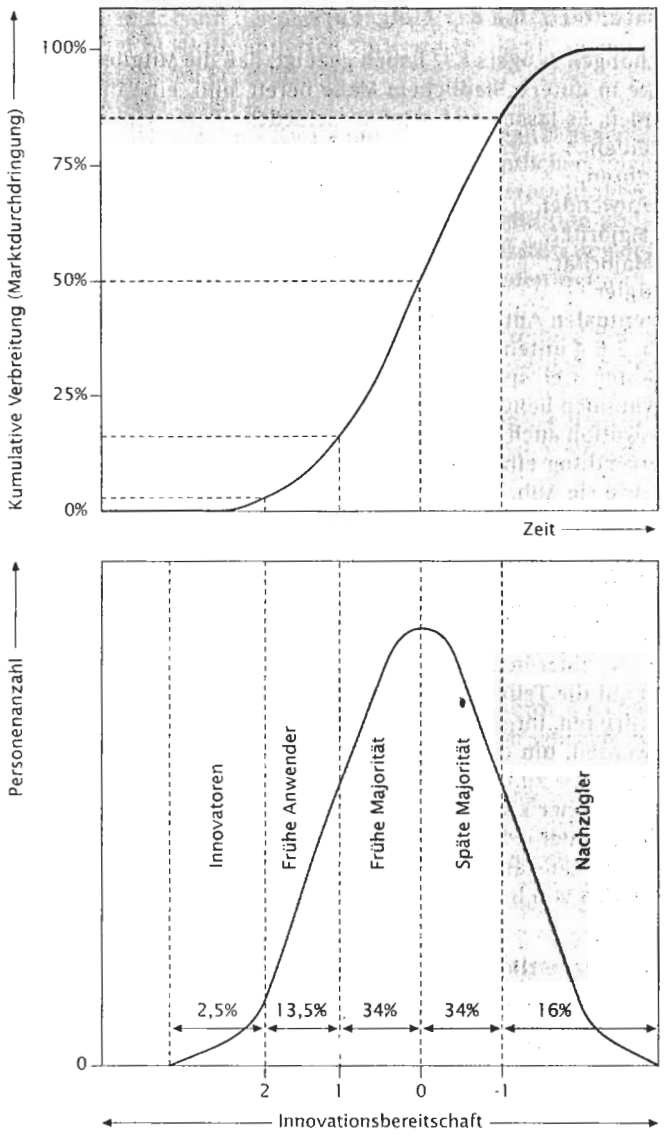


Abb. 5.6-4: Prozentualer Anteil von Personenkategorien bezogen auf ihre Innovationsbereitschaft /Rogers 83/, /Asthana 95/ (unten), Verbreitungsgeschwindigkeit einer Innovation in der Zielgruppe (S-Kurve) /Asthana 95, S 53/ (oben)

nem System Ordnung und Stabilität und reduzieren die Unsicherheiten des menschlichen Verhaltens in einem solchen System. *Informale Strukturen* wie Freundschaften, Freizeit-Aktivitäten usw. entstehen wegen der sozialen Bedürfnisse der Mitglieder. Die soziale Struktur kann die Verbreitung einer Innovation fördern oder behindern. Die Bereitschaft einer Person, Innovationen zu übernehmen, hängt neben individuellen Eigenschaften auch von der Natur des sozialen Systems ab. In einem konservativen Umfeld wird die Begeisterung früher Anwender gedämpft, während in einem liberalen Umfeld zögernde Anwender aufnahmebereiter für Innovationen sind. Eine Firma, deren wirtschaftlicher Erfolg von der ständigen Innovation im Software-Bereich abhängt, muß ein liberales Umfeld schaffen, um die Innovationsbereitschaft zu fördern.

Empfehlun
les Umfeld

Soziale Normen sind etablierte Verhaltensmuster für die Mitglieder eines sozialen Systems. Sie definieren einen Bereich tolerierten Verhaltens und dienen als eine Richtlinie für die Mitglieder. Starre soziale Normen können eine ernsthafte Barriere für die Verbreitung von Innovationen sein. Sind sie vorhanden, dann sollten sie abgebaut werden.

Soziale No

Empfehlun
Normen abl

Meinungsbildner *(opinion leaders)* und **Innovationsförderer** *(change agents)* sind Schlüsselpersonen in jedem sozialen System. Meinungsbildner sind normalerweise technische Leiter mit umfangreicher Erfahrung, die eine hohe Glaubwürdigkeit besitzen. Sie haben daher einen großen Einfluß auf das Verhalten und die Einstellung von anderen Personen. Innovationsförderer sind formal autorisiert und bevollmächtigt, Änderungen in einem sozialen System vorzunehmen. Anders als Meinungsbildner müssen sie keine technischen Leiter sein.

opinion le
change ag

Meinungsbildner und Innovationsförderer spielen eine kritische Rolle bei der Einführung von Innovationen. Ihre Einstellung zu einer Innovation bestimmt zum großen Teil das Ergebnis des Einführungsprozesses. Unterstützen sie eine Innovation, dann kann dies ein effektiver Katalysator für eine schnelle Einführung und Anwendung sein. Kritisieren sie die Innovation, dann kann dies die Verbreitung behindern.

Innovationsförderer müssen folgende Gesichtspunkte beachten:

- Nicht nur die technischen Aspekte einer Innovation betrachten, sondern insbesondere die sozialen Aspekte beachten.
- Sich nicht einseitig an eigenen Interessen und Vorstellungen orientieren.
- Das Fachwissen der Betroffenen, deren Kenntnisse und Fähigkeiten aus der täglichen praktischen Arbeit nicht unterschätzen.
- Den eigenen Informationsvorsprung durch das eigene Verhalten nicht demonstrativ unterstreichen.

Empfehlung: Methodenberater

Abschnitt 5.6.7

Um Innovationen zu fördern, ist die Stelle eines Innovationsförderers einzurichten und kompetent zu besetzen. In der Software-Technik wird ein solcher Mitarbeiter oft **Methodenberater** genannt. Der Methodenberater ist vom Management zu unterstützen und mit entsprechenden Kompetenzen auszustatten.

Art der Innovationsentscheidung

Die Akzeptanz einer Innovation hängt wesentlich von der Art und Weise ab, wie die **Innovationsentscheidung** getroffen wird.

Die Entscheidung kann durch ein *Individuum*, z.B. den Innovationsförderer, getroffen werden, unabhängig von den anderen Mitgliedern des sozialen Systems. Trifft eine *Gruppe* gemeinsam und freiwillig die Entscheidung, eine Innovation einzuführen, dann beschleunigt dies wesentlich die Einführung und Anwendung.

Entscheidungen durch *Autoritäten* erfolgen normalerweise durch einige Schlüsselpersonen, die die Macht, den Status oder das technische Wissen besitzen. Ist die Entscheidung gefallen, dann ist sie bindend für die Mitglieder des sozialen Systems.

Auf welche Art und Weise eine Innovationsentscheidung gefällt wird, hat viele Implikationen für den Verbreitungsprozeß.

Eine Gruppenentscheidung kann länger dauern als eine individuelle Entscheidung. Ist die Entscheidung aber gefallen, dann erfolgt die Verbreitung schnell, da die Entscheidung auf einem Konsens beruht. Entscheidungen durch Autoritäten können die Verbreitung beschleunigen oder verlangsamen in Abhängigkeit davon, wie die Einstellung der Mitglieder zu den Autoritäten ist.

Empfehlungen

Innovationsentscheidungen sollten idealerweise durch Konsensbildung in den jeweiligen Gruppen getroffen werden. Die Initiative kann vom Methodenberater oder vom Management kommen. Meinungsbildner sollten identifiziert und in Innovationsentscheidungen eingebunden werden.

Mitwirkung der Betroffenen

Ist das Idealmodell nicht zu verwirklichen, weil die Zielgruppe z.B. zu groß ist, dann sollte durch eine geeignete Organisationsform dafür gesorgt werden, daß die Betroffenen am Auswahl- und Einführungsprozeß mitwirken können.

frühzeitige und fortlaufende Information

Wichtig ist eine frühzeitige und fortlaufende Information der Zielgruppe über die Gründe der Innovation sowie über die ökonomischen, technischen und sozialen Zusammenhänge und Folgen. Dadurch können Ungewißheiten und Unsicherheiten reduziert sowie Spekulationen und Gerüchtebildungen weitgehend vermieden werden.

Weiterbildung

Befürchtungen der Betroffenen, den zukünftigen Anforderungen mit den bisherigen Kenntnissen und Fertigkeiten nicht mehr gewachsen zu sein, kann durch frühzeitige Weiterbildungsmaßnahmen begegnet werden.

Methodentraining

Im Rahmen einer CASE-Einführung ist es dabei besonders wichtig, daß die zugrundeliegenden Methoden vor der CASE-Einführung geschult und trainiert werden.

Übungsarbeitsplatz

Eine Möglichkeit, Ängste abzubauen, besteht auch darin, einen Übungsarbeitsplatz einzurichten, zu dem die betroffenen Mitarbeiter freien Zugang haben. So können sie sich mit dem neuen System »spielerisch« vertraut machen.

Information und Weiterbildung befähigen die Betroffenen zur Mitwirkung an der Veränderung. Gewarnt werden muß jedoch vor einer Pseudopartizipation, die meist schnell durchschaut wird und zu zusätzlichen Widerständen führen kann.

nur eine CASE-Umgebung

Bezogen auf CASE-Umgebungen muß jedoch darauf geachtet werden, daß nur ein Produkt firmenweit eingeführt wird.

Konsequenze der Einführu

Die Einführung und Anwendung einer Innovation kann zu folgenden **Konsequenzen** führen:

- wünschenswert oder nicht wünschenswert,
- direkt oder indirekt,
- vorhersehbar oder nicht vorhersehbar.

Diejenige Innovation wird in einem sozialen System akzeptiert, die direkte, wünschenswerte und vorhersehbare Konsequenzen bewirkt. Hat eine Innovation viele ungewünschte Konsequenzen, dann wird sie abgelehnt, selbst wenn sie zunächst akzeptiert wurde.

Die Konsequenzen können allerdings für die verschiedenen Mitglieder unterschiedlich sein. Was für die einen erwünscht ist, kann für die anderen unerwünscht sein. Dies kann zu Konflikten führen. Erfahrungsgemäß bezieht sich Widerstand gegen eine Innovation nicht auf die technischen, sondern auf die damit verbundenen sozialen Veränderungen. Hierzu zählen Veränderungen bezüglich Autonomie, Arbeitsinhalt, Status, Qualifikationsanforderungen usw.

Die Konsequenzen einer Innovationseinführung sollten offen und transparent dargestellt und diskutiert werden. Durch Partizipation können die Betroffenen die Konsequenzen mit beeinflussen.

5.6.5 Charakteristika des Kommunikationsprozesses

Kommunikatio

Die Einführung einer Innovation erfordert die Vermittlung von Informationen über die Innovation. Die zu vermittelnden Informationen lassen sich unterteilen in harte und weiche Informationen.

Harte Informat

Harte Informationen beschreiben die Details einer Innovation, welche Konzepte realisiert sind, wie die Innovation arbeitet und wie sie benutzt werden sollte.

Weiche Informationen

Weiche Informationen machen Aussagen über Kosten und Nutzen der Innovation und beschreiben die potentiellen Effekte, Implikationen und Risiken. Diese Informationen sind wichtig für Auswahl- und Einführungsentscheidungen.

Realistische Erwartungshaltung

Wichtig ist, daß sowohl die harten als auch die weichen Informationen eine **realistische Erwartungshaltung** erzeugen. Dies wird unterstützt durch die Angabe der Ziele, die mit dem Produkt erreicht und *nicht* erreicht werden können.

Beispiel

Bezogen auf CASE führen folgende globale Feststellungen zu einer realistischen Erwartungshaltung:

- Ein guter Software-Ingenieur kann mit CASE seine Produktivität und Qualität wesentlich steigern.
- Ein schlechter Software-Ingenieur kann mit CASE in noch kürzerer Zeit noch mehr schlechte Software erstellen.
- Eine CASE-Einführung erfordert eine langandauernde Anstrengung.
- Am Anfang ist nicht mit Produktivitätssteigerungen, sondern mit Produktivitätseinbußen zu rechnen.

Richtige Verpackung

Die Einführung einer Innovation wird erleichtert, wenn die Innovation richtig »verpackt« ist. Das bedeutet, daß Trainingsmaterial, realistische Fallstudien, empirische Erkenntnisse usw. zur Innovation dazugehören oder während des Einführungsprozesses erstellt werden müssen.

Kommunikationskanäle

Die Informationsvermittlung über eine Innovation kann über *Massenmedien* oder *zwischenmenschliche Kommunikation* erfolgen.

Massenmedien

Massenmedien sind schnell und effizient, wenn es darum geht, auf eine Innovation aufmerksam zu machen.

Zwischenmenschliche Kommunikation

Zwischenmenschliche Kommunikation ist effektiver, wenn es darum geht, Personen zur Einführung neuer Ideen zu bewegen.

Eine erfolgreiche Verbreitung einer Innovation erfordert ein passendes Gleichgewicht zwischen den Kanälen Massenmedien und zwischenmenschliche Kommunikation. Das geeignete Gleichgewicht hängt von den Charakteristika der Innovation und den sozialen und kulturellen Aspekten der Zielgruppe ab.

Gleichartigkeit der Zielgruppe

Die Kommunikation wird außerdem durch die Gleichartigkeit der Zielgruppe beeinflußt. Haben die Mitglieder der Zielgruppe gleiche Auffassungen, gleichartige Ausbildungen und einen ähnlichen sozialen Status, dann ist die Kommunikation effektiver.

Empfehlungen

Für eine **CASE-Einführung** ergeben sich daraus folgende Empfehlungen:

Aufgaben des Herstellers

- Der Hersteller oder Anbieter eines CASE-Produktes sollte nicht nur harte, sondern auch weiche Informationen zur Verfügung stellen. Außerdem sollte er in Massenmedien (Artikel in Fachzeitschriften und Büchern, Anzeigen) auf sein Produkt aufmerksam machen.

Aufgaben innerhalb der Firma

- Innerhalb einer Firma sollte durch zwischenmenschliche Kommunikation das Produkt diskutiert werden. Als Zielgruppe sollte eine Gruppe ausgewählt werden, deren Mitglieder gleiche Auffassungen, gleichartige Ausbildungen und einen ähnlichen sozialen Status haben.

5.6.6 Regeln zur Erleichterung einer CASE-Einführung

Verschiedene Personengruppen können dazu beitragen, daß eine CASE-Einführung von optimalen Voraussetzungen ausgehen kann.

CASE-Her CASE-Anl

Der **CASE-Hersteller** oder **CASE-Anbieter** kann folgendes tun:

- Bereitstellung von harten und weichen Informationen.
- Aufbau realistischer Erwartungen ermöglichen.
- Umfassende Unterstützung des Produktes durch gut verständliches Trainingsmaterial, realistische Fallstudien, didaktisch gut gestaltete Benutzerhandbücher, in das Produkt integrierte Tutorials und Hilfesysteme.
- Erhöhung des Bekanntheitsgrades des Produktes durch Artikel und Anzeigen in Fachzeitschriften und Büchern.
- Unterstützung von Standardmethoden, dadurch Erhöhung der Kompatibilität.
- Inkrementellen CASE-Einsatz, beginnend bei kleinen Anwendungen, ermöglichen.
- Durch grafische Darstellungen der Ergebnisse und verständliche textuelle Ausgaben die Ergebnisse des CASE-Einsatzes sichtbar machen.
- Evaluations- und Probeinstallationen ermöglichen.

Forschung

Die Software-Technik-**Forschung** kann folgendes beitragen:

- Anpassung der Erkenntnisse der Technologie-Transfer-Forschung an die Software-Technik-Charakteristika.
- Aktive Rolle als Meinungsbildner spielen.
- Durch empirische Studien zu realistischen Erwartungshaltungen beitragen.
- Ausbildung von Methodenberatern.

Manageme

Das **Management** einer Firma kann folgende Beiträge leisten:

- Schaffen einer innovationsfreundlichen Firmenkultur (liberales Umfeld, keine starren Normen).
- Stelle eines Methodenberaters schaffen und kompetent besetzen.
- Innovationsimpulse geben und Methodenberater fördern und unterstützen.
- Notwendige Ressourcen zur Verfügung stellen (Zeit und Mittel für die Weiterbildung der Mitarbeiter, Mittel für CASE-Umgebung und Entwicklungsmaschinen).
- Innovationsfreudige Mitarbeiter einstellen.

Methodenb

Der **Methodenberater** hat den meisten Einfluß auf die Akzeptanz einer Innovation. Er kann eine CASE-Einführung folgendermaßen erleichtern:

Informatio

- Frühzeitige und fortlaufende Information der betroffenen Mitarbeiter.

Partizipatio

- Den Mitarbeitern Mitwirkungsmöglichkeiten einräumen.

Weiterbildu

- Eine frühzeitige und umfassende Weiterbildung durchführen.

Einführung
- Schrittweise Einführung der neuen Methoden und der CASE-Umgebung, so daß Teilerfolge sichtbar werden.
- Einbinden von Meinungsführern.
- Versuchen, eine kollektive Akzeptanz der Innovationsentscheidung zu erhalten.
- Verstehen der sozialen Strukturen und Normen der Zielgruppe.
- Auswahl einer geeigneten Gruppe zur Durchführung des ersten Projektes.
- Beachten der unterschiedlichen Neigungen, Innovationen anzuwenden (S-Kurve).
- Bei der Auswahl der Methoden und Produkte darauf achten, daß der technologische »Sprung« für die Zielgruppe nicht zu groß ist.
- Entwicklung von Strategien, um die Probleme der Inkompatibilität zu bewältigen und den Übergang zu erleichtern.
- Auswahl einer geeigneten Kommunikationsstrategie.
- Realistische Ziele definieren und Konsequenzen offen und transparent darstellen.

Mitarbeiter

Die **betroffenen Mitarbeiter** können ebenfalls dazu beitragen, eine CASE-Einführung zu erleichtern, in dem sie
- konstruktiv bei der Auswahl und Einführung mitwirken.
- Erfahrungen und Verbesserungsvorschläge einbringen.
- keine unrealistischen Anforderungen stellen wie »alle unsere Wünsche müssen von der CASE-Umgebung zu 100 Prozent erfüllt werden«.
- Weiterbildung als Chance begreifen, die eigene Qualifikation und damit auch den eigenen »Marktwert« zu verbessern.

5.6.7 Eigenschaften eines Methodenberaters

Ein Methodenberater spielt eine zentrale Rolle bei der Einführung von Innovationen. Daher ist es entscheidend, den richtigen Mitarbeiter für diese Position zu gewinnen. Um seine Aufgaben optimal erfüllen zu können, sollte ein Methodenberater über folgende Eigenschaften verfügen:
- Er sollte umfangreiches Wissen über die Methoden der Software-Technik, ihre gegenseitigen Abhängigkeiten und ihre Trends besitzen.
- Wünschenswert sind Erfahrungen in der Methodenberatung.
- Er muß in der Lage sein, neue Methoden daraufhin zu beurteilen, ob der technische »Sprung« vom derzeitigen Technologieniveau zu den neuen Methoden vertretbar und erfolgsversprechend ist. Ziel eines Methodenberaters darf es nicht sein, von der einen wissenschaftlichen Tagung zur anderen zu fahren, die neuesten wissenschaftlichen Ideen »aufzuschnappen« und dabei seine Zielgruppe völlig aus den Augen zu verlieren.

- Er sollte einen Überblick über marktgängige CASE-Produkte haben und die sich abzeichnenden Trends kennen.
- Er muß praktische Erfahrungen in der Software-Entwicklung besitzen und selbst eine Anzahl von CASE-Werkzeugen bereits angewendet haben.
- Er muß fachliche Kenntnisse über das Anwendungsgebiet besitzen, für das Software entwickelt wird.
- Er benötigt großes psychologisches Einfühlungsvermögen, insbesondere um sich in die Rolle der Betroffenen versetzen zu können.
- Er muß sich durchsetzen können, sowohl gegenüber Vorgesetzten als auch gegenüber destruktiven Mitarbeitern.
- Er muß sich durch »echte« Mitarbeit in Projekten qualifizieren. Dadurch wird eine Isolierung von der Zielgruppe vermieden. Nur so werden seine Aussagen respektiert und ernstgenommen. Ist der Arbeitsaufwand für einen Methodenberater zu groß, dann ist es besser, zwei Methodenberater einzusetzen, die beide in Projekten mitarbeiten, als einen Methodenberater zu haben, der nur noch Technologie-Transfer betreibt.
- Er praktiziert eine »Politik der offenen Tür«, d.h. wenn Mitarbeiter Probleme haben, dann können sie den Methodenberater kurzfristig ansprechen.
- Er besitzt die Charaktereigenschaften, die man Innovatoren und frühen Anwendern zuordnet.

Da der Methodenberater »kritisch« für den Technologie-Transfer ist, dürfen die Aufgaben des Methodenberaters keine »Feigenblattfunktion« haben. Die Stelle eines Methodenberaters muß daher von den Kompetenzen und dem Gehalt so ausgestattet sein, daß qualifizierte Mitarbeiter dafür gewonnen werden können.

5.6.8 Eigenschaften des ersten Projekts

Die Auswahl des richtigen ersten Projekts spielt für den Erfolg einer Innovation eine entscheidende Rolle. Folgende Eigenschaften sollte das »erste« Projekt besitzen:

Echtes Projekt
- Es muß sich um ein »echtes« Projekt handeln. Projekte mit Fallstudiencharakter werden nicht ernstgenommen.

Normales Projek
- Es muß ein »normales« Projekt sein. Besonders riskante oder komplexe Projekte sind erst dann anzugehen, wenn ausreichende Erfahrungen über die Stärken und Schwächen der neuen Methode oder des neuen CASE-Produkts vorliegen.

Vorteile sichtbar machen
- Das Projekt sollte die Vorteile der neuen Technik sichtbar machen. Werden beispielsweise Methoden und Werkzeuge neu eingesetzt, die die Definitionsphase eines Projekts unterstützen, dann sollte das Projekt hier auch seine Schwerpunkte haben und nicht z.B. in der Algorithmik.

Bekanntes Problemfeld
- Das Projekt muß aus einem bekannten Anwendungsgebiet sein. Die Mitarbeiter müssen Erfahrungen auf dem Problemfeld haben. Es ist z.B. nicht sinnvoll, Mitarbeiter, die bisher nur kommerzielle Software-Systeme entwickelt haben, mit dem neuen CASE-Produkt ein Echtzeit-System entwickeln zu lassen.

Mittlerer Projektumfang
- Das Projekt soll einen mittleren Projektumfang besitzen. Erfolge bei kleinen Projekten werden oft nicht ernstgenommen. Große Projekte besitzen eine hohe Eigenkomplexität und es müssen viele Mitarbeiter gleichzeitig betreut werden.

Neues Projekt
- Es muß ein »neues« Projekt sein, d.h. das Projekt hat noch nicht begonnen. Es darf nicht die Situation eintreten, daß ein bereits laufendes Projekt in Schwierigkeiten gerät und die neue Methode oder das neue CASE-Produkt das Projekt jetzt »retten« soll.

Kein extremer Termindruck
- Der Termindruck darf nicht zu groß sein. Gerade die erste Anwendung einer neuen Technologie erfordert zusätzliche Zeit, da Verhaltensänderungen erforderlich sind und jeder Schritt bewußt vollzogen werden muß.

Positiv eingestellte, qualifizierte, »normale« Mitarbeiter
- Es sind qualifizierte Mitarbeiter auszuwählen, die der neuen Technologie gegenüber positiv eingestellt sind. Dies gilt besonders für den Projektleiter. Das Team darf aber nicht nur aus »Stars« zusammengesetzt sein, sonst sind die Erfolge des ersten Projekts bei späteren Projekten nicht wiederholbar.

Training vor Projektbeginn
- Das Projektteam ist unmittelbar vor der Anwendung der neuen Methoden bzw. CASE-Werkzeuge zu trainieren. Ein *on-the-job*-Training während des Projekts ist zu vermeiden, da es den Projektfortschritt behindert.

Mentor gewinnen
- Es sollte ein Mentor aus dem oberen Management für das erste Projekt gewonnen werden *(organizational champion)*. Dadurch erhält das Projekt die notwendige Aufmerksamkeit und die Chancen für einen Erfolg werden erhöht.

Metriken ermitteln
- Das Projekt sollte während des Projektablaufs durch Metriken »vermessen« werden, so daß laufend aktuelle, nachvollziehbare Informationen vorliegen. Das Problem besteht oft darin, daß keine vergleichbaren Metriken aus anderen Projekten vorliegen.

Besitzt ein in Betracht gezogenes Projekt mehrere der beschriebenen Eigenschaften nicht, dann sollte der Methodenberater das Projekt als ungeeignet für den Ersteinsatz der neuen Technologie ablehnen. Kompromisse in dieser Frage erhöhen u.U. drastisch das Risiko des Technologie-Transfers.

Literaturhinweise

In /Yourdon 86/ werden die Eigenschaften eines Pilotprojektes für die Einführung neuer Methoden angegeben. /Fischer 88/ behandelt die Fragestellung für CASE-Produkte.

5.6.9 Beispiel einer Migrationsstrategie

In der Regel müssen in der Software-Technik nicht nur Innovationen eingeführt werden, sondern es muß auch überlegt werden, wie die bisherige Software-Entwicklung auf die Innovation umgestellt werden soll.

Migration

Solche Umstellungen bezeichnet man in der Software-Technik als **Migration**. Das Wort stammt aus dem Lateinischen (migratio) und bedeutet »(Aus)wanderung« oder »Übersiedlung«, im übertragenen Sinne »Transportieren«. Bei jeder Migration ist stets zu fragen /Stahlknecht 93, S. 310/,

- *Was* umgestellt wird,
- *Wozu* umgestellt wird und
- *Wie* umgestellt wird.

Beispiel: strukturiert ≠ objektorientiert

Als Beispiel für eine Migration wird im folgenden die Umstellung von einer strukturierten Software-Entwicklung auf eine objektorientierte Software-Entwicklung betrachtet (Was). Die Umstellung erfolgt, um die Vorteile der Objektorientierung nutzen zu können (Wozu).

Für das »Wie« gibt es zwei Alternativen:

1 Alles auf einmal?
Die Objektorientierung wird in den Phasen Definition, Entwurf und Implementierung gleichzeitig eingeführt.

2 Schritt für Schritt?
Es wird zunächst nur in einer Phase die Objektorientierung eingeführt. Nach der Etablierung in dieser Phase erfolgt die Einführung in einer weiteren Phase. Bei der schrittweisen Migration stellt sich die Frage, in welcher Phase am besten mit der Objektorientierung begonnen wird. In der Praxis findet man zwei Alternativen:

a Objektorientierung zunächst in der Definitionsphase einführen, d.h. mit der objektorientierten Analyse (OOA) beginnen.

b Objektorientierung zunächst in der Implementierungsphase einführen, d.h. zunächst eine objektorientierte Programmiersprache (OOP) einführen.

Vor- und Nachte

Die erste Alternative »Alles auf einmal« birgt ein hohes Risiko, vermeidet aber jeden Zusatzaufwand.

Die zweite Alternative »Schritt für Schritt« ist weniger risikoreich, erfordert aber einen Zusatzaufwand, z.B. zusätzliche Transformationen in eine klassische Programmiersprache.

Empfehlungen bei schrittweise Umstellung

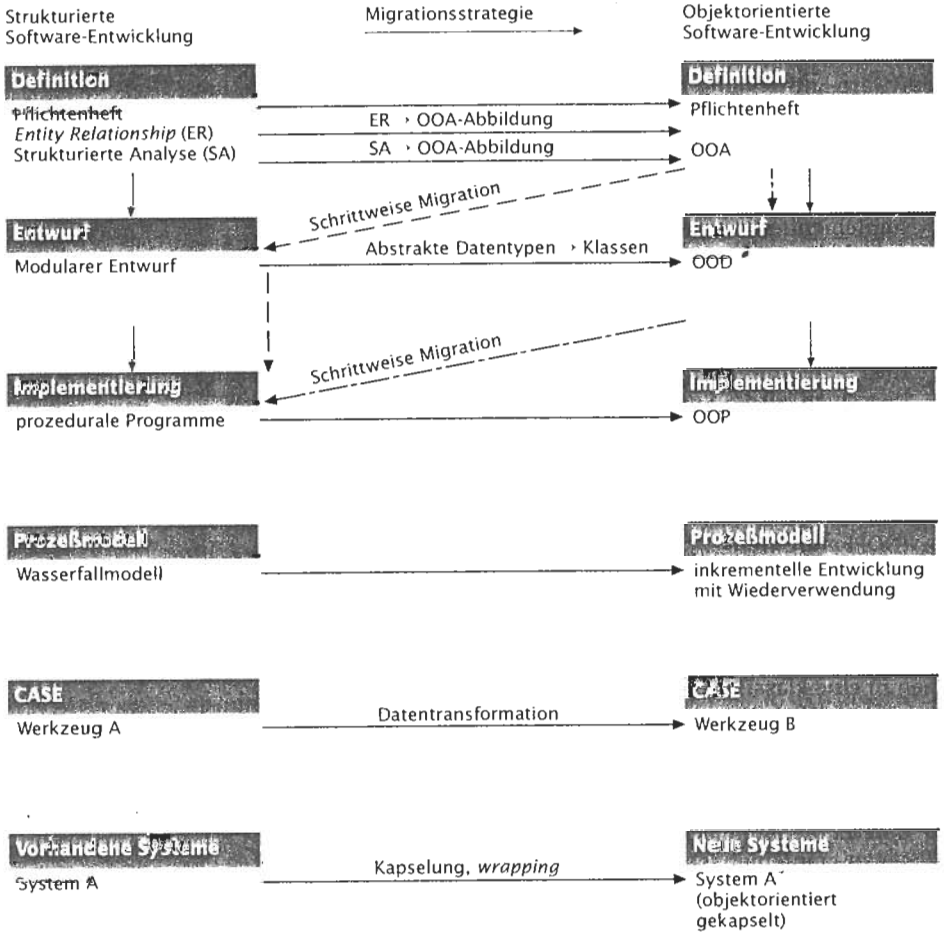
Erfolgt eine schrittweise Umstellung, dann sollte

- zuerst die Definitionsphase auf Objektorientierung umgestellt werden,
- dann die Entwurfsphase und
- zuletzt die Implementierungsphase.

Nicht zu empfehlen ist die Reihenfolge:
Zuerst Implementierung, dann Entwurf und zuletzt Definition.

Wird zuerst auf eine objektorientierte Programmiersprache umgestellt, dann wird versucht, in der Implementierungsphase Klassen zu identifizieren und diese direkt in der Notation der verwendeten Programmiersprache zu beschreiben. Dabei besteht die große Gefahr, daß fachbezogene, technische und programmiersprachenspezifische Aspekte »wild« gemischt werden. Viele Firmen, die diese Migrationsstrategie gewählt haben, sind daran gescheitert. Die Ursache, eine solche Migrationsstrategie zu wählen, liegt oft darin, daß es in diesen Firmen bisher keine »saubere« Definitions- und Entwurfsphase gibt, abgesehen von einem mehr oder weniger guten verbalen Pflichtenheft.

Abb. 5.6-5: Beispiel für eine Migrationsstrategie



Ist die grundlegende Migrationsstrategie festgelegt, dann ist detailliert festzulegen, wie phasenweise eine Umstellung erfolgen soll und wie bereits vorhandene Systeme in die neue Welt überführt werden sollen.

Die detaillierte Migration hängt natürlich stark von der bisherigen Ausgangssituation ab. Abb. 5.6-5 zeigt für eine gegebene Ausgangssituation mögliche Migrationen. Natürlich wird man keine bereits begonnene Entwicklung umstellen (siehe Abschnitt 5.6.8).

Die Abbildung bekannter Konzepte und Methoden auf die neuen Konzepte und Methoden ist jedoch für das Training wichtig, um ein »Lernen durch Analogien« zu erleichtern /Manns, Nelson 96/. Außerdem ist sie für Sanierungs-Projekte wichtig.

Hauptkapitel IV

Liegt als Unternehmensmodell beispielsweise ein *Entity Relationship*-Modell vor, dann kann ein solches Modell systematisch in ein OOA-Modell transformiert werden (Abb. 5.6-6).

Ein SA-Modell kann mit dem in Abschnitt I 3.9.7 beschriebenen Verfahren zumindest ansatzweise in ein OOA-Modell transformiert werden. Die Datenflußdiagramme können für die Klassenbildung ausgeweitet werden. Die *Data Dictionary*-Einträge können für die Attributspezifikation, die Minispecs für die Spezifikation der Operationen verwendet werden.

Abschnitt I 3.9.

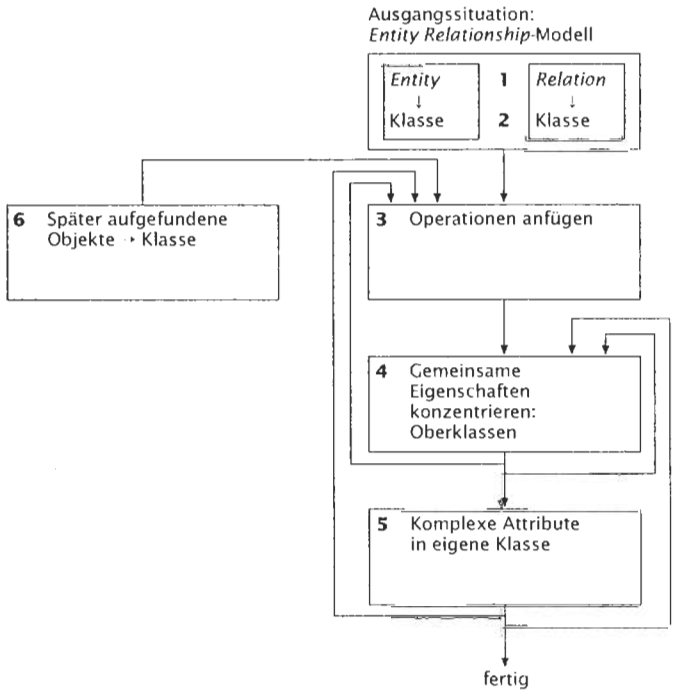


Abb. 5.6-6: Migration eines ER-Modells in ein OOA-Modell (in Anlehnung an /Offerbein 93/).

OOP ≠ prozedurale Sprache

Wählt man eine schrittweise Migration, dann gibt es in der Literatur, z.B. in /Coad, Yourdon 91/, /Coad, Nicola 93/, genügend Vorschläge, um beispielsweise einen objektorientierten Entwurf in eine prozedurale Programmiersprache zu transformieren.

Prozeßmodell

Zu prüfen ist, ob das bisher verwendete Prozeß- bzw. Vorgehensmodell noch adäquat ist. Zumindest ist das vorhandene Modell um das Suchen und Ablegen wiederverwendbarer Komponenten zu ergänzen.

Abschnitt I 3.3.5

Eine bereits vorhandene CASE-Umgebung muß entweder um Werkzeuge für die Objektorientierung ergänzt werden (wenn dies der Hersteller anbietet) oder es muß eine neue CASE-Umgebung beschafft werden. In diesem Fall sollte es eine Importmöglichkeit für Daten aus der vorhandenen CASE-Umgebung geben, damit bei Sanierungsprojekten auf die »alten« Daten zugegriffen werden kann.

CASE Hauptkapitel IV 2

Kapselung, *wrapping*

In der Regel werden laufende Anwendungen, die noch nicht »sanierungsreif« sind, mit den neu entwickelten Anwendungen kooperieren müssen. Eine Möglichkeit besteht darin, die vorhandene Anwendung »objektorientiert« zu verpacken *(wrapping)* oder zu verkapseln, so daß sie sich nach außen hin objektorientiert verhält, z.B. durch die Bereitstellung von Operationen.

Abschnitt IV 4.3.1

Literaturhinweise

In der Literatur gibt es zu verschiedenen Migrationsfragestellungen Vorschläge und Hinweise:

- Einführung der Objektorientierung (allgemein): /OOPSLA 91/, /Lee 94/, /Lindner 95/, /Callaghan 96/, /Swanstrom 95/
- Übergang auf objektorientierte Programmiersprachen: /Pinso 94/, /Pflüger, Roth, Schmidt 93/
- Übergang auf objektorientierte Datenbanken: /Dittrich 93/
- Kapselung von vorhandenen Systemen: /Deubler, Koestler 94/

5.6.10 Die Lernkurve

Bei allen Innovationseinführungen ist die Lernkurve zu beachten. Um neue Fähigkeiten zu erlernen, benötigt man Zeit. Im Laufe der Zeit beherrscht man diese Fähigkeiten immer besser, und die Tätigkeiten, die man mit diesen Fähigkeiten ausführt, werden schneller und besser erledigt. Die Lernkurve zeigt, wie sich diese zunehmende »Routine« bei der Anwendung neu erlernter Fähigkeiten auf die Produktivität oder andere Eigenschaften niederschlägt (Abb. 5.6-7).

Die Abbildung zeigt, daß die Projektkosten des ersten »neuen« Projekts relativ gesehen höher sind als die der weiteren Projekte. Das erste Projekt mit neuer Technologie ist sogar teurer als Projekte, die mit der alten Technologie durchgeführt wurden. Dieser Effekt wird von den Anwendern einer Innovation normalerweise nicht erwartet und daher auch nicht berücksichtigt.

Lernkurven wurden 1925 in der Luftfahrtindustrie endeckt. T.P. Wright beobachtete, daß die Kosten, um ein Flugzeug zu bauen, mit

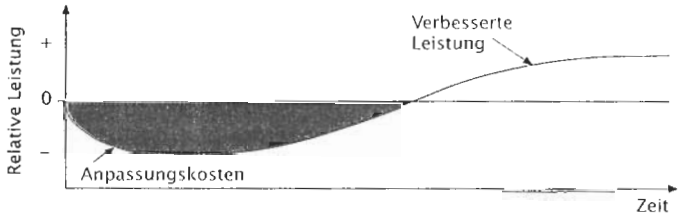


Abb. 5.6-7: Leistung über Zeit mit Lernef /Kemerer 92, S. 24/

der Anzahl der gebauten Flugzeuge abnehmen. Die Kostenabnahme entsprach einem bestimmten Muster. Wright stellte ebenfalls fest, daß dieses Muster mit jeder neuen Produktionsserie von vorne begann. Entsprechende Lernkurven wurden später auch in der Fertigungsindustrie, im Bergbau und im Konstruktionswesen festgestellt. Seit den 90er Jahren beobachtet man Lernkurven auch in der Software-Technik, siehe z.B. /Kemerer 92/.

Eine **Lernkurve** gibt an, wie die durchschnittlichen Stückkosten einer Produktion in Abhängigkeit von der kumulativen Anzahl der produzierten Einheiten sinken (Abb. 5.6-8).

Lernkurve

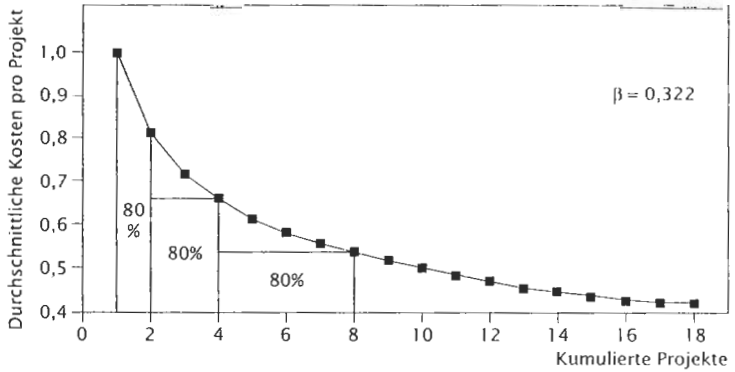


Abb. 5.6-8: Eine traditione 80 Prozent-Le kurve /Kemer 92, S. 26/

Das Gefälle der Kurve wird durch die Lernrate bestimmt. Sie beschreibt, wie die Stückkosten sinken, wenn sich jeweils die kumulierte Anzahl der produzierten Einheiten verdoppelt. Eine 80 Prozent-Lernrate bedeutet, daß zu jedem Zeitpunkt, in dem sich die kumulierten Einheiten verdoppelt haben, die Stückkosten auf 80 Prozent sinken. Wenn das 100ste Stück 1 DM kostet, dann kostet das 200ste Stück 0,80 DM und das 400ste Stück 0,64 DM.

Die früheste industrielle Lernkurve ist die Wright-Kurve bzw. die kumulierte Durchschnittskurve, dargestellt durch:

$$y = \alpha X^{\beta} + \varepsilon \,,\ \beta > 0$$

wobei y die durchschnittlichen Kosten, α die Kosten der ersten Einheit, X die Anzahl aller Einheiten und β die Lernrate bezeichnet. β kann folgendermaßen bestimmt werden:

$$\ln y = \ln \alpha - \beta \ln X.$$

β wird oft in Prozent ausgedrückt: $\beta = \ln(\%)/\ln 2$

Typische Prozentraten, die in der Praxis beobachtet werden, liegen zwischen 70 und 95 Prozent. Je kleiner die Lernrate ist, desto schneller verläuft der Lernprozeß und desto schneller sinken die Stückkosten.

Lernkurven beschreiben nicht nur das individuelle Lernen, sondern auch das Lernverhalten von Teams und Organisationen. Verschiedene Faktoren beeinflussen die Lernkurve. Dazu gehören:

- Arbeitseffizienz in der Produktion und im Management,
- verbesserte Methoden und Technologien,
- Produktverbesserungen durch Reduktion oder Elimination von kostenträchtigen Merkmalen und
- Produktionsstandardisierung durch Reduktion von Änderungen.

Alle diese Faktoren treffen auch auf die Software-Entwicklung zu.

Lernkurven können auch dazu verwendet werden, Kosten zu schätzen (Abb. 5.6-9).

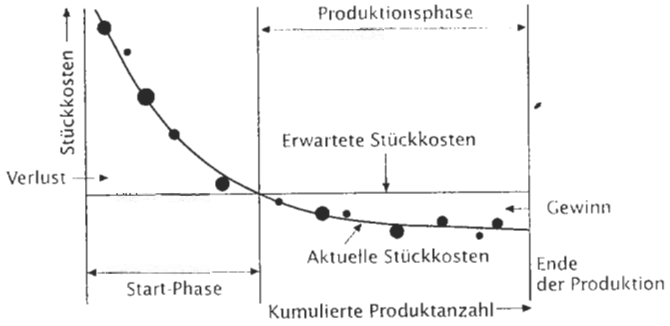


Abb. 5.6-9: Lernkurve zur Kostenbestimmung (in Anlehnung an /Raccoon 96, S. 78/)

Folgerungen

Aus Lernkurven lassen sich folgende Schlußfolgerungen ziehen (siehe auch /Raccoon 96/):

- Bei der Einführung von Innovationen ist davon auszugehen, daß die ersten Projekte teurer sind als die bisherigen Projekte (ohne Innovationen).
- Mit jedem zusätzlichen Projekt mit der eingeführten Innovation sinken die Kosten durch den Lerneffekt der Mitarbeiter.
- Mitarbeiter sind am Ende eines Projektes am produktivsten, daher sollten sie es nicht vor Projektende verlassen.
- Für kurze Projekte sind Mitarbeiter mit Erfahrungen auf dem entsprechenden Gebiet am geeignetsten, da nicht genug Zeit bleibt, um Erfahrungen zu sammeln.
- Für lange Projekte sind Mitarbeiter, die am schnellsten lernen, am geeignetsten, da sie ihre gelernten Fähigkeiten in dem Projekt noch genügend einsetzen können.

Innovation Die planvolle, zielgerichtete Erneuerung und auch Neugestaltung von Teilbereichen, Funktionselementen oder Verhaltensweisen im Rahmen eines bereits bestehenden Funktionszusammenhangs (soziale oder wirtschaftliche Organisation) mit dem Ziel, entweder bereits bestehende Verfahrensweisen zu optimieren oder neu auftretenden und veränderten Funktionsanforderungen besser zu entsprechen (Brockhaus 89).

Lernkurve Graduelle Verbesserung (Produktivität, Qualität, Unfallhäufigkeit) einer Aufgabenausführung (Fertigung, Entwicklung) über die Zeit hinweg. Auch für Software-Projekte empirisch nachgewiesen.

Migration Jede Art von Umstellung in der Software-Technik, meist im Zusammenhang mit der Einführung von → Innovationen.

Eine zentrale Aufgabe des Software-Managements besteht darin, Innovationen der Software-Technik in der eigenen Software-Entwicklung einzuführen. Verbunden mit einer Innovationseinführung sollte immer eine Migrationsstrategie sein. Zu berücksichtigen ist außerdem die Lernkurve. Zur Erleichterung von Innovationseinführungen sollten die bestimmenden Faktoren der Innovation selbst, der Individuen, des sozialen Systems und der Kommunikation so beeinflußt werden, daß der Transfer-Prozeß von günstigen Voraussetzungen ausgehen kann.

Faktoren einer Innovationseinführung

Der Erfolg einer Innovationseinführung hängt neben den oben genannten Faktoren natürlich vom Grad der Innovation bezogen auf das jetzige Technologieniveau ab (Abb. 5.6-10). Werden überhaupt noch keine modernen Prinzipien, Methoden und Werkzeuge angewandt, und soll ein akzeptables Technologieniveau erreicht werden, dann erfordert dies natürlich umfassendere Maßnahmen als die singuläre Einführung einer neuen Methode oder eines einzelnen CASE-Werkzeuges.

Soll z.B. eine moderne CASE-Umgebung einschließlich der damit verbundenen Methoden eingeführt werden, dann handelt es sich um eine **Systemeinführung** und keine Produkteinführung. Unter System

Systemeinführ

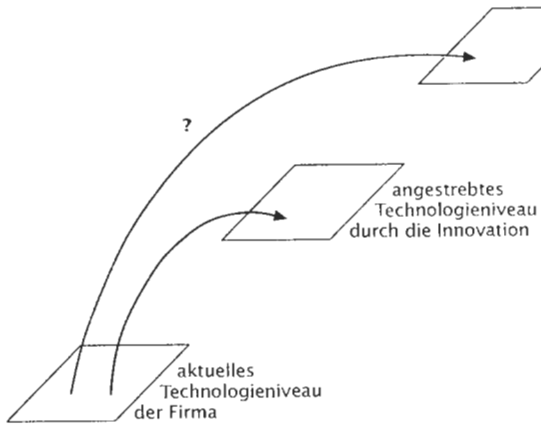


Abb. 5.6-10: Technologiesp oder evolution Weiterentwickl

ist in diesem Zusammenhang die Kombination von Methoden, Produkten, Infrastruktur, Training und Support zu verstehen. Analog handelte es sich bei der Einführung des Jumbo-Jet durch eine Fluggesellschaft um eine Systemeinführung.

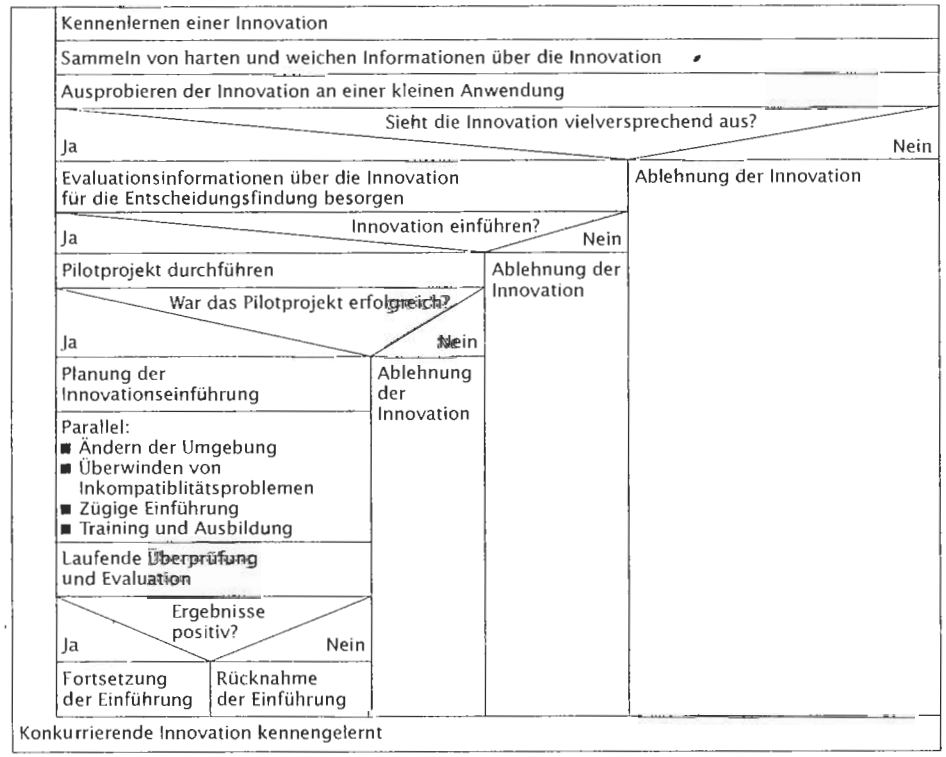
unternehmensstrategische Entscheidung

Da eine Systemeinführung ein Unternehmen auf fünf bis zehn Jahre bindet, ist eine solche Entscheidung eine **unternehmensstrategische Entscheidung** für software-orientierte Unternehmen. Daher ist eine solche Entscheidung auch auf der entsprechenden Managementebene zu treffen.

Vor einer Fehlentwicklung muß jedoch gewarnt werden. Drückt sich das Management vor einer solchen Entscheidung oder übersieht den notwendigen technologischen Wandel, dann kann das dazu führen, daß auf der Projektebene singuläre und unterschiedliche Insellösungen entstehen, die nicht kompatibel und zukunftsorientiert sind.

Diese »kleinen« Lösungen sind zwar leichter einzuführen und zeigen auch punktuelle Erfolge. Der entstehende ungeplante »Wildwuchs« kostet im Endeffekt jedoch mehr und ist kaum noch auf eine einheitliche Linie zu bringen.

Abb. 5.6-11: Typischer Entscheidungsprozeß zur Einführung von Innovationen (in Anlehnung an /Raghavan, Chand 89, S. 84/)



Folgende Größen sind die kritischen Erfolgsfaktoren:

- Das verantwortliche Management muß die Innovation wollen, unterstützen und sogar initiieren.
- Die Durchführung und Betreuung obliegt einem hauptamtlichen Methodenberater.
- Die betroffenen Mitarbeiter werden informiert, trainiert und können mitwirken.
- Die Ziele und Erwartungshaltungen sind realistisch und können schrittweise erreicht werden.
- Im Vordergrund stehen Prinzipien und Methoden, erst dann kommen Werkzeuge.

Sind ein oder zwei dieser Faktoren nicht erfüllt, dann ist die Wahrscheinlichkeit des Scheiterns sehr hoch. Einen typischen Entscheidungsprozeß zur Einführung von Innovationen zeigt Abb. 5.6-11.

/Bouldin 89/
Bouldin B. M., *Agents of Change – Managing the Introduction of Automated Tools*, Englewood Cliffs: Prentice Hall, 1989, 198 Seiten
Die Autorin beschreibt aus der Sicht eines Innovationseinführers die notwendigen Vorgehensschritte und gibt praktische Ratschläge, was zu berücksichtigen und zu beachten ist. Nicht speziell auf CASE, sondern allgemein auf Software-Produkte bezogen.

/Fisher 88/
Fisher A. S., CASE – *Using Software Development Tools*, New York: John Wiley & Sons, 1988, pp. 220–236
Beschreibt auf den angegebenen Seiten die Einführung von CASE in einem Unternehmen.

/Humphrey 90/
Humphrey W. S., *Managing the Software Process*, Reading: Addison-Wesley 1990
Beschreibt in umfassender Weise verschiedene Technologie-Niveaus einer Firma bezogen auf die Software-Technik. Standardwerk, das den Reifegrad eines Unternehmens nach den SEI-Reifestufen beschreibt (SEI = *Software Engineering Institute*). Enthält auch Aussagen zum Transferprozeß.

/Pressman 88/
Pressman R. S., *Making Software Engineering Happen – A Guide for Instituting the Technology*, Englewood Cliffs: Prentice Hall 1988, 258 Seiten
Beschreibt in umfassender Weise die Einführung moderner Software-Technik in ein Unternehmen.

/Rogers 83/
Rogers E. M., *Diffusion of Innovations*, Third Edition, New York: Free Press, 1983
Erstes Standardwerk über den Technologie-Transfer. Die erste Auflage erschien 1962. Beschreibt einen weitgehend allgemeingültigen Rahmen für die Einführung von Innovationen. Dieser Rahmen wurde durch Anwendung auf vielfältige Gebiete empirisch abgesichert und bestätigt. Das Gebiet Software-Technik wird nicht behandelt.

/Schnaars 89/
Schnaars S. P., *Megamistakes – Forecasting and the Myth of Rapid Technological Change*, New York: The Free Press 1989
Kritische Hinterfragung von Vorhersagen. Spannend zu lesen!

/Spinas, Troy, Ulich 83/
Spinas P., Troy N., Ulich E., *Leitfaden zur Einführung und Gestaltung von Arbeit mit Bildschirmsystemen*, München: CW-Publikationen, 1983, 116 Seiten

Dieser Leitfaden enthält ein Kapitel »Psychologische Aspekte der Planung und Einführung von Veränderungen« sowie eine Checkliste, die technisch organisatorische Veränderungen in einem Unternehmen generell behandelt. Viele dieser Gesichtspunkte sind auch für die Einführung von CASE relevant. Das Kapitel einschließlich der Checkliste besteht aus 11 Seiten und ist allgemeinverständlich geschrieben.

/Yourdon 86/
Yourdon E., *Managing the Structured Techniques - Strategies for Software Development in the 1990's*, Englewood Cliffs: Yourdon Press 1986, Third Edition, pp. 191–194
Beschreibt in einem Kapitel, wie ein Pilotprojekt ausgewählt werden soll.

Zitierte Literatur

/Asthana 95/
Asthana P., *Jumping the Technology S-Curve*, in: IEEE Spectrum, June 1995, pp. 49–54
/Callaghan 96/
Callaghan A., *Choosing Expert Help to Cross the Paradigm Gap*, in: Object Expert, Jan.-Febr. 1996, pp. 57–59
/Coad, Nicola 93/
Coad P., Nicola J., *Object-orientedProgramming*, Englewood Cliffs: Prentice Hall 1993
/Coad, Yourdon 91/
Coad P.,Yourdon E.,*Object-oriented Design*, Englewood Cliffs: Yourdon Press, 1991
/Deubler, Koestler 94/
Deubler H.-H., Koestler M., *Introducing Object Orientation into Large and Complex Systems*, in: IEEE Transactions on Software Engineering, Nov. 1994, pp. 840–848
/Dittrich 93/
Dittrich K. R., *Migration von konventionellen zu objektorientierten Datenbanken: soll man, muß man – oder nicht?*, in: Wirtschaftsinformatik, 35 (1993) 4, S. 346–352
/Kemerer 92/
Kemerer C. F., *How the Learning Curve Affects CASE Tool Adaption*, in: IEEE Software, May 1992, pp. 23–28
/Lee 94/
Lee W., *How to Adapt OO Development Methods in a Software Development Organisation – A Case Study*, in: Addendum to the Proceedings, OOPSLA 94, pp. 19–24
/Lindner 95/
Lindner U., *Für's erste stehen die beiden Paradigmen nebeneinander*, in: Computerwoche 3, 20.1.1995, S. 43–44
/Manns, Nelson 96/
Mann M. L., Nelson H. J., *Retraining procedure-oriented developers: An issue of skill transfer*, in: JOOP, Nov.-Dec. 1996, pp. 6–10
/Offerbein 93/
Offerbein T., *Erreichen von Wiederverwendbarkeit und Erweiterbarkeit von Klassenbibliotheken am Beispiel eines Leitstandes*, in: Konferenzunterlagen I.I.R. ReUse-Konferenz, 8.-9. Sept. 1993, München
/OOPSLA 91/
Managing the Transition to Object-Oriented Technology, in: Addendum to the Proceedings, Ponal, OOPSLA 91, Phoenix, Arizona, pp. 55–62
/Pflüger, Roth, Schmidt 93/
Pflüger C., Roth H., Schmidt K. P., *Umstellung alter COBOL-Programme in objektorientierte Systeme*, in: Wirtschaftsinformatik, 35 (1993) 4, S. 353–359
/Pinson 94/
Pinson L. J., *Moving from COBOL to C and C++: OOP's biggest challenge*, in: JOOP, Oct. 1994, pp. 54–56

/Raccoon 96/
Raccoon L. B. S., *A Learning Curve Primer for Software Engineers*, in: Software Engineering Notes, Vol. 21, No 1, Jan. 1996, pp. 77–86
/Raghavan, Chand 89/
Raghavan S. A., Chand D. R., *Diffusing Software Engineering Methods*, in: IEEE Software, July 1989, S. 81–90
/Redwine, Riddle 86/
Redwine S. T., Riddle W. E., *Software Technology Maturation*, in: Proceedings »International Conference On Software Engineering«, 1986, S. 189–200
/Stahlknecht 93/
Stahlknecht P., *Migration – und kein Ende*, in: Wirtschaftsinformatik, 35 (1993) 4, S. 309–310
/Swanstrom 95/
Swanstrom E., *Beyond methodology transfer: O-O mentoring meets project management*, in: JOOP, March-April 95, pp. 57–59

1 *Lernziel: Die Personenkategorien, bezogen auf eine Innovationseinführung, nennen und charakterisieren sowie die S-Kurve erläutern können.* — Muß-Aufgabe, *10 Minuten*
Versuchen Sie die charakterlichen Eigenschaften der Personenkategorien bei einer Innovationseinführung zu beschreiben.

2 *Lernziel: Darlegen können, welche Personengruppen auf welche Weise eine CASE-Einführung erleichtern können.* — Muß-Aufgabe, *10 Minuten*
Sie stellen CASE-Werkzeuge her. Was müssen Sie tun, um die Einführung Ihrer CASE-Werkzeuge in fremden Unternehmen zu erleichtern?

3 *Lernziel: Darlegen können, welche Personengruppen auf welche Weise eine CASE-Einführung erleichtern können.* — Kann-Aufgabe, *10 Minuten*
Sie haben eine neue Test-Methode entwickelt. Was müssen Sie tun, um die Einführung zu erleichtern?

4 *Lernziel: Die Eigenschaften von Lernkurven erklären und auf Problemstellungen der Software-Technik anwenden können.* — Muß-Aufgabe, *10 Minuten*
Steht das Brooks'sche Gesetz *»Adding manpower to a late software project makes it later«* (Abschnitt 3.2.4) im Einklang mit der Lernkurve? Begründen Sie Ihre Meinung!

5 *Lernziele: Anhand von vorgegebenen Szenarien prüfen können, inwieweit eine Innovation anhand der fünf Charakteristika leicht oder schwer einzuführen ist. Für vorgegebene Szenarien eine Innovationseinführung planen und begründen können.* — Muß-Aufgabe, *30 Minuten*
In einem großen Unternehmen soll ein Netzwerk mit Novell-*Servern* und Windows 3.1-*Clients* auf eine Kombination mit Windows NT-*Servern* und Windows NT-*Workstations* umgestellt werden. Die neuen Betriebssysteme sind aus Fachzeitschriften hinlänglich bekannt. Sie erlauben den Mitarbeitern die Verwendung von neuen Anwendungen, allerdings ist das Erlernen einer neuen Benutzungsoberfläche erforderlich. Einige Mitarbeiter kennen die neue Benutzungsoberfläche und Teile der neuen Anwendungen bereits von ihrem privaten PC. Die Umstellung auf der *Client*-Seite kann in mehreren Schritten erfolgen, da Windows NT-*Server* auch mit dem alten *Client*-Betriebssystem zusammenarbeiten kann.
a Beurteilen Sie anhand der fünf Charakteristika einer Innovation den Schwierigkeitsgrad der Innovation.
b Beschreiben Sie, auf welche Weise Sie das neue Betriebssystem einführen würden.

Muß-Aufgabe *10 Minuten*

6 *Lernziel: Die Eigenschaften eines Methodenberaters skizzieren können.*
Ein Unternehmen will die Stelle eines Methodenberaters schaffen. Dazu gibt die Firma folgende Stellenanzeige auf:
»Junges, dynamisches Unternehmen aus dem Bereich Software-Entwicklung sucht einen Software-Ingenieur für die Stelle eines Methodenberaters. Sie sollen im Unternehmen CASE einführen und dazu eine geeignete Methode und ein Werkzeug auswählen sowie die Mitarbeiter schulen.
Sie sollten bereits über Erfahrungen in diesem Bereich verfügen und ein umfangreiches Wissen über die Methoden der Software-Technik verfügen. Wir erwarten von Ihnen gute Englischkenntnisse, die Bereitschaft zum Lernen und eine gute Urteilsfähigkeit. Bei Interesse schicken Sie bitte Ihre vollständigen Bewerbungsunterlagen an ... «.
Auf welche Fähigkeiten sollte das Unternehmen zusätzlich noch Wert legen?

Muß-Aufgabe *20 Minuten*

7 *Lernziel: Beurteilen können, ob ein Software-Entwicklungsprojekt für die Einführung einer Innovation geeignet ist.*
Im folgenden wird Ihnen ein Software-Entwicklungsprojekt vorgestellt. Die Firma, die dieses Projekt durchführt, arbeitet bislang mit einer objektorientierten Definitionsphase und führt in erster Linie Projekte im kaufmännisch/administrativen Umfeld durch. Im Rahmen dieses Projekts will man auch den Entwurf und die Implementierung durchführen. Halten Sie das für sinnvoll? Begründen Sie Ihre Meinung und nennen Sie weitere Randbedingungen, die für einen erfolgreichen Projektabschluß erforderlich sind. Es handelt sich um folgendes Projekt:
Es soll ein System zur Verwaltung von Lehrveranstaltungen an einer Universität entwickelt werden. Das System umfaßt die Planung der Raumbelegung, verwaltet Dozenten und soll über das campusweite Informationssystem abrufbar sein. Die ausführende Firma rechnet mit einem Umfang von 5 Mitarbeiterjahren. Üblicherweise wickelt das Unternehmen Aufträge mit einem Umfang von 2 bis 10 Mitarbeiterjahren ab. Für dieses Projekt sind 8 Mitarbeiter vorgesehen, die 10 Monate Zeit haben. Zwei der für das Projekt vorgesehenen Mitarbeiter haben bis vor kurzem für eine andere Firma gearbeitet und Erfahrungen in objektorientierter Programmierung gesammelt.

Muß-Aufgabe *5 Minuten*

8 *Lernziel: Die bestimmenden Faktoren des Technologie-Transfers kennen und an Beispielen erläutern können.*
Nennen Sie grob die Faktoren, die die Ausbreitung einer Innovation in einer Zielgruppe bestimmen. Betrachten Sie dabei auch den Einfluß des Managements und der betroffenen Mitarbeiter.

Muß-Aufgabe *10 Minuten*

9 *Lernziele: Die Charakteristika des sozialen Systems aufzählen und ihre Auswirkungen auf den Innovationsprozeß beschreiben können. Den Kommunikationsprozeß charakterisieren und seinen Einfluß auf den Innovationsprozeß darstellen können.*

a Wie beeinflussen das soziale System eines Unternehmens und der Kommunikationsprozeß die Einführung von Innovationen?

b Was kann man tun, um das soziale System innovationsfreundlicher zu gestalten?

Weitere Aufgaben befinden sich auf der beiliegenden CD-ROM.

6 Kontrolle

verstehe

- Die Konzepte Konfiguration, Version und Variante erklären können.
- Ziele, Aufgaben, Werkzeuge und Dokumente des Konfigurationsmanagements nennen und erläutern können.
- Ziele, Aufgaben und Probleme der Kontrolle schildern können.
- Ziele, Aufgaben und Probleme der Software-Meßtechnik beschreiben können.
- Metriken klassifizieren sowie Gütekriterien angeben und Beispiele nennen können.
- Anhand von Beispielen Konfigurationen beschreiben und Versionszählungen durchführen können. anwende
- Anhand von Szenarien Konfigurations- und Änderungsmanagement unter Berücksichtigung der verschiedenen Zustandsübergänge exemplarisch durchführen können.
- Gegebene Metriken klassifizieren und auf Gütekriterien hin überprüfen können.
- Das verwendete Projektplanungssystem für die Projektkontrolle einsetzen können.
- Das *Checkin/Checkout*-Modell auf Beispiele anwenden können.

Auf der beigefügten CD-ROM befinden sich mehrere Werkzeuge zum Konfigurationsmanagement.

6.1 Grundlagen

Kontrolle

Zur **Kontrolle** einer Software-Entwicklung gehören alle Managementaktivitäten, die sicherstellen, daß die laufenden Tätigkeiten mit dem Plan übereinstimmen.

Literatur: /Thayer 90, S. 43 ff/

Pläne legen Produktanforderungen, Zeit und Kosten fest. Für die Ausführung werden außerdem Prozeßmodelle, Richtlinien, Methoden und Werkzeuge vorgegeben, die im folgenden zusammengefaßt als Standards bezeichnet werden.

Die Ausführung wird gegen den Plan und die Standards gemessen. Treten Abweichungen auf, dann werden sie aufgezeigt. Aktionen werden gestartet, um Abweichungen zu korrigieren, damit Plan und Standards eingehalten werden können. Der grundlegende Kontrollprozeß (Abb. 6.1-1) beinhaltet

- das Einrichten von Plänen und Standards,
- das Messen der Ausführung gegen diese Pläne und Standards sowie
- die Korrektur der Abweichungen.

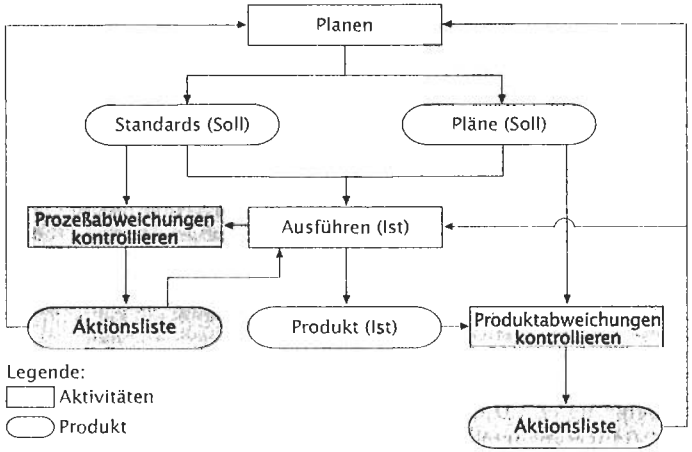
Kontrolle ist ein Rückkopplungssystem, das Informationen darüber bereitstellt, wie gut die Entwicklung, bezogen auf das Produkt und den Prozeß, voranschreitet.

Der Kontrollprozeß muß in das jeweilige Prozeßmodell und in die organisatorische Struktur integriert werden. Es muß beispielsweise festgelegt werden, wer für das Messen des Entwicklungsfortschritts verantwortlich ist. Wer unternimmt Aktionen, wenn Probleme berichtet werden?

Anforderungen an Kontroll-Methoden & -Werkzeuge

Methoden und Werkzeuge zur Kontrolle müssen *objektiv, anpaßbar* und *ökonomisch* sein. Abweichungen vom Plan und den Standards müssen ohne Rücksicht auf die betroffenen Mitarbeiter und Stellen aufgezeigt werden. Methoden und Werkzeuge müssen auf die

Abb. 6.1-1: Der prinzipielle Kontrollprozeß



firmenspezifischen Umgebungen zuschneidbar sein und sich den wechselnden Situationen anpassen können. Die Kosten der Kontrolle dürfen die Vorteile, die die Kontrolle bringt, nicht übersteigen.

Kontrolle muß zu korrigierenden Aktionen führen, entweder um den Prozeß zu den Standards und das Produkt zum Plan zurückzubringen, die Standards und den Plan zu ändern oder den Prozeß zu beenden.

Probleme

Mit der Kontrolle sind folgende Probleme verbunden:

- Viele Kontrollmethoden nehmen die bisher benötigte Zeit und die bisher verbrauchten Kosten als Maßstab für den Entwicklungsfortschritt. Die Beziehung zwischen Zeit, Kosten und Entwicklungsfortschritt ist im besten Fall grob, im schlechtesten Fall völlig unzutreffend.
- Standards für Entwicklungsaktivitäten sind entweder nicht vorhanden oder nicht schriftlich fixiert. Wenn sie vorhanden sind, dann wird ihre Durchsetzung oft nicht erzwungen. Firmenstandards werden zugunsten von lokalen ad hoc-Lösungen, die oft ungeeignet sind, umgangen.
- Eine Software-Meßtechnik, die Software-Maße (Metriken) über den Entwicklungsprozeß und das Produkt bereitstellt, ist noch nicht voll entwickelt.

Kapitel 6.2 Aufgaben

Um eine Entwicklung zu kontrollieren, sind vom Software-Management folgende Aufgaben durchzuführen:

1 Standards entwickeln und festlegen,
2 Kontroll- und Berichtssystem etablieren,
3 Prozesse und Produkte vermessen,
4 Korrigierende Aktionen initiieren,
5 Loben und Tadeln.

Standards

1 Standards entwickeln und festlegen

Sowohl für die Software-Entwicklung als auch für das Software-Produkt müssen Standards festgelegt werden. Standards können für eine gesamte Firma oder für jeweils ein Projekt verbindlich vorgeschrieben werden. Sie können selbst entwickelt werden oder von Organisationen oder anderen Firmen übernommen werden (z.B. das V-Modell). Folgende Teilaktivitäten sind durchzuführen:

- Entwickeln von Quantitäts- und Qualitätsstandards, Hauptkapit
- Festlegen des Prozeßmodells, Kapitel 3.3
- Festlegen von Qualitätssicherungsmethoden, Hauptkapit
- Entwickeln von Produktivitäts-, Qualitäts- und Prozeßmetriken. Kapitel 6.2

Das Etablieren von Standards stellt für das Software-Management eine Gratwanderung dar. Viele Software-Ingenieure sind noch der Meinung, daß jede Einschränkung ihrer Arbeitsweise ihre Kreativität negativ beeinflußt. Auf der anderen Seite muß man sich darüber im klaren sein, daß viele Mitarbeiter an einer Entwicklung arbeiten und daß

das Produkt von verschiedenen Personengruppen »in die Hand« genommen wird. Das Software-Management muß also auf der einen Seite Software-Bürokratie vermeiden und auf der anderen Seite »ungezügelten« Individualismus begrenzen.

Vorteile

Standards sollten immer daraufhin überprüft werden, ob sie folgende Vorteile bringen:

- ✚ Einarbeitungs- und Umschulungskosten sinken.
- ✚ Die Kommunikation zwischen Teammitarbeitern wird verbessert.
- ✚ Der Personalaustausch zwischen Projekten wird erleichtert.
- ✚ Erfahrungen können besser weitergegeben werden.
- ✚ Die besten Erfahrungen erfolgreicher Projekte können einheitlich angewandt werden.
- ✚ Wartung und Pflege werden vereinfacht.
- ✚ Standards können kontrolliert werden.

Kontroll- und Berichtssystem

2 Kontroll- und Berichtssystem etablieren

Kontroll- und Berichtssysteme müssen ausgesucht oder entwickelt werden, um den Entwicklungsprozeß zu überwachen und jederzeit den Entwicklungsstatus bestimmen zu können. Für Entwicklungsberichte ist der Typ, die Häufigkeit, der Ersteller und der Empfänger festzulegen. Die Art und der Umfang des verwendeten Kontroll- und Berichtssystems hängen von folgenden Parametern ab:

Kapitel 5.2
- Verwendeter Führungsstil *(Management-by...)*,

Hauptkapitel 3
- Verwendete Aufbau- und Ablauforganisation (Prozeßmodelle),
- Umfang der Entwicklung (Zeit, Anzahl, Mitarbeiter),

Hauptkapitel IV 2
- Eingesetzte CASE-Umgebung.

Tab. 6.1-1 listet einige typische Kontroll- und Berichtssysteme bzw. entsprechende Methoden auf. Berichte dienen dazu, dem Management Statusinformationen zur Verfügung zu stellen. Tab. 6.1-2 zeigt

Tab. 6.1-1: Kontroll- und Berichtssysteme

Methode	Definition oder Erläuterung
Kontrolle des Prozesses	
■ Budgetüberprüfungen	Vergleich des geschätzten Budgets mit den aktuellen Ausgaben, um Übereinstimmung oder Abweichung vom Plan festzustellen.
■ Meilensteinüberprüfungen	Überprüfung des Prozesses und des Produkts an den Meilensteinen, die in der Planung vorgesehen sind.
■ Verfolgung der *Top ten*-Risiken	Risiko-Überwachung durch Konzentration auf die jeweiligen 10 kritischen Risiken (siehe Abschnitt 5.5).
■ Qualitätssicherung	Geplante und systematische Überprüfung des Prozesses auf Einhaltung der vorgegebenen Prozeßstandards (siehe Kapitel 3.3).
Kontrolle des Produkts	
■ Konfigurationsmanagement	Methode zur Kontrolle eines Software-Status (siehe Kapitel 6.3).
■ Qualitätssicherung	Geplante und systematische Überprüfung des Produkts auf Einhaltung der vorgegebenen Qualitätsziele (siehe Hauptkapitel III 2)

Tab. 6.1-2: Berichtstyp

Berichtstyp	Definition oder Erläuterung
■ Budgetbericht	Vergleicht das Budget mit den Ausgaben und hilft neue Budgetschätzungen vorzunehmen.
■ Terminübersicht	Vergleicht den Terminstatus mit den fertiggestellten Meilensteinen.
■ Mitarbeiterstunden/ Aktivitäten-Bericht	Zeigt die Anzahl der Stunden, die benötigt wurden, um eine Aktivität durchzuführen.
■ Mitarbeitertage/ Aufgaben-Bericht	Zeigt die Anzahl der Tage, die für eine Aufgabe benötigt wurden.
■ Meilensteinfälligkeitsbericht	Gibt einen Status über die erreichten und überfälligen Meilensteine einschließlich der Gründe für die nicht erreichten Meilensteine.
■ Projektfortschrittsbericht	Ein nichtformalisierter Bericht über den Projektfortschritt oder eine Liste der erledigten Aktivitäten.
■ Aktivitätenbericht	Periodenbezogener Bericht, der angibt, welche Aktivitäten in der Periode erledigt wurden.
■ Trenddiagramm	Zeigt Trends in bestimmten Gebieten auf, wie Budgettrends, gefundene Fehler, Krankenstand usw. Trenddiagramme dienen dazu, die Zukunft vorherzusagen.
■ Änderungsbericht	Zeigt Ausnahmen vom Plan und signifikante positive und negative Veränderungen. Normalerweise zeigt er einen Entwicklungsrückstand an.
■ *Top ten*-Risikoelementliste	Periodischer Bericht, der pro Periode die 10 Risikoelemente mit den größten Risiken angibt (siehe Abschnitt 5.5).

Quellen: /Thayer 90, S.48/, /Boehm 91, S.39/

Beispiele für solche Berichte, die von Kontrollsystemen generiert werden können. Sie bieten die Möglichkeit, Überschreitungen vorgegebener Grenzen festzustellen. Außerdem erlauben sie es dem Manager, zukünftige Entwicklungen vorherzusagen. Der Software-Manager ist dafür verantwortlich, daß Projektdaten erfaßt werden, damit genauere Vorhersagetechniken entwickelt werden können.

Auf das Konfigurationsmanagement wird im Abschnitt 6.3 ausführlich eingegangen.

Strichliste

Zur Aufwandsermittlung pro Phase kann eine einfache Strichliste verwendet werden (Abb. 6.1-2), die ökonomisch ausgefüllt werden kann. Die Auswertung trägt wesentlich dazu bei, Aufwandsschätzungen für neue Projekte genauer vorzunehmen. Jeder Mitarbeiter markiert in der Strichliste durch einen Strich pro Stunde, welche Aktivität er in welcher Phase durchgeführt hat.

Hauptkapite

Hat man eine Projektplanung mit Hilfe eines Planungssystems vorgenommen, dann kann dieses Planungssystem auch für die Projektkontrolle verwendet werden.

Terminpläne können detailliert überwacht werden:

Abschnitt 6.

- Anfangs- und Endtermine sowie die Dauer von Vorgängen,
- Prozentsatz, zu dem Vorgänge bereits abgeschlossen sind,
- Kosten des Projekts, einzelner Vorgänge und Ressourcen,
- Arbeitsstunden, die von jeder Ressource ausgeführt wurden.

Abb. 6.1-2: Strichliste zur Erfassung des Aufwandes je Phase

Projektstatistik		Monat/Jahr:	
Projekt:		Mitarbeiter:	
Phase	Phasenplanung	Phasenrealisierung	Phasenüberprüfung
Planung			
Definition			
Entwurf			
Implementierung			
Abnahme & Einführung			
Wartung & Pflege			
Projektleitung			
Schulung, Einarbeitung			
Dienstreisen, Tagungen,			

Messen

3 Prozesse und Produkte vermessen

Aufgabe des Software-Managements ist es, geeignete Meß- und Überprüfungsverfahren zu entwickeln oder auszuwählen und einzuführen. Auf das Thema der Software-Metriken wird im nächsten Abschnitt ausführlich eingegangen.

Kapitel 6.2

Für die Qualitätsüberprüfung sowohl des Prozesses als auch des Produktes ist die Qualitätssicherung zuständig.

Hauptkapitel III 2

Sind entsprechende Meßverfahren etabliert, dann muß das Software-Management sicherstellen, daß die Messungen konsequent durchgeführt werden.

Korrigieren

4 Korrigierende Aktionen initiieren

Gibt es Abweichungen vom Plan oder von den Standards, dann muß der Software-Manager korrigierende Aktionen einleiten.

Beispielsweise kann der Plan oder ein Standard geändert werden, oder es werden Überstunden angeordnet, oder es werden andere Maßnahmen solange ergriffen, bis sich das Team wieder im Plan und im Standard befindet. Als letzte Maßnahme können noch die Anforderungen an das Produkt geändert werden, indem z.B. ein geringerer Funktionsumfang ausgeliefert wird.

Plan- und Standardänderungen können zu zusätzlichen Budgetanforderungen, zusätzlichen Mitarbeitern oder leistungsstärkeren Arbeitsplatzrechnern führen.

Hauptkapitel I 1

Manchmal ist es möglich, einen Teil des Plans einzuhalten, wenn die Zeit für die Einhaltung eines anderen Plans vergrößert wird. Dadurch erhöhen sich die benötigten Ressourcen. Manchmal ist es auch möglich, die geplanten Kosten einzuhalten, wenn der Endtermin nach hinten verschoben wird. Eine Änderung der Anforderungen kann bedeuten, daß die Software zunächst ohne vollständige Dokumentation ausgeliefert wird oder daß nicht alle Funktionen realisiert werden.

Lob & Tadel

5 Loben und Tadeln

Mitarbeiter, die den Plan und die Standards einhalten, sollten vom Manager gelobt werden, die anderen sollten getadelt werden. Lob sollte sich insbesondere durch nichtmonetäre Belohnungen ausdrükken, z.B. einen zusätzlichen freien Tag, eine Kongreßreise in die USA u.ä.

Die Aufgaben **1** und **2** sind in der Regel einmal für die gesamte Software-Entwicklung durchzuführen. Nur bei außergewöhnlichen oder besonders umfangreichen Projekten erscheint eine projektspezifische Festlegung sinnvoll. Die Aufgaben **3** bis **5** sind permanent für jede Software-Entwicklung durchzuführen.

DeMarco schreibt in seinem Buch *»Controlling Software Projects«* /DeMarco 82/: *»You can't control what you can't measure«*. Daher kommt einer Software-Meßtechnik eine große Bedeutung zu. Dieser Aspekt wird im nächsten Kapitel ausführlich behandelt.

Ein Software-Produkt ist kein monolithischer Block, sondern es besteht aus vielen verschiedenen Elementen, die z.T. unterschiedlichen Änderungszyklen unterworfen sind. Außerdem erhalten Kunden oft verschiedene Varianten eines Produktes. Um diese Vielfalt in den Griff zu bekommen, benötigt man ein Konfigurationsmanagement. Es wird in Kapitel 6.3 behandelt.

6.2 Metriken definieren, einführen und anwenden

Metrik

Eine Software-**Metrik** definiert, wie eine Kenngröße eines Software-Produkts oder eines Software-Prozesses gemessen wird.

Einige typische Kenngrößen eines Software-Prozesses und eines Software-Produktes zeigt Abb. 6.2-1. In der Literatur wird eine Vielzahl von Metriken beschrieben. Auf produktbezogene Metriken wird in den Hauptkapiteln III 5 und 6 eingegangen. Eine mathematische Definition der Begriffe Software-Metrik und Software-Maß wird in /Schmidt 84, S. 41f/ angegeben (siehe auch /Fenton 94/).

Ziele *»You can't control what you can't measure«*

Software-Metriken werden ermittelt, um dem Software- und Qualitätsmanagement quantitative Angaben über den Software-Entwicklungsprozeß und das Software-Produkt zur Verfügung zu stellen. Diese

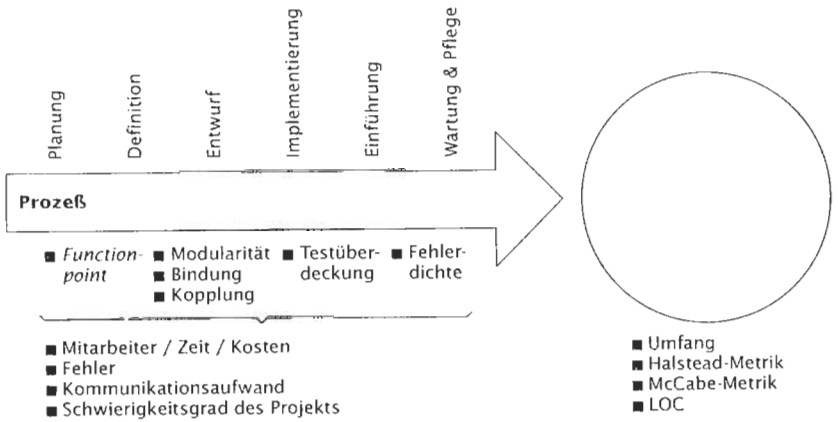


Abb. 6.2-1: Typische Kenngrößen von Software-Prozessen und Software-Produkten

Angaben sollen dazu beitragen, die Qualität und Produktivität des Erstellungsprozesses sowie die Qualität des Produktes zu kontrollieren. Außerdem sollen Vorhersagen und Planungen erleichtert bzw. verbessert werden.

Eine Vorgehensweise für das Messen wird in /Wallmüller 90, S. 28/ angegeben:

Meßtechnik

1 Definition der Meßziele,
2 Ableitung der Meßaufgaben aus den Meßzielen,
3 Bestimmung der Meßobjekte,
4 Festlegen der Meßgröße und Meßeinheit,
5 Zuordnung der Meßmethoden und Meßwerkzeuge zu den Meßobjekten und Meßgrößen,
6 Ermittlung der Meßwerte,
7 Interpretation der Meßwerte.

Beispiel 1

Als Beispiel für das Messen wird das Ermitteln von nichtkommentierten Quellanweisungen betrachtet.

Meßziel: Bestimmung der Anzahl der nichtkommentierten Quellanweisungen NCSS *(non-commented source statements)*.

Meßaufgabe: Zählen der Anzahl der nichtkommentierten Quellanweisungen eines Programms.

Meßobjekt: Auswahl eines zu vermessenden Programms.

Kenngröße: Anzahl Quellanweisungen einschließlich Compiler-Anweisungen, Datendeklarationen, ausführbaren Anweisungen, aber ohne Leerzeilen oder Zeilen, die vollständig aus Kommentaren bestehen. Jede Codezeile wird einmal gezählt. Jede enthaltene Datei wird einmal gezählt.

Meßeinheit: KNCSS (1000 NCSS).

Meßmethoden/
Meßwerkzeuge: Automatischer Zeilenzähler.

Interpretation: Repräsentiert den Umfang der produzierten Software.

Die Software-Metrik Umfang *(size)* ist bei der Firma HP /Grady, Caswell 87, S. 58/ so definiert.

Um von einem Software-Maß sprechen zu können, muß eine Software-Metrik bestimmte Gütekriterien erfüllen (Tab. 6.2-1). Gütekrit

Tab. 6.2 Gütekriterien für Software-Metriken

1	**Objektivität** (Intersubjektivität) Ein Maß ist objektiv, wenn keine subjektiven Einflüße des Messenden auf die Messung möglich sind.
2	**Zuverlässigkeit** (Meßgenauigkeit) Bei der Wiederholung der Messung unter denselben Meßbedingungen werden dieselben Meßergebnisse erzielt, d.h. das Maß ist stabil und präzise (zuverlässig).
3	**Validität** (Gültigkeit, Meßtauglichkeit) Die Meßergebnisse erlauben einen eindeutigen und unmittelbaren Rückschluß auf die Ausprägung der Kenngröße.
4	**Normierung** Es gibt eine Skala, auf der die Meßergebnisse eindeutig abgebildet werden. Gibt es eine Vergleichbarkeitsskala, dann ist ein Maß normiert.
5	**Vergleichbarkeit** Kann ein Maß mit anderen Maßen in eine Relation gesetzt werden, dann heißt es vergleichbar.
6	**Ökonomie** Die Messung muß mit geringen Kosten durchgeführt werden können. Die Ökonomie hängt vom Automatisierungsgrad, der Anzahl der Meßgrößen und der Anzahl der Berechnungsschritte ab.
7	**Nützlichkeit** Werden mit einer Messung praktische Bedürfnisse erfüllt, dann ist ein Maß nützlich.

Quellen: /Itzfeld 83/, /Itzfeld, Schmidt, Timm 84/; ähnliche und zusätzliche Kriterien sind in /Humphrey 89, S.308/, /Kearney et al. 86, S.1046 ff/ und /Baumann, Richter 92, S.625/ aufgeführt.

Das am schwierigsten nachzuweisende Kriterium ist die Validität, d.h. der Nachweis, daß das Maß tatsächlich das mißt, was es vorgibt zu messen. Maße ohne Validitätsaussagen sind für objektive Bewertungen unbrauchbar. Je nach Verwendungszweck werden an die Erfüllung der Gütekriterien unterschiedlich hohe Anforderungen gestellt. Für Ursache-Wirkungs-Analysen muß beispielsweise die Validität höher sein als für Zeitvergleiche. Für Ergiebigkeits-Analysen, z.B. zur Ermittlung von Produktivitätsfortschritten, werden sehr zuverlässige Metriken benötigt.

Ist eine Funktion von der realen Welt in eine formale numerische Welt vorhanden, dann gibt es auch eine Skala. Cromls, Raiffa und Thral (siehe /Zuse 85/) unterscheiden eine Hierarchie von aufeinander aufbauenden Skalen (Abb. 6.2-2). Die Skalen unterscheiden sich durch die zulässigen Transformationen, die auf ihnen durchgeführt werden können. Skalen

Abb. 6.2-2: Skalenhierarchie

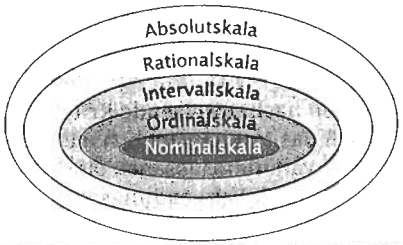
Nominalskala
Wird für qualitative Maße verwendet.
Operationen: Gleichheit oder Ungleichheit der Merkmalsausprägungen. Eindeutige Zugehörigkeit eines Objekts zu einer Klasse.
Beispiel: Matrikelnummern bei Studentenausweisen.

Ordinalskala
Operationen: Jede monoton steigende Funktion als Transformation erlaubt. Größer-, Kleiner- oder Gleichrelationen sind möglich. Median, Rang und Rangkorrelationskoeffizient können berechnet werden.
Beispiele: Schulnoten, Windstärken, Hubraumklassen von Autos.

Intervallskala
Operationen: Jede positiv lineare Funktion als Transformation zulässig. Rangordnung und Differenzen bleiben bei Transformationen invariant. Arithmetisches Mittel und Standardabweichung können berechnet werden.
Beispiel: Temperaturskala.

Rationalskala
Operationen: Jede Ähnlichkeitsfunktion ($f' = u \cdot f$, u reell, $u \geq 0$) als Transformation erlaubt. Es dürfen willkürliche Einheiten verwendet werden. Absoluter oder natürlicher Nullpunkt kann vorhanden sein. Quotienten können gebildet, Mittelwert und Varianz ausgerechnet werden.
Beispiele: Preise, Längen, Zeit, Volumen.

Absolutskala
Operationen: Jede Identitätsfunktion ($f' = f$) als Transformation zulässig. Da nur Identitätstransformationen erlaubt sind, bleibt alles invariant.
Beispiele: Häufigkeiten, Wahrscheinlichkeiten.



Quelle: /Zuse 85/

Klassifikation

Metriken lassen sich klassifizieren (Tab. 6.2-2).

Ziel der Datensammlung und -analyse ist es, eine zunehmende Anzahl von objektiven, absoluten, expliziten und dynamischen Metriken zur Kontrolle und Verbesserung der Entwicklung einzusetzen. Im Laufe der Zeit sollte die Vorhersagegenauigkeit, basierend auf diesen Metriken, graduell zunehmen bis sie sich nahe an der aktuellen Erfahrung befindet.

Die Einführung von Metriken sollte mit einer begrenzten Anzahl von expliziten, globalen Metriken beginnen.

Literaturhinweis

Wie jede Innovationseinführung erfordert auch die Einführung von Metriken viel psychologisches »Fingerspitzengefühl«.

Tab. 6.2-2: Klassifikat von Metrik

Objektiv/Subjektiv
Objektive Metriken sind leicht quantifizierbar und meßbar, z.B. Programmumfang, verbrauchte Zeit für eine Phase, Fehleranzahl usw.
Subjektive Metriken erfordern eine menschliche Einschätzung. Ein Beispiel für eine solche Metrik ist die Kundenzufriedenheit. Daten für subjektive Metriken werden oft durch Interviews oder statistische Erhebungen ermittelt. Daten können Antwortklassen zugeordnet werden, z.B. ausgezeichnet, gut, befriedigend, schlecht. Diese Klassen sollten durch Referenzpunkte auf einer Skala definiert werden. Referenzpunkte können aus leichtverständlichen Beispielen der Attribute bestehen.

Absolut/Relativ
Absolute Metriken sind invariant gegenüber der Hinzufügung neuer Elemente. Der Programmumfang eines Programms ist beispielsweise absolut und unabhängig vom Umfang anderer Programme. **Relative Metriken** ändern sich, z.B. der Durchschnitt oder die Steigung einer Kurve. Objektive Metriken sind oft absolut, während subjektive Metriken dazu tendieren, relativ zu sein.

Explizit/Abgeleitet (Primär/Sekundär) (Intern/Extern) (Basis/Berechnet)
Explizite Metriken, auch Basis-Metriken genannt, werden direkt ermittelt, während **abgeleitete Metriken**, auch berechnete Metriken genannt, von anderen expliziten oder abgeleiteten Metriken berechnet werden.
Ein Beispiel einer expliziten Metrik sind die entdeckten Fehler in einem Programm. Eine daraus abgeleitete Metrik ist die Anzahl der Fehler dividiert durch die Anzahl von tausend Quellzeilen.
Explizite Metriken bilden oft die Basis für erweiterte zusätzliche Metriken. Die Anzahl der Änderungswünsche der Kunden kann erweitert werden zu folgenden Metriken: Änderungswünsche bezogen auf Änderungsklassen und Änderungswünsche bezogen auf eine Zeitperiode.

Dynamisch/Statisch
Dynamische Metriken besitzen eine Zeitdimension, z.B. gefundene Fehler pro Monat. Die Werte dieser Metriken ändern sich in Abhängigkeit davon, wann die Messung im Lebenszyklus durchgeführt wurde. **Statische Metriken** bleiben invariant, z.B. die Anzahl der gefundenen Fehler während der gesamten Entwicklungszeit.

Vorhersagend/Erklärend
Vorhersagende Metriken können im voraus ermittelt oder generiert werden, während **erklärende Metriken** hinterher ermittelt werden.

Prozeßorientiert/Produktorientiert
Eine **prozeßorientierte Metrik** ist ein Attribut des Entwicklungs- und Pflegeprozesses, z.B. die Kosten der Entwicklung, oder der Entwicklungsumgebung, z.B. die durchschnittliche Programmiererfahrung der Mitarbeiter in Jahren.
Eine **produktorientierte Metrik** wird am Produkt gemessen. Sie sagt nichts darüber aus, wie das Produkt entstanden ist und warum das Produkt gerade in diesem aktuellen Zustand ist. Beispiele für Produktmetriken sind: Umfang des Produkts, Struktur- und Datenstrukturkomplexität.

Global/Speziell
Globale Metriken sind vor allem für Software-Manager von Interesse. Sie sind Indikatoren auf einem hohen Abstraktionsniveau und umfassen meist mehrere Phasen des Entwicklungsprozesses. Sie erlauben Einsichten in den Entwicklungsstatus, dargestellt durch Umfang, Produkt- und Prozeßqualität.
Spezielle Metriken sind Indikatoren für jeweils eine spezielle Phase im Entwicklungsprozeß.

Quellen: /Humphrey 89, S.308/, /Möller, Paulisch 93, S.57ff/

In /Grady, Caswell 87/ werden die Metrik-Auswahl und der Einführungsprozeß ausführlich beschrieben. Kapitel 5.5

Beispiel HP

Die Firma Hewlett-Packard hat schrittweise Metriken im gesamten Konzern eingeführt /Grady, Caswell 87/. Ursprünglich wurden sechs Metriken erhoben:

- Mitarbeiter/Zeit/Kosten *(people/time/cost)*,
- Umfang *(size)*,
- Fehler *(defects)*,
- Kommunikation *(communications)*,
- Schwierigkeit *(difficulty)*,
- Wartung *(maintainance)*.

Erfahrungen

Die Erfahrungen haben gezeigt, daß das Erfassungsformular für die Wartungsmetriken niemals komplett ausgefüllt wurde. Ein Grund dafür wird darin gesehen, daß die Verfolgung von Wartungsaktivitäten nicht mit derselben Begeisterung durchgeführt wird, wie das bei Entwicklungsaktivitäten der Fall ist. Außerdem ist es ohne ein hochentwickeltes Werkzeug nicht möglich, die hinzugefügten, geänderten und entfernten Zeilen zurückzuverfolgen.

Das ursprünglich vorgesehene Formular zur Erfassung der Projektkommunikation wurde im *Release 2* gestrichen, da es nur von wenigen Mitarbeitern ausgefüllt worden war.

Die Metrik »Schwierigkeit« soll etwas darüber aussagen, wie komplex die zu erstellende Software ist und wie stark die Randbedingungen des Projekts sind. Zur Ermittlung dieser Metrik wird ein

Tab. 6.2-3: HP-Erfassungsformular für die Metrik Mitarbeiter/Zeit/Kosten (Release 3)

Mitarbeiter/Zeit/Kosten

Projektname: Release-Nr. :

Aktivitäten	Lohnkostenaufwand in Ingenieurmonaten	Kalendermonate
Definition		
Entwurf		
Implementierung		
Test		
Summen		

% Überstunden (oder Minderarbeit) = _____ %

Gebrauchsanleitung

- Am Ende jeder Aktivität ist die entsprechende Zeile auszufüllen.
- Minderarbeit durch ein Minuszeichen kennzeichnen.
- Mit der Produktfreigabe ist das ausgefüllte Formular an die Metrikgruppe zu senden.

Definitionen

- Lohnkostenaufwand in Ingenieurmonaten
  Summe der Kalenderlohnkostenmonate, die jedem Projektingenieur zugeordnet wurden, einschl. der Mitarbeiter, die Tests durchführen. Längere Urlaube und Abwesenheiten werden nicht berücksichtigt. Zeiten, die Projektmanager für Managementaufgaben benötigt haben, werden nicht eingetragen.
- Überstunden/Minderarbeit
  Ingenieurzeit, die über/unter der 40 Stunden-Ingenieurwoche im Durchschnitt des Projekts lag. %-Über-/Unterzeit kann als Normalisierungsfaktor für den Lohnkostenmonat verwendet werden.
- Kalendermonate
  Die vergangene Zeit in Kalendermonaten zwischen speziellen Projekt-Kontrollpunkten.

Quelle: /Grady, Caswell 87, S.53f/, übersetzt und modifiziert.

13seitiger Fragebogen ausgefüllt. Es wird nach der Stabilität der Anforderungen, der Erfahrung der Mitarbeiter, der Vertrautheit mit dem Typ der Software und der Entwicklungsumgebung, dem Zugriff auf die benötigte Hardware und vielen anderen, allgemeinen Projektspezifika gefragt.

In *Release 3* wurden die Formulare, die ausgefüllt werden müssen (4 Seiten), und die Formulare, die optional sind, in zwei getrennte Pakete aufgeteilt.

Zwingend ausgefüllt werden müssen ein allgemeines Projekt-/Produktformular sowie die Formulare für die Metriken Mitarbeiter/Zeit/Kosten, Umfang und Fehler. Die Erfassungsformulare und Definitionen für diese drei Metriken sind in den Tabellen 6.2-3 bis 6.2-5 in übersetzter Form aufgeführt. Sie zeigen, wie man auf pragmatische Art Metriken formulieren und einführen kann.

Beispiele für Auswertungen dieser Metriken zeigen die Abb. 1.3-1, 1.4-4 und 1.4-5 im Hauptkapitel 1.

Hauptka

Die grundsätzliche Problematik von Metriken liegt darin begründet, daß man das, was einen interessiert und man eigentlich messen möchte, nicht direkt messen kann. Beispielsweise will man die Qualität eines Produktes wissen.

Zur Prob von Met

Tab. 6.2 HP-Erfas formula Metrik F (Release

Fehler vor der Auslieferung *(pre-release defects)*

Projektname: Release-Nr. :

Aktivitäten	Fehler eingebracht (optional)	Fehler gefunden	Fehlerbehebung abgeschlossen
Definition			
Entwurf			
Implementierung			
Test			
Summen			

Gebrauchsanleitung

- Am Ende jeder Aktivität sind die gefundenen Fehler und abgeschlossenen Fehlerbehebungen einzutragen. Die eingebrachten Fehler sind zu aktualisieren.
- Wurden Fehler während einer Aktivität nicht gezählt, dann ist nichts einzutragen; keine Null eintragen.
- Mit der Produktfreigabe ist das ausgefüllte Formular an die Metrikgruppe zu senden.

Definitionen

- Fehler:
  Ein Fehler ist eine Abweichung von der Produktspezifikation oder ein Fehler in der Spezifikation, wenn der Fehler nicht entdeckt und korrigiert wurde.
  Konnte der Fehler nicht entdeckt werden oder wurde er entdeckt aber nicht behoben, dann handelt es sich um eine Erweiterung und nicht um einen Fehler.
  Fehler schließen typographische oder grammatikalische Fehler in der Dokumentation *nicht* ein.
- Fehler eingebracht:
  Anzahl der Fehler, die dem Ergebnis einer Aktivität zugeordnet und nicht vor der letzten Aktivität gefunden werden konnten.
- Fehler gefunden:
  Anzahl der Fehler, die in einer Aktivität gefunden wurden.
- Fehlerbehebung abgeschlossen:
  Anzahl der Fehler, die in einer Aktivität korrigiert wurden.

Quelle: /Grady, Caswell 87, S.55f/, übersetzt und modifiziert.

Tab. 6.2-5: HP-Erfassungsformular für die Metrik Umfang (Release 3)

Ausgelieferter Umfang

Projektname: Release-Nr. :

Sprache A: Sprache B:

Zeilenzähler-Werkzeug (oder andere Technik):

	Sprache A:	Sprache B:
NCSS		
Kommentarzeilen		
Leerzeilen		
% wiederverwendeter Code		
# Prozeduren		
Bytes Objektcode		
# Zeilen Ingenieurdokumentation		
# Abbildungen in der Ingenieurdokumentation		

Gebrauchsanleitung

- Benutzen Sie einen automatischen Zeilenzähler. Ist kein Werkzeug verfügbar, dann schätzen Sie NCSS, Kommentarzeilen und Leerzeilen (Zuverlässigkeitsgrad der Schätzung = _____%).
- Senden Sie das ausgefüllte Formular an die Metrik-Gruppe.

Definitionen

- Ausgelieferter Umfang
  Die Codezeilen, die zum Produkt gehören, das an den Kunden ausgeliefert wird.
- NCSS
  siehe Anfang dieses Buchabschnitts, Beispiel 1.
- Kommentarzeilen
  Zeilen die nur Kommentar enthalten. Eine kommentierte ausführbare Zeile wird als ausführbarer Code gezählt, nicht als Kommentar. Leerzeilen werden nicht als Kommentarzeilen gezählt.
- Ingenieurdokumentation
  Dokumentation, die nicht im Quellcode oder in der Endbenutzerdokumentation enthalten ist.
  Wenn die Zeilen geschätzt werden, dann ist von 54 Zeichen pro Zeile auszugehen.
- Wiederverwerteter Code
  Code, der in dieses Produkt eingegliedert wurde, und der vorher in einem anderen Produkt oder einem anderen Teil dieses Produktes einwandfrei arbeitete.

Quelle: /Grady, Caswell 87, S.57f./, übersetzt und modifiziert.

Literatur: /Baumann, Richter 92/

Um dieses Problem zu lösen, stützt man sich auf Hypothesen. In einer Formel faßt man die quantitative Beziehung zwischen meßbaren und interessierenden Größen zusammen. Da jedoch der Software-Entwicklungsprozeß noch nicht vollständig verstanden ist, fehlt quantitativen Aussagen eine sichere Basis.

Software-Metriken sind daher mit der notwendigen Vorsicht und Skepsis zu betrachten. Sie liefern bestenfalls relative Aussagen und weisen in der Regel auf Anomalien hin, die sowohl positiv als auch negativ sein können.

Beispiel

Mit Hilfe der *Function Point*-Methode versucht man den Aufwand einer Software-Entwicklung (abhängige Variable) in Abhängigkeit von Art und Umfang der Produktanforderungen sowie vom Schwierigkeitsgrad des Produkts (unabhängige Variablen) zu ermitteln: $y = f(x_1, \ldots x_n)$.

Kapitel I 1.5

Aufwand = f (Art, Umfang, Schwierigkeitsgrad)

Der grundsätzliche Zusammenhang zwischen den Kenngrößen Art,

Umfang und Schwierigkeitsgrad und dem Aufwand wurde vom Erfinder der *Function Point*-Methode postuliert. Die konkrete Formel wurde dann mittels statistischer Analyse ermittelt. Im Fall der *Function Point*-Methode werden Art und Umfang aus den verbalen Produktanforderungen ermittelt und der Schwierigkeitsgrad anhand eines Kriterienrasters geschätzt (Abb. 6.2-3).

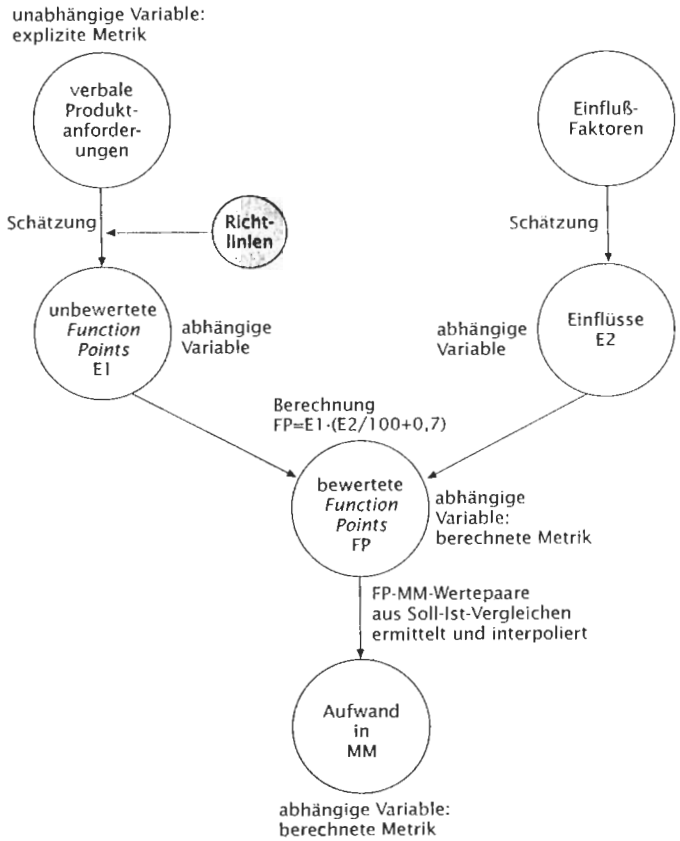


Abb. 6.2-3: Ableitung der Mitarbeitermo (MM) bei der Function Point Methode

Das Beispiel zeigt deutlich, daß die Aussagekraft einer Software-Metrik von der Güte des zugrundeliegenden Modells und der entsprechenden Formel abhängig ist. Eine Metrik ist nur dann valide, wenn die Beziehungen zwischen den direkt gemessenen Größen und den gesuchten Kenngrößen gut verstanden sind.

6.3 Konfigurationsmanagement etablieren

Zur Historie: Das Konfigurationsmanagement (KM) wurde ursprünglich von der amerikanischen Raumfahrtindustrie in den fünfziger Jahren eingeführt. Raumfahrzeuge unterlagen damals während ihrer Entwicklung zahlreichen undokumentierten Änderungen. Außerdem wurden die Raumfahrzeuge im Test normalerweise zerstört. Nach einem erfolgreichen Test waren die Hersteller nicht in der Lage, eine Serienfertigung aufzubauen oder auch nur einen Nachbau durchzuführen. Die Pläne waren veraltet und der Prototyp mit allen Änderungen vernichtet. KM wurde erfunden, um diesen Informationsverlust zu verhindern.

Bei jeder Software-Entwicklung stellen sich – in mehr oder weniger großem Umfang – folgende Probleme ein:

- Häufige Änderungen an Software-Elementen verursachen ein Chaos. Bereits korrigierte Fehler tauchen wieder auf. Es ist unklar, warum und von wem welche Änderungen durchgeführt wurden.
  Software-Konfigurationsmanagement kann diese Probleme durch Aufzeichnung der Historie reduzieren. Alle Änderungen einschließlich des Grunds, des Zeitpunktes und der Verantwortlichen werden festgehalten.
- Es ist unklar, ob ein Fehler bereits behoben wurde oder nicht. Was in der neuen Freigabe geändert wurde, ist unbekannt.
  Software-Konfigurationsmangement verknüpft Änderungswünsche und vorgenommene Änderungen und überwacht den Änderungsprozeß.
- Es ist schwierig, das System so zu konfigurieren, daß alle Fehlermeldungen bis vor zwei Wochen berücksichtigt sind. Die letzten Verbesserungen waren fehlerhaft, sie müssen aus der Konfiguration entfernt werden.
  Software-Konfigurationsmangement erlaubt eine automatische Versions- und Konfigurationsselektion und hilft bei der Zusammenstellung von konsistenten Konfigurationen mit gewünschten Eigenschaften.
- Man ist unsicher, ob alles neu übersetzt wurde und ob die Kunden die neueste Freigabe haben.
  Software-Konfigurationsmangement sorgt dafür, daß kein Arbeitsschritt bei der Vor- oder Nachbearbeitung von Software-Elementen mit Werkzeugen vergessen wird.

Diese Problembereiche zeigen deutlich die Notwendigkeit eines Software-Konfigurationsmanagements. Im folgenden werden zunächst Konfigurationen, Versionen und Varianten betrachtet, bevor auf das Konfigurations- und Änderungsmanagement eingegangen wird.

6.3.1 Konfigurationen

Ein Software-Produkt ist kein einheitliches Gebilde, sondern besteht aus einer Vielzahl unterschiedlicher Software-Elemente, z.B. Pflichtenheft, Produktmodell, Entwurfsdokumentation, Modul 1 bis n, Testfälle 1 bis n, Benutzerhandbuch, Projektpläne usw. Damit man weiß, welche Software-Elemente zu einem Software-Produkt gehören, beschreibt man die Zusammengehörigkeit durch eine Konfiguration.

Konfiguration

Eine **Software-Konfiguration** ist eine benannte und formal freigegebene Menge von Software-Elementen, mit den jeweils gültigen Versionsangaben, die zu einem bestimmten Zeitpunkt im Produktlebenszyklus in ihrer Wirkungsweise und ihren Schnittstellen aufeinander abgestimmt sind und gemeinsam eine vorgesehene Aufgabe erfüllen sollen.

Ein **Software-Element** ist jeder identifizierbare und maschinenlesbare Bestandteil des entstehenden Produkts oder der entstehenden Produktlinie.

Element

Jedes Software-Element muß einen eindeutigen Bezeichner besitzen, der kein zweites Mal vergeben werden darf. Jede Änderung erzeugt ein neues Element mit einem neuen Bezeichner.

Software-Elemente lassen sich nach ihrer Entstehungsart klassifizieren:

- Quellelement:
  Software-Element, das durch manuelle Eingaben erzeugt wird, z.B. unter Zuhilfenahme eines Editors.
- Abgeleitetes Element:
  Software-Element, das vollautomatisch durch ein Programm erzeugt wird, z.B. Objektcode.

Entstehu

Quellelemente erfordern menschliche Arbeitskraft und sind bei Verlust oft nicht exakt reproduzierbar. Sie dürfen daher nur unter eingeschränkten Bedingungen gelöscht werden. Abgeleitete Elemente können dagegen bei Platzproblemen gelöscht werden, wenn ihre Ausgangselemente und ihre Erzeugerprogramme noch vorhanden sind.

Software-Elemente lassen sich außerdem nach ihrer Struktur klassifizieren:

- Atom:
  Software-Element, das für eine Software-Konfiguration eine unteilbare Einheit bildet. Ein Atom enthält keine Unterheiten, die unabhängig voneinander variieren.
- Konfigurationen:
  Software-Element, das aus mehreren anderen Software-Elementen zusammengesetzt ist, die unabhängig voneinander variieren können.

Struktur

In einem **Konfigurations-Identifikationsdokument** (KID) wird für eine Konfiguration aufgeführt, welche Software-Elemente zu ihr gehören. Ein solches Dokument wird auch als **Konfigurationshierarchie** oder **Elementstrukturplan** bezeichnet. Ähnlich wie bei einer Stückliste im Hardwarebereich werden alle Elemente aufgelistet, die zu einem Produkt gehören.

ID

In der Regel werden auch solche Software-Elemente aufgeführt, die als Hilfsmittel und Werkzeuge zur Erstellung verwendet wurden, aber nicht an den Auftraggeber bzw. Käufer ausgeliefert werden. Beispielsweise muß vermerkt werden, welche Compilerversionen zum Übersetzen der Quellprogramme verwendet wurden. Diese Compilerversionen müssen außerdem archiviert werden.

einschließli Werkzeuge

Beispiel 1a Anhang I A

Ein Konfigurations-Identifikationsdokument für die ausgelieferte Konfiguration des Produkts Seminarorganisation kann folgendermaßen aussehen (auf die Versionszählung wird anschließend näher eingegangen):

KID Seminarorganisation
Typ der Konfiguration: Produktkonfiguration
Versionsnummer der Konfiguration: V 1.0
Zustand der Konfiguration: akzeptiert
Datum der letzten Konfigurationsänderung: 3/5/97
Produktbestandteile:
a Pflichtenheft SemOrgV3.7 (Datei: SemOrg/Def/PfV37)
b OOA-Modell SemOrgV2.5 (Datei: SemOrg/Def/OOAV25)
c Benutzungsoberfläche SemOrgV1.7 (Datei: SemOrg/Def/GUIV17)
d Benutzerhandbuch SemOrgV2.4 (Datei: SemOrg/Def/BHV24)
e OOD-Modell SemOrgV1.3 (Datei: SemOrg/Ent/OODV13)
f Klasse Kunde SemOrgV1.8 (Datei: SemOrg/Imp/KundeV18.cpp)
(Datei: SemOrg/Imp/KundeV18.ob)
g Klasse Buchung SemOrgV.2.1 ...
...
n Ausführbares Programm SemOrgV3.1
(Datei: SemOrg/Imp/SemV31.exe)
o Testfälle SemOrgV5.2 (Datei: SemOrg/Imp/TestV52.txt)
p Projektplan SemOrgV3.2 (Datei: SemOrg/Pro/PlanV32)
Werkzeugbestandteile:
I Textsystem Word V7.0 (für die Produkte a, d und o)
(Datei: SemOrg/Tools/WordV70/...)
II OO-Case-Tool OTool V2.3 (für die Produkte b und e)
(Datei: SemOrg/Tools/OToolV23/...)
III GUI-Tool GUI-Builder V1.7 (für das Produkt c)
(Datei: SemOrg/Tools/GUIBV17/...)
IV C++-Compiler V3.2 (für die Produkte f bis m)
(Datei: SemOrg/Tools/CPPV32/...)
V Projektplaner Project V2.5 (für Produkt p)
(Datei: SemOrg/Tools/ProV25/...)
Ausgelieferte Teile:
An den Kunden werden die Teile d und n ausgeliefert.

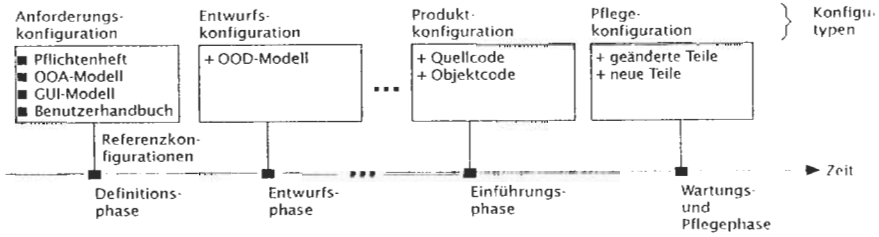
Nach der Auslieferung wird ein Produkt gewartet und gepflegt. Jede Änderung, die sich daraus ergibt, führt zu einer neuen Konfiguration, die in einem neuen Konfigurations-Identifikationsdokument festgehalten wird. Ist beispielsweise eine neue Konfiguration fehlerhaft, dann kann man sich auf die vorherige Konfiguration zurückziehen.

Um solche Möglichkeiten auch während des Software-Entwicklungsprozesses zu haben, werden definierte und freigegebene Zwischenergebnisse ebenfalls zu Konfigurationen zusammengefaßt und entsprechend dokumentiert. Solche Konfigurationen bezeichnet man im Englischen als *baselines* (Grundlinie, Standlinie), im folgenden wird der Begriff Referenzkonfiguration dafür verwendet.

Referenzkonfiguration

Eine **Referenzkonfiguration *(baseline)*** ist ein zu einem bestimmten Zeitpunkt im Entwicklungsprozeß ausgewähltes, gesichertes und freigegebenes Zwischenergebnis. Abb. 6.3-1 veranschaulicht dieses Konzept.

Abb. 6.3-1: Konfigurationstypen und Referenzkonfiguration



Beispiel 1b

Die Anforderungskonfiguration für das Produkt Seminarorganisation kann folgendermaßen aussehen:

KID Seminarorganisation
Typ der Konfiguration: Anforderungskonfiguration
Versionsnummer der Konfiguration: V1.0
Zustand der Konfiguration: akzeptiert
Datum der letzten Konfigurationsänderung: 10/1/97
Produktbestandteile:
a Pflichtenheft SemOrgV2.3 (Datei: SemOrg/Def/PfV23)
b OOA-Modell SemOrgV2.1 (Datei: SemOrg/Def/OOAV21)
c Benutzungsoberfläche SemOrgV1.2 (Datei: SemOrg/Def/GUIV12)
d Benutzerhandbuch SemOrgV1.3 (Datei: SemOrg/Def/BHV13)
e Projektplan SemOrgV1.7 (Datei: SemOrg/Pro/PlanV17)
Werkzeugbestandteile:
I Textsystem Word V7.0 (für die Produkte a und d)
(Datei: SemOrg/Tools/WordV70/...)
II OO-Case-Tool OTool V2.3 (für das Produkt b)
(Datei: SemOrg/Tools/OToolV23/...)
III GUI-Tool GUI-Builder V1.7 (für das Produkt c)
(Datei: SemOrg/Tools/GUIBV17/...)
IV Projektplaner Project V2.5 (für Produkt p)
(Datei: SemOrg/Tools/ProV25/...)

Neben der Anzahl der Software-Elemente unterscheiden sich die Konfigurationen in Beispiel 1a und 1b im wesentlichen durch die Versionsnummer. Versionsnummern spielen daher eine wichtige Rolle bei der Konfigurationsverwaltung. Es ist erforderlich, ein Verfahren für die Vergabe von Versionsnummern zu definieren.

6.3.2 Versionen und ihre Verwaltung

Version

Eine **Version** kennzeichnet die Ausprägung eines Software-Elements zu einem bestimmten Zeitpunkt. Unter Versionen werden zeitlich nacheinander liegende Ausprägungen eines Software-Elements verstanden. Eine Version wird in der Regel durch eine Nummer beschrieben.

Die Versionsnummer besteht im allgemeinen aus zwei Teilen:

Release = Freigabe
- der *Release*-Nummer und

Level = Niveau
- der *Level*-Nummer.

Die ***Release***-Nummer (im allgemeinen einstellig) steht, getrennt durch einen Punkt, vor der *Level*-Nummer (ein- bis zweistellig), z.B. 1.1, 2.15. Ein neues Software-Element erhält die Versionsnummer 1.0. Bei jeder kleineren, formalen oder inhaltlichen Änderung an dem Software-Element wird die *Level*-Nummer um 1 erhöht. Bei jeder größeren oder gravierenden Änderung an dem Software-Element wird die *Release*-Nummer um 1 erhöht und gleichzeitig die *Level*-Nummer auf 0 gesetzt (Abb. 6.3-2).

Beispiel 2a

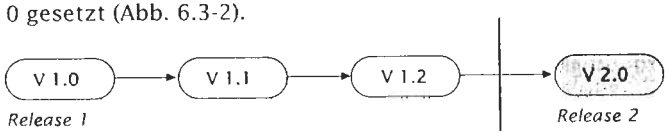


Abb. 6.3-2: Versionszählung

Checkin/Checkout-Modell

Das verbreitetste Modell zur Versionsverwaltung ist das ***Checkin/Checkout*-Modell**. Software-Elemente werden in diesem Modell in Archiven gesammelt.

Eine *Checkout*- oder Ausbuche-Operation holt eine Kopie eines Software-Elements aus dem Archiv und reserviert es für den Ausbucher. Die Kopie darf geändert werden.

Wird versucht, das reservierte Element nochmals auszubuchen, dann gibt es entweder eine Fehlermeldung oder es wird ein paralleler Entwicklungsast abgespalten (siehe nächsten Abschnitt). Dieser Reservierungsmechanismus sorgt aber dafür, daß sich Änderungen nicht gegenseitig überschreiben und keine unbeabsichtigte Parallelentwicklung stattfindet.

Nach der Überarbeitung befördert die *Checkin*- oder Einbuche-Operation das neue Software-Element in das Archiv und löscht die Reservierung. Zusätzlich erledigt diese Operation die Geschichtsschreibung: Sie vermerkt den Autor des Elements, den Einbuchungszeitpunkt, einen Logbucheintrag, der die Änderung zusammenfaßt, sowie weitere Informationen. Ein eingebuchtes Element ist »eingefroren«, d.h. es kann nicht mehr geändert werden. Jede Überarbeitung erfordert einen *Checkin/Checkout*-Zyklus und erzeugt ein neues Element.

Für unterschiedliche Elemente können im Archiv gleichzeitig mehrere Reservierungen bestehen. Ein Lesezugriff wird durch die Reservierung nicht verhindert.

Das *Checkin/Checkout*-Modell wird meist durch die Deltatechnik realisiert. Anstelle kompletter Kopien werden nur die Unterschiede (Deltas) von zeitlich aufeinanderfolgenden Elementen gespeichert. Der Benutzer merkt von der Deltatechnik nichts.

Realisierung

6.3.3 Varianten

Verschiedene Versionen eines Software-Elements führen zu einem sequentiellen Versions-Stamm. Für die Praxis reichen solche Versions-Stämme aber nicht aus. Durch Varianten werden zusätzliche Anforderungen abgedeckt. Der Variantenbegriff ist heute nicht einheitlich definiert /Mahler 94/.

Varianten können

Varianten

1. zeitgleich nebeneinander liegende Ausprägungen von Software-Elementen sein (V-Modell),
2. verwendet werden, um parallele Entwicklungslinien darzustellen,
3. unterschiedliche, d.h. variante, Implementierungen derselben Schnittstelle sein, d.h. die Funktionalität bleibt gleich,
4. sich durch bedingte Übersetzungen *(conditional compilation)* unterscheiden,
5. auf unterschiedliche Hardware- und/oder Systemsoftware-Konstellationen zugeschnitten sein,
6. ab einem bestimmten Abstraktionsniveau *nicht* mehr unterschieden werden.

Abschnitt 3.3.2

Beispiel 2b

Ein Produkt befindet sich in zwei Versionen bei Kunden (V1.0 und V1.1). Die Entwicklung arbeitet an Version 1.2. Der Kunde mit der Version 1.0 findet einen gravierenden Fehler. Da er aufgrund seiner Hardwarevoraussetzungen nicht auf Version 1.1 wechseln kann, wird der Fehler behoben und es entsteht die Variante V1.0.1.0. (Abb. 6.3-3). Da sich diese Variante weiterhin gut verkauft, wird beschlossen, sie funktionell zu erweitern und als *Light*-Produkt zu verkaufen (V1.0.1.1).

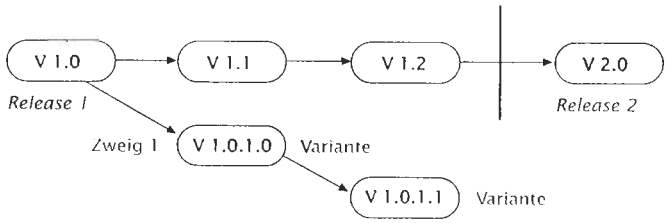


Abb. 6.3-3: Versi mit einem Zweig

Varianten, die aus Verzweigungen von Versionen bestehen, kann man folgendermaßen aufbauen:

Variantennummer = release.level.branch.sequence

Beispiel 2c Der Kunde mit der Variante V1.0.1.0 benötigt eine spezielle Funktion; er erhält die Variante V1.0.2.0 und nach einer Fehlerbehebung die Variante V1.0.2.1 (Abb. 6.3-4). Die Version 2.0 erweist sich als so fehlerhaft, daß die Version 2.1 aus der stabilen Version 1.1 abgeleitet wird.

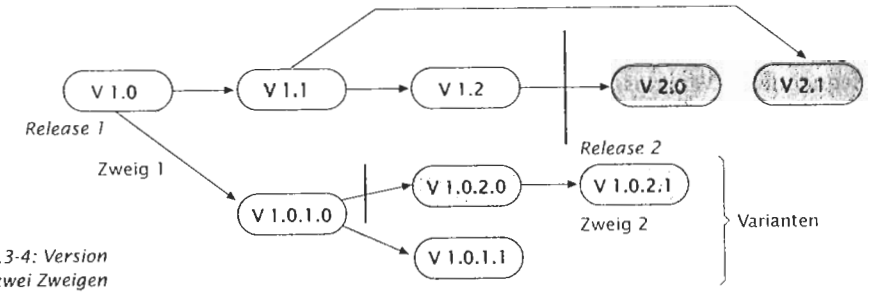


Abb. 6.3-4: Version mit zwei Zweigen

Dieses Beispiel zeigt, daß man nicht nur ein Schema für die Versionssicherung benötigt, sondern u.U. auch die Vorgängerversionsnummer angeben muß, wenn man von der sequentiellen Folge abweicht.

Oft ist es notwendig, ein Software-Element auf unterschiedliche Hardware- und Systemsoftwarekonstellationen zuzuschneiden.

Beispiel Soll ein Software-Werkzeug auf den Computerplattformen Intel und SUN für die Betriebssysteme Windows und Solaris entwickelt werden, dann sind folgende Varianten sinnvoll: Variante a (Intel, Windows), Variante b (Intel, Solaris), Variante c (SUN, Solaris).

Identifikation Damit eine Software-Konfiguration gebildet werden kann, muß ein Software-Element eindeutig identifizierbar sein. Es ist daher notwendig, ein Numerierungsschema für Software-Elemente einzuführen. Ein Numerierungsschema muß folgendes festlegen:

- Struktur bzw. Aufbau des Identifikators,
- Informationen, die der Identifikator enthalten soll,
- Verfahren, wie sich der Identifikator bei Änderungen des Elements (z.B. neue Version, neue Variante) verhält.

Identifikator Als Identifikator kann beispielsweise ein hierarchisches Namensschema verwendet werden, das mit der Versions- und Variantenkennung kombiniert ist. Der Identifikator kann auch durch das verwendete Konfigurationsmanagement-Werkzeug bestimmt werden.

Beispiel SemOrg/Def/PfV23a.doc
SemOrg/Ent/OODV13a
SemOrg/Imp/KundeV18c.cpp
Der erste Teil des Identifikators gibt an, daß es sich um das Dokument Pflichtenheft in der Version 2.3 in der Variante a handelt. Das

Pflichtenheft gehört zur Definition des Produktes Seminarorganisation.
Dieser Identifikator gibt gleichzeitig die formale Beziehung zu anderen Elementen an (Abb. 6.3-5).

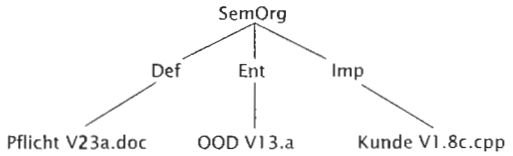


Abb. 6.3 Dokumen hierarchi

Die Blätter der Hierarchie können identisch sein mit den entsprechenden Dateinamen.

Konfigurationen erfüllen also folgende Zwecke:

- Sie legen fest, welche Entwicklungsergebnisse (Software-Elemente) zu welchen Produkten oder Teilprodukten gehören.
- Sie bringen Ordnung in die Vielfalt der Entwicklungs-, Wartungs- und Pflegeergebnisse.
- Sie bilden Bezugspunkte für definierte Entwicklungs-, Wartungs- und Pflegeschritte, da sie die Ergebnisse vorangegangener Schritte festhalten und damit eine wohldefinierte Basis darstellen.
- Sie frieren Zwischenergebnisse eines Produktes über seine gesamte Lebensdauer ein und erleichtern dadurch die Wartbarkeit und Wiederverwendbarkeit des Produktes.
- Sie ermöglichen eine Analyse des Entwicklungsprozesses und des Produktes.

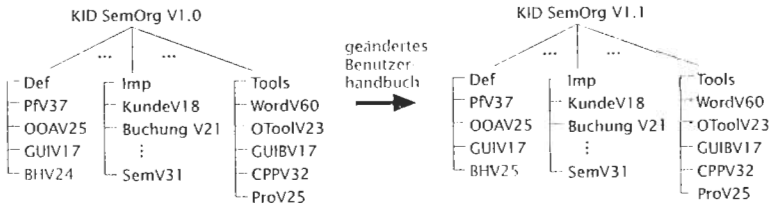
Konfigurationen und Varianten können analog wie Versionen mit dem *Checkin/Checkout*-Modell verwaltet werden. Realisier

6.3.4 Konfigurations- und Änderungsmanagement

Änderung Wird ein Software-Element, das zu einer Konfiguration gehört, geändert, dann ändert sich auch die gesamte Konfiguration. Daher müssen diese Änderungen einen formalen Änderungsprozeß durchlaufen, der in eine neue Konfiguration mündet. Die Informationen über die Konfigurationen müssen in einer geeigneten Datenbasis verwaltet werden.

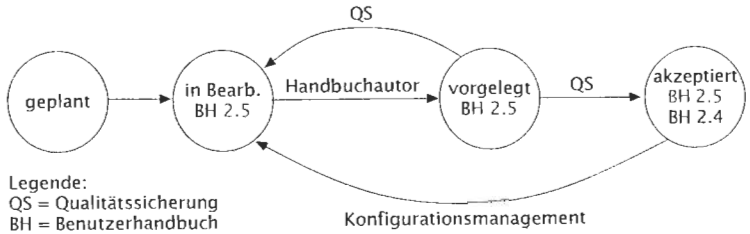
Beispiel 1 Die Abb. 6.3-6 zeigt, daß Änderungen am Benutzerhandbuch zu einer neuen Konfiguration führen. Alle zu einer Konfiguration gehörenden Software-Elemente befinden sich im Zustand »akzeptiert« (Abb. 6.3-7):
Werden Änderungen an einem akzeptierten Software-Element vorgenommen, dann wird der Zustand des Software-Elements von »akzeptiert« in den Zustand »in Bearbeitung« zurückversetzt. Die Versions-

Abb. 6.3-6: Entstehung einer neuen Konfiguration



nummer wird um Eins erhöht, bei gravierenden Änderungen wird die *Release*-Nummer erhöht (Abb. 6.3-7).

Abb. 6.3-7: Zustandsübergänge des Benutzerhandbuchs



Damit das Benutzerhandbuch wieder in eine neue Konfiguration eingebracht werden kann, muß es nach der Bearbeitung durch den Handbuchautor der Qualitätssicherung vorgelegt werden. Gibt die Qualitätssicherung das geänderte Handbuch frei, dann erhält die neue Handbuchversion V2.5 den Zustand »akzeptiert«. Damit kann dieses Software-Element in eine neue Konfiguration eingebettet werden.
Einen ähnlichen Zyklus durchläuft die neue Konfiguration. Das Konfigurations-Identifikationsdokument »KID SemOrg V1.0« wird in den Bearbeitungszustand versetzt und die Versionsnummer wird erhöht. Physikalisch wird eine Kopie von dem alten Dokument erstellt und mit der neuen Versionsnummer versehen. Der Bearbeiter der neuen Konfiguration wartet, bis das neue Benutzerhandbuch akzeptiert ist, ändert dann das Konfigurations-Identifikationsdokument und legt es der Qualitätssicherung vor. Nach Freigabe durch diese liegt die neue akzeptierte Konfiguration SemOrg V1.1 vor.

Das Beispiel 1 c zeigt, daß eine Vielzahl von Schritten erforderlich ist, um eine Konfiguration fortzuschreiben. Diese Aktivitäten werden vom Konfigurationsmanagement durchgeführt.

Konfigurationsmanagement

Das **Software-Konfigurationsmanagement** (SKM) unterstützt die Identifikation, Initialisierung und effiziente Verwaltung von Software-Konfigurationen durch geeignete Methoden, Werkzeuge und Hilfsmittel, um durch kontrollierte Änderungen die Entwicklung und Pflege eines Software-Produkts zu erleichtern und transparent zu machen.

Ziele

Die Ziele des Software-Konfigurationsmanagements sind:

- Sicherstellung der Sichtbarkeit, Verfolgbarkeit und Kontrollierbarkeit eines Produkts und seiner Teile im Lebenszyklus.
- Überwachung der Konfigurationen während des Lebenszyklus, so daß die Zusammenhänge und Unterschiede zwischen früheren Konfigurationen und den aktuellen Konfigurationen jederzeit erkennbar sind.
- Sicherstellung, daß jederzeit auf vorangegangene Versionen zurückgegriffen werden kann, damit Änderungen nachvollziehbar und überprüfbar sind.

Aufgaben

Um die Ziele zu erreichen, müssen durch das Konfigurationsmanagement folgende Aufgaben erledigt werden (hier dargestellt in Anlehnung an das V-Modell /V-Modell 97, S. 6-1ff/, siehe dazu auch /Bersoff 84/):

1 Planung des Konfigurationsmanagements
Der organisatorische und verfahrenstechnische Rahmen des Konfigurationsmanagements wird in einem Konfigurationsmanagement-Plan festgelegt. Außerdem ist eine Produktbibliothek mit zugehörigen Werkzeugen bereitzustellen. In der Produktbibliothek werden alle Produkte, soweit sinnvoll und möglich, zusammengefaßt und verwaltet. Sie ist Teil der CASE-Umgebung.

Hauptkapitel I

2 Produkt- und Konfigurationsverwaltung
Es werden alle Software-Elemente einer Konfiguration in der Produktbibliothek archiviert und katalogisiert, so daß die Software-Elemente weder absichtlich noch unabsichtlich zerstört werden können. Alle Software-Elemente sind eindeutig identifizierbar.

3 Änderungsmanagement (Konfigurationssteuerung)

a Erfassung und Verwaltung eingehender Fehlermeldungen, Problemmeldungen und Verbesserungsvorschläge in Form von Änderungsanträgen/Problemmeldungen.

b Entscheidung über die Bearbeitung von Änderungsanträgen/Problemmeldungen (Ablehnung/Annahme; Auswahl eines Lösungsvorschlags) unter Berücksichtigung der technischen und zeitlichen Auswirkungen auf den Entwicklungsverlauf und Veranlassung der Bearbeitung.

c Abschluß der Änderung und Information aller Betroffenen.
Eine Änderung hat abhängig vom Entwicklungsfortschritt und von getroffenen Entscheidungen einen definierten Status (Abb. 6.3-8).

4 Konfigurationsmanagement-Dienste

a Daten verwalten mit dem Ziel eines zentralen bzw. unternehmensweiten Datenkatalogs und konsistenter, vereinheitlichter Datendefinition.

b SW-/HW-Produkte katalogisieren, um die Wiederverwendbarkeit der Produkte zu ermöglichen.

c Schnittstellen koordinieren, damit sie kompatibel bleiben.

d Ergebnisse sichern (Datensicherung).

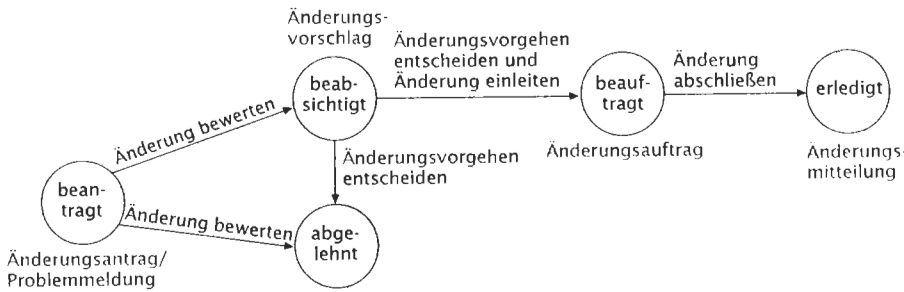


Abb. 6.3-8: Mögliche Zustände einer Änderung und zugehörige Formulare

e Konfigurationsmanagement-Dokumentation führen, damit Detailunterlagen und Übersichten erstellt werden können.

f *Release*-Management durchführen, damit eine umfassende, nachvollziehbare Dokumentation über den gesamten Projektverlauf vorhanden ist.

g Projekthistorie führen, damit eine umfassende, nachvollziehbare Dokumentation über den gesamten Projektverlauf vorhanden ist.

Beispiel 1d

Der Änderungsantrag für das Benutzerhandbuch V2.4 kann beispielsweise vom Auftraggeber kommen. Ein im Konfigurationsmanagement-Plan festgelegtes Gremium – oft *Change Control Board* (CCB) genannt – prüft den Änderungsantrag. Die beantragte Änderung wird bewertet, und es wird ein Änderungsvorschlag erstellt. Wird der Änderungsvorschlag angenommen, dann wird anschließend über das Änderungsvorgehen entschieden und ein Änderungsauftrag erteilt. Im Falle des Benutzerhandbuches steht im Änderungsvorschlag, welche Kapitel überarbeitet werden müssen. Der Änderungsauftrag geht an den Handbuchautor mit den Vorgaben, welche Änderungen in welcher Form durchzuführen sind. Liegt das überarbeitete Benutzerhandbuch vor, dann wird eine Änderungsmitteilung an alle betroffenen Personenkreise verschickt.
Damit geht das Benutzerhandbuch in den Zustand »vorgelegt« über (Abb. 6.3-7), wird von der Qualitätssicherung überprüft und ist anschließend »akzeptiert« oder wird wieder in den Zustand »in Bearbeitung« versetzt.

Die Werkzeuge, die die Konfigurationen verwalten, müssen in der Lage sein, folgende typische Anfragen zu beantworten /Sommerville 89, S. 555/:

- Welcher Kunde hat eine spezielle Version des Produktes erhalten?
- Welche Hardware- und Systemsoftware-Konfiguration wird für eine geplante Produktversion benötigt?
- Wie viele Versionen des Produkts wurden wann erzeugt?
- Welche Versionen des Produkts sind betroffen, wenn eine spezielle Komponente geändert wird?

- Wieviele Änderungsanträge oder Problemmeldungen sind für eine spezielle Version noch unerledigt?
- Wieviele Fehlermeldungen liegen für eine spezielle Version vor?

Einen Funktionsüberblick über das Submodell Konfigurationsmanagement des V-Modells zeigt die Abb. 6.3-9.

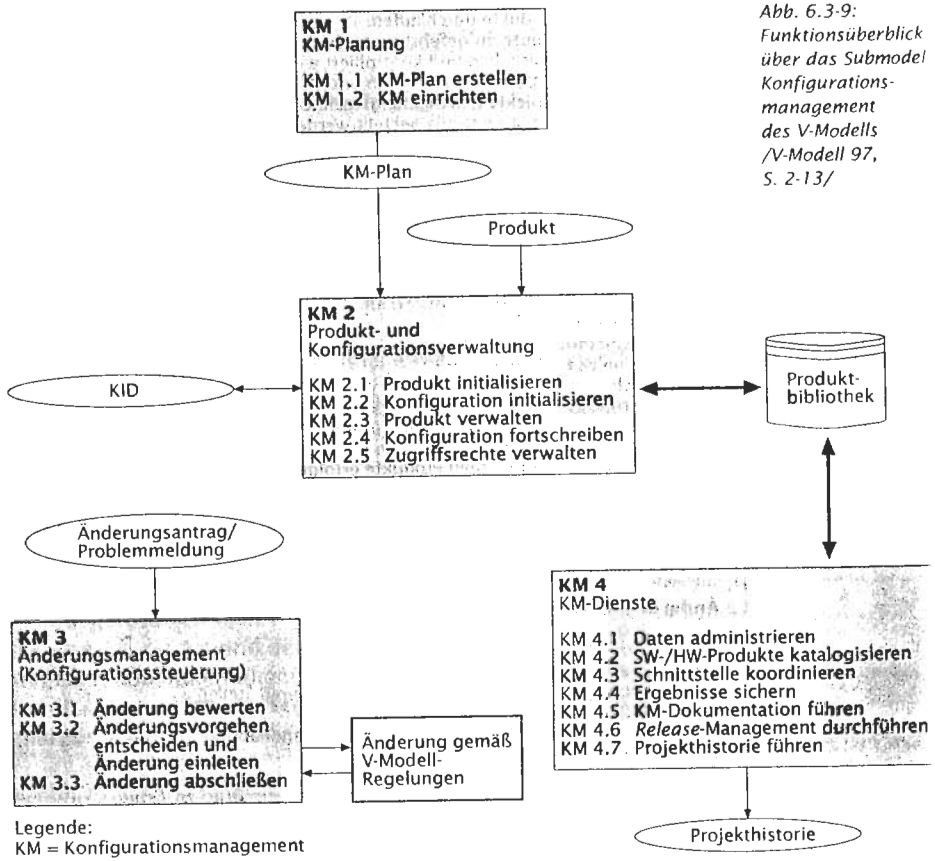


Abb. 6.3-9: Funktionsüberblick über das Submodel Konfigurations-management des V-Modells /V-Modell 97, S. 2-13/

Zur Durchführung des Konfigurationsmanagements (KM) werden verschiedene Produkte benötigt (hier dargestellt in Anlehnung an das V-Modell /V-Modell 97, S. 8-45ff/):

Abschnitt 3.3.2 KM-Produkte

Der **Konfigurationsmanagement-Plan** legt alle organisatorischen und verfahrenstechnischen Details des Konfigurationsmanagements fest. Aufbau und Inhalte eines KM-Plans zeigt Abb. 6.3-10.

KM-Plan

Abb. 6.3-10: Aufbau und Inhalte eines Konfigurations-management-Plans

2 Einführung des Konfigurationsmanagements (KM)
2.1 Produktbibliothek
Hier ist die Produktbibliothek mit ihrer Systematik der Produkt-, Nutzer-, Rechte- und Relationsverwaltung zu beschreiben, bzw. auf geeignete Dokumente oder Handbücher zu verweisen.
2.2 Projektspezifische Festlegungen
Dieser Gliederungspunkt legt fest,
- welche Produkte unter KM-Kontrolle gestellt werden,
- anhand welcher Kriterien die Konfigurationseinheiten gebildet werden,
- welche Zustände Produkte durchlaufen,
- welche Produktattribute mitgeführt werden,
- wie Zugriffsrechte vergeben und kontrolliert werden,
- wie Produkte von Unterauftragnehmern unter KM-Kontrolle gestellt werden,
- welche sonstigen Objekte (Entwicklungsrechner, Werkzeuge, Prüfumgebung usw.) unter KM-Kontrolle gestellt werden,
- wie KM-Instanzen (z.B. *Change Control Board)* eingeführt und besetzt werden,
- welche Hilfsmittel (z.B. KM-Werkzeuge, Formblätter) bereitgestellt werden.

Zu Produktattribute:
Es wird festgeschrieben, welche Produktattribute geführt werden, welchen Initialwert sie besitzen und wie mögliche Wertfolgen der Attribute aussehen können.
Beispiele für Produktattribute sind:
- Zustand
- Eigentümer
- Bearbeiter
- Entstehungsdatum
- Änderungshistorie
- Vorgegebene Bearbeitungsdauer
- Zugriffsrecht
- Klassifizierung (Vertraulichkeitsvermerke)
- Schlüsselbegriffe (dienen zum Wiederauffinden)
- Zugehörige andere Produkte (Hier kann eine Verkettung aller zu einem Modul/einer Komponente gehörenden Produkte erfolgen)
- Versionsattribute (Ist das gegenwärtige Produkt eine Version eines Vorgängers, so werden die Versionsattribute eingetragen. Jede Änderung an dem Produkt, die mit einer Zustandsänderung verknüpft ist, bedeutet automatisch eine Änderung der Versionsbezeichnung (normalerweise Versionserhöhung)).

2.3 Konventionen zur Identifikation

3 Änderungsmanagement
3.1 Änderungsprozedur
3.1.1 Verfahren im Änderungswesen
3.1.2 Änderungsaufträge
3.1.3 Schnittstellen zum Auftraggeber und zu anderen Stellen
3.1.4 Änderungskontrolle und Statusanzeige
3.2 Formulare des Änderunsgwesens und deren Handhabung
3.3 Versionskontrolle
3.4 Dokumente des Konfigurationsmanagements

4 Sicherung und Archivierung

Quelle: /V-Modell 97, S.8-45 ff/, gekürzt und leicht modifiziert.

KID In **Konfigurations-Identifikationsdokumenten** werden die einzelnen Konfigurationen beschrieben. Es werden folgende Informationen angegeben:
- Version/Nummer der Konfiguration
- Projektbezeichnung
- KM-Verantwortlicher
- Zustand der Konfiguration
- Datum der letzten Konfigurationsänderung

Die einzelnen Software-Elemente der Konfiguration werden dann mit einer eindeutigen Kennung und ihrer Version aufgelistet.

Die Relationen zwischen den einzelnen Software-Elementen (interne Bezüge innerhalb der Konfiguration) sind ebenfalls zu dokumentieren. Um Generierungen automatisieren zu können, müssen hier redundante und kopierte Produktteile, Schnittstellen und die hierarchischen Beziehungen zwischen den Bausteinen festgehalten werden.

Änderungsformulare

Für das Änderungsmanagement werden folgende **Änderungsformulare** benötigt:

Änderungsantrag/Problemmeldung
Enthält den schriftlich formulierten Wunsch eines Projektmitglieds, des Auftraggebers oder Anwenders nach Durchführung einer Änderung.

Änderungsvorschlag
Beinhaltet die technische und wirtschaftliche Bewertung des Änderungsantrags/der Problemmeldung und zeigt die verschiedenen Möglichkeiten zur Durchführung der gewünschten Pflege/Änderung auf.

Änderungsauftrag
Enthält eine Verfeinerung des ausgewählten Lösungswegs aus dem Änderungsvorschlag. Der Auftrag ergibt sich aus einem Genehmigungsverfahren. Der Änderungsauftrag hat Anforderungscharakter und legt die durchzuführende Änderung im Detail fest.

Änderungsmitteilung
Beschreibt die aufgrund eines Änderungsantrags/einer Problemmeldung durchgeführten Änderungen.

Änderungsstatusliste
In dieser Liste werden alle eingehenden Änderungsanträge/Problemmeldungen eingetragen. Aufgabe der Liste ist es, einen Überblick über die Anzahl, die Art und den Bearbeitungszustand von Änderungsanträgen/Problemmeldungen zu erhalten, den Status aller Änderungen sichtbar zu machen und die Verfolgung aller Änderungsaufträge sicherzustellen.

baseline →Referenzkonfiguration
Konfigurations-Identifikationsdokument (KID) Beschreibt, welche →Software-Elemente zu einer →Software-Konfiguration gehören. Der Begriff stammt aus dem V-Modell. Ein Konfigurations-Identifikationsdokument unterliegt der Versionszählung (→Version).
Kontrolle Alle Managementaktivitäten, die dazu beitragen, daß Vorgaben für eine Software-Entwicklung (Plan, Standards) mit den laufenden Aktivitäten übereinstimmen. Bei Abweichungen sind Aktivitäten zu initiieren, damit Soll und Ist wieder in Übereinstimmung kommen.
Metrik Sie soll in kompakter Form über technisch oder wirtschaftlich interessierende Sachverhalte informieren und meßbare Eigenschaften dieser Sachverhalte in Ziffern ausdrücken.
Referenzkonfiguration Ein zu einem bestimmten Zeitpunkt im Entwicklungsprozeß ausgewähltes und freigegebenes Zwischenergebnis, das zu einer →Software-Konfiguration zusammengefaßt wird, damit man sich im weiteren Entwicklungsprozeß darauf beziehen kann. Im Englischen *baseline* genannt.

Release Größere Änderungen an einer →Version führen zu einem neuen *Release*-Stand (Freigabe-Stand) eines → Software-Elements.

Software-Element Atomarer Bestandteil einer Software-Konfiguration oder selbst eine Software-Konfiguration. Ein Software-Element kann ein Dokument, ein Programm oder ein Werkzeug sein. Jedes Software-Element unterliegt der Versionszählung (→Version), ist identifizierbar und maschinenlesbar.

Software-Konfiguration Legt fest, welche →Software-Elemente zu einem bestimmten Zeitpunkt im Produktlebenszyklus gemeinsam ein Produkt oder ein Zwischenergebnis (→Referenzkonfiguration) darstellen. (→Konfigurations-Identifikationsdokument, →Software-Konfigurationsmanagement).

Software-Konfigurationsmanagement Einrichtung, Verwaltung und Überwachung von →Software-Konfigurationen, so daß Zusammenhänge und Unterschiede zwischen den verschiedenen Konfigurationen erkennbar sind und daß auf vorangegangene Konfigurationen jederzeit zugegriffen werden kann.

Variante Zeitgleich nebeneinander liegende Ausführungsstände von →Software-Elementen oder unterschiedlichen Implementierungen derselben Schnittstelle, d.h. gleiche Funktionalität (siehe auch →Version).

Version Ausführungsstand eines →Software-Elements zu einem bestimmten Zeitpunkt. Versionen sind zeitlich nacheinander liegende Ausprägungen eines Software-Elements. Er wird in der Regel durch eine Nummer angegeben. Das →Software-Konfigurationsmanagment muß eine Konvention zur Versionszählung und Versionsbezeichnung festlegen *(→Release)*.

Die notwendige Ergänzung zu jeder Planung ist die Kontrolle. Die Aufgabe der Kontrolle besteht darin, Abweichungen von Plänen und Standards festzustellen und durch die Veranlassung korrigierender Maßnahmen Soll und Ist wieder in Einklang zu bringen.

Um Prozeß- und Produktabweichungen feststellen zu können, ist es erforderlich, sowohl den Prozeß als auch das Produkt zu vermessen. Dazu ist es erforderlich, für relevante Eigenschaften Software-Metriken zu definieren, sowie während der Entwicklung diese Metriken zu erfassen und zum Zwecke der Kontrolle und Prognose auszuwerten.

Um die Kontrolle des Produktes sowohl während der Entwicklung als auch in der Wartung und Pflege sicherzustellen, ist ein Software-Konfigurationsmanagement erforderlich. Alle Software-Elemente werden einer Versionszählung unterworfen. Änderungen an Software-Elementen ergeben eine neue Version. Größere Änderungen führen zu einem neuen *Release*. Varianten kennzeichnen oft parallel vorliegende Ausprägungen von Software-Elementen.

In einem Konfigurations-Identifikationsdokument werden alle Software-Elemente, die zu einer Software-Konfiguration gehören, aufgelistet. Software-Konfigurationen, die Zwischenergebnisse darstellen, werden als Referenzkonfiguration *(baseline)* bezeichnet.

/Conte, Dunsmore, Shan 86/
Conte S.D., Dunsmore H.E., Shan V.Y., *Software Engineering Metrics and Models*, Menlo Park: Benjamin/Cummings Publishing Company 1986, 396 Seiten
Umfangreiche Darstellung einer Software-Meßtechnik mit ausführlicher Beschreibung vieler Metriken.

/DeMarco 82/
DeMarco T., *Controlling Software Projects*, Englewood Cliffs: Yourdon Press 1982, 284 Seiten
Ausführliche Behandlung des Kontrollaspekts von Software-Entwicklungen. Software-Metriken, Kostenmodelle und Software-Qualität werden ebenfalls dargestellt.

/Grady, Caswell 87/
Grady R.B., Caswell D.L., *Software Metrics: Establishing a Company-Wide Program*, Englewood Cliffs: Prentice Hall 1987, 288 Seiten
Ausgezeichnetes Buch, das die Definition und Einführung von Metriken bei der Firma Hewlett-Packard behandelt. Gute Hilfestellung, um ein eigenes Metrikprogramm zu realisieren.

/Grady 92/
Grady R.B., *Practical Software Metrics for Project Management and Process Improvement*, Englewood Cliffs: Prentice Hall 1992, 270 Seiten
Ausgezeichnetes Buch, das das umfangreiche Metrik-Programm bei der Firma Hewlett-Packard beschreibt. Es wird dargestellt, wie man ein eigenes Metrik-Programm organisiert oder ein vorhandenes verbessert. Viele Grafiken geben Ergebnisse von HP-Projekten wieder. Das Buch enthält 400 Literaturhinweise.

/Möller, Paulisch 93/
Möller K.H., Paulisch D.J., *Software-Metriken in der Praxis*, München: Oldenburg Verlag 1993, 262 Seiten
Enthält Hinweise für die Einführung eines Metrikprogramms, gibt Beispielmetriken an und berichtet über Erfahrungen bei verschiedenen Firmen.

/Tichy 94/
Tichy W. F., (ed.), *Configuration Management*, Chichester: John Wiley & Sons 1994
Behandelt den aktuellen Stand des Konfigurationsmanagements.

/Tichy 95/
Tichy W. F., *Software-Konfigurationsmanagement: Wie, wann, was, warum?* In: Proceedings Softwaretechnik 95, Braunschweig 1995, S. 17–23
Guter Übersichtsartikel zum Konfigurationsmanagement mit Hinweisen auf den Stand der Forschung.

Zitierte Lit

/Baumann, Richter 92/
Baumann P., Richter L., *Wie groß ist die Aussagekraft heutiger Software-Metriken?*, in: Wirtschaftsinformatik, Dez. 1992, S. 624–631.

/Bersoff 84/
Bersoff E.H., *Elements of Software Configuration Management*, in: IEEE Transactions on Software Engineering, Jan. 1984, S. 79–87

/Boehm 91/
Boehm B.W., *Software Risk Management: Principles and Practices*, in: IEEE Software, Jan. 1991, pp. 32–41

/Humphrey 89/
Humphrey W.S., *Managing the Software Process*, Reading: Addison-Wesley 1989, 494 Seiten.

/Femick 90/
Femick S., *Implementing Management Metrics: An Army Program*, in: IEEE Software, March 1990, S. 65–72

/Fenton 94/
Fenton N., *Software Measurement: A Necessary Scientific Basis*, in: IEEE Transactions on Software Engineering, March 1994, S. 199–206

/Itzfeld 83/
Itzfeld W., *Methodische Anforderungen an Software-Kennzahlen*, in: Angewandte Informatik, 2/83, S. 55–61.

/Itzfeld, Schmidt, Timm 84/
Itzfeld W., Schmidt M., Timm M., *Spezifikation von Verfahren zur Validierung von Software-Qualitätsmaßen*, in: Angewandte Informatik, 1/1984, S. 12–21.

/KBSt 92/
Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung, *Vorgehensmodell*, Band 27/1, August 1992, Bundesanzeiger.

/Kearney et al. 84/
Kearney J.K., Sedlmeyer R.L., Thompson W.B., Gray M.A., Adler M.A., *Software Complexity Measurement*, in: Communications of the ACM, Nov. 86, S. 1044–1050

/Mahler 94/
Mahler, A., *Variants: Keeping things together and telling them apart*, in: Configuration Managment, Chichester: John Wiley & Sons 1994, pp. 73–97

/MS Project 94/
Microsoft Project, Benutzerhandbuch, Microsoft Corporation 1994

/Pfleeger 93/
Pfleeger S.L., *Lessons Learned in Building a Corporate Metrics Program*, in: IEEE Software, May 1993, S. 67–74

/Schmidt 84/
Schmidt M., *Software-Metrik*, in: Informatik-Spektrum, Das aktuelle Schlagwort, 7/1984, S. 41–42.

/Sommerville 89/
Sommerville I., *Software-Engineering* - 3rd ed., Wokingham: Addison-Wesley, 1989

/Thayer 90/
Thayer R.H., *Tutorial Software Engineering Project Management*, Los Alamitos: IEEE Computer Society Press 1990.

/V-Modell 97/
Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell, Teil 1: Regelungsteil, Allgemeiner Umdruck Nr. 250/1, Juni 1997, BWB IT V 15, Koblenz

/Wallmüller 90/
Wallmüller E., *Software-Qualitätssicherung in der Praxis*, München: Carl Hanser Verlag 1990

/Zuse 85/
Zuse H., *Meßtheoretische Analyse von statischen Softwarekomplexitätsmaßen*, TU Berlin, Dissertation 1985.

Muß-Aufgabe
10 Minuten

1 *Lernziel: Ziele, Aufgaben und Probleme der Kontrolle schildern können.*

a Welche Aufgaben müssen bei der Kontrolle einer Software-Entwicklung durchgeführt werden.

b Ordnen Sie jeder Aufgabe die bearbeitende(n) Person(en) zu. Gehen Sie von den Rollen »Manager« und »Entwickler« aus und begründen Sie Ihre Lösung.

Muß-Aufgabe
45 Minuten

2 *Lernziel: Gegebene Metriken klassifizieren und auf Gütekriterien hin überprüfen können.*

a Geben Sie eine sinnvolle Definition für die Software-Metrik »Umfang Objektcode« an.

b Klassifizieren Sie die in **a** definierte Metrik anhand von Tab. 6.2-2.

c Überprüfen Sie die Metrik anhand der Gütekriterien für die Software-Metriken.

Muß-Aufgabe
35 Minuten

3 *Lernziele: Die Konzepte Konfiguration, Version und Variante erklären können. Anhand von Beispielen Konfigurationen beschreiben und Versionszählungen durchführen können.*

a Welche Eigenschaften hat ein Software-Element und wie lassen sich Software-Elemente klassifizieren.

b Erklären Sie den Zusammenhang zwischen einer Software-Konfiguration, einem Software-Element und einem Konfigurations-Identifikationsdokument (KID).

c Optional, nur wenn die objektorientierte Analyse bekannt ist:
Entwerfen Sie ein Metamodell, das es ermöglicht, Konfigurations-Identifikationsdokumente computergestützt zu verwalten. Verwenden Sie dazu die in **a** und **b** identifizierten Komponenten und deren Eigenschaften.

Muß-Aufgabe
60 Minuten

4 *Lernziel: Das verwendete Projektplanungssystem für die Projektkontrolle einsetzen können.*

Die Grundlage zu den weiteren Aufgabenstellungen ist das in Kapitel 2.5 eingeführte Beispiel der Projektplanung der Definitionsphase einer Anwendung. Übernehmen Sie die in Abb. 2.5-5, 2.5-8 und 2.6-1 aufgeführten Daten in das von Ihnen verwendete Planungssystem. Für /MS Project 94/ finden Sie die geplanten Daten in der Datei »LE8A4.MPP« auf der beiliegenden CD.

a Bei der Projektkontrolle hat sich herausgestellt, daß
- der Vorgang »Pflichtenheft erstellen« mit einer Verzögerung von einem Tag begonnen hat,
- der Vorgang »Pflichtenheft erstellen« zu 100 % abgeschlossen ist und
- der Vorgang »Benutzerhandbuch erstellen« zu 50 % abgeschlossen ist.

Übernehmen Sie die aktuellen Projektdaten in die Projektplanung und prüfen Sie das erhaltene Ergebnis anhand der in Abb. 6.1-3 dargestellten Daten.

b Der weitere Verlauf des Projekts wird wie geplant durchgeführt.
Nehmen Sie die Ist-Daten in das Planungssystem auf und erstellen Sie eine Projektübersicht.

c Um wieviele Tage verzögert sich die Durchführung des geplanten Projekts insgesamt?

Muß-Aufgabe
45 Minuten

5 *Lernziele: Das Checkin/Checkout-Modell auf Beispiele anwenden können. Anhand von Szenarien Konfigurations- und Änderungsmanagement unter Berücksichtigung der verschiedenen Zusammenhänge exemplarisch durchführen können.*

a Installieren Sie eines der auf der CD enthaltenen Werkzeuge zur Versionsverwaltung. Erstellen sie ein Archiv für zwei Beispieldokumente. Buchen Sie dort die zunächst leeren Dateien »ex1.txt« und »ex2.txt« ein.

b Ändern Sie jetzt die ersten Versionen der Dateien, indem sie einen beliebigen Text hinzufügen. Welche Arbeitsschritte sind dazu notwendig?

c Wiederholen Sie die in b getätigten Arbeitsschritte mehrmals. Wie reagiert die Versionsverwaltung, wenn Sie versuchen eine Datei zweimal hintereinander auszubuchen?

d Buchen Sie die erste Version der Datei »ex2.txt« aus.

e Worauf müssen Sie achten, wenn Sie eine Programmiersprache benutzen, die deklarative Anteile von ihren Implementationen trennt.

Muß-Aufgabe
5 Minuten

6 *Lernziel: Die Konzepte Konfiguration, Version und Variante erklären können.*

Erklären Sie den Zusammenhang zwischen einer Konfiguration, einer Version und einer Variante.