Projektmanagement

Karl Blümlinger

17. Juni 2003

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Ziel dieses Skriptums

Dieses Skriptum dient als Ergänzung zur Vorlesung *Projektmanagement*. Es erhebt nicht den Anspruch, den man an ein Buch über Projektmanagement stellen würde, nämlich eine umfassende in sich geschlossene Abhandlung des Themas. Ich bin natürlich bemüht, diesem Anspruch möglichst nahe zu kommen, soweit es meine Zeit erlaubt. Stilblüten, Rechtschreibfehler und Inkonsistenzen möge mir der Leser dieses Skriptums verzeihen. Um sich daher nicht nur auf dieses Skriptum verlassen zu müssen, wird es an entsprechenden Stellen Literaturverweise geben.

Die einzelnen Kapitel dieses Skriptums werden im Laufe dieses Semesters vervollständigt.
(FIXXME: hier kommt noch eine Kapitelübersicht) ⇐

1.2 Was ist ein Projekt ?

Die Frage, was ist ein Projekt, mag einigen Lesern überflüssig erscheinen, schließlich ist man mit diesem Begriff stets konfrontiert. Eine Begriffsdefinition ist an dieser Stelle aber unbedingt erforderlich, da einige der hier vorgestellten Methoden des Projektmanagements nur auf Projekte sinnvoll und zielführend angewandt werden können. Der Duden definiert ein Projekt als:

> [großangelegte] geplante oder bereits begonnene Unternehmung; [großangelegtes] Vorhaben;

Neben der Definition des Dudens gibt es in der Fachliteratur unzählige Definitionen, von denen hier einige vorgestellt werden (in [1] findet man eine etwas ausführlichere Abhandlung zu diesem Thema).

Bernd Madauss [1] definiert ein Projekt als

> ein einmaliges Vorhaben, das durch zeitliche Befristung, hohe Komplexität und interdisziplinäre Aufgabenstellung gekennzeichnet ist.

Auch in der deutschen Industrienorm (DIN) findet man eine Definition des Begriffs [4]:

> Ein Vorhaben, das im wesentlichen durch Einmaligkeit der Bedingungen in ihrer Gesamtheit gekennzeichnet ist, wie z.B. Zielvorgabe,
>
> - zeitliche, finanzielle, personelle oder andere Begrenzungen,
> - Abgrenzung gegenüber anderen Vorhaben und
> - projektspezifische Organisationen.

Zuletzt sei hier noch die Definition des PMI gebracht ([2]):

> A project is a temporary endeavor undertaken to create a unique project or service.

Projekte sind also zeitlich begrenzt, das heißt, sie haben einen klar definierten Anfang und Abschluss. Das Ende eines Projekts kann viele Ursachen haben. Von einem erfolgreichen Abschluss kann gesprochen werden, wenn die Projektziele termingerecht erreicht wurden. Ein weiteres wesentliches Merkmal eines Projekts ist seine Einzigartigkeit. Hierbei muss der Kontext berücksichtigt werden, in dem das Projekt durchgeführt wird. Aufgaben, die für eine Firma Routine ist, kann für eine andere Firma durchaus Projektcharakter besitzen. Dieser Unterschied ist wesentlich, da Routineaufgaben in Hinsicht auf Planung und Management einfacher zu bewerkstelligen sind als Projekte, die Methoden des Managements daher differieren. Für einfache Vorhaben, auch wenn sie neuartig und zeitlich begrenzt sind, ist die Bezeichnung *Projekt* unangebracht (da weder ein Projektmanagement noch die in diesem Dokument vorgestellten Methoden zur Durchführung notwendig sind). Beispiele von Projekten sind:

- Der Bau einer Brücke
- Die Entwicklung von Software
- Eine Werbekampagne
- Restrukturierung einer Firma

All diesen Vorhaben ist gemeinsam, dass sie immer unter verschiedenen Rahmenbedingungen durchgeführt werden. Eine Brücke wird selten zweimal am selben Ort gebaut (es ändern sich daher zumindest die geologischen Verhältnisse), eine Werbekampagne muss an die beworbene Person angepasst werden, Software unterscheidet sich durch ihre Funktionalität und Architektur, Firmen durch Größe und Organisation (die hier aufgezeigten Unterschiede bzw. Eigenschaften sind natürlich nicht vollständig).

Die Einzigartigkeit von Projekten impliziert auch eine progressive Ausarbeitung (engl. progressive evaluation) ihrer Eigenschaften. *Progressiv* bedeutet hier *Schritt für Schritt*. Damit ist gemeint, dass die Eigenschaften des Produkts in der frühen Projektphase grobkörnig festgelegt werden (jedoch so genau wie möglich), und später verfeinert werden. Das Ideal wäre, alles zu Beginn genau festzulegen und zu definieren, doch leider ist dieses in der Praxis wegen der Neuartigkeit des zu entwickelnden Produkts kaum zu erreichen. Mit zunehmender Projektdauer, die durch eine immer feinere Zerlegung in Projektteile begleitet wird, wird auch das Produkt immer genauer bekannt. Die Schwierigkeit dieser progressiven Ausarbeitung liegt darin, die zu Beginn festgelegten Merkmale des

Produkts nicht zu unterwandern oder abzuändern, sondern nur zu verfeinern (siehe Kapitel 4).

Die oben angeführten Definitionen des Begriffs *Projekt* entstanden keineswegs aus theoretischen Überlegungen, vielmehr weisen viele Vorhaben aus der Praxis genau diese projektspezifischen Eigenschaften auf. Und eben diese Charakteristika erfordern eine durchdachte Vorgehensweise bei der Umsetzung dieser Vorhaben. Ziel dieses Dokuments ist es, Methoden vorzustellen, die die Verwirklichung eines Projekts erleichtern.

1.3 Was ist Projektmanagement

Der Begriff *Projektmanagement* wird in der Literatur unterschiedlich definiert. Das PMI versteht darunter ausschließlich die Anwendung geeigneter Methoden, Werkzeuge und Techniken, um die Projektziele zu erreichen [2]. In [1] und [3] wird darunter die Projektleitungsfunktion verstanden, die sich zur Durchführung von Projekten gewisser Methoden bedient (das PMI bezeichnet diese Tätigkeit als *Mangement by projects*). Dieses Dokument beschreibt primär die Methoden des Projektmanagements, die erforderliche organisatorische Struktur, um Projekte optimal zu leiten wird im Rahmen dieses Dokuments nicht behandelt. Damit soll nicht gesagt werden, das die organisatorische Struktur von Institutionen erst in zweiter Linie zu betrachten ist.

Verantwortlich für die Durchführung von Projekten ist ein Projektteam, dessen Arbeit die Initiierung, Planung, Ausführung, Kontrolle und das Beenden des Projekts umfasst. Die verwendeten Methoden müssen der Komplexität und Größe des jeweiligen Projekts angepasst sein. Es besteht sonst die Gefahr einer überhand nehmenden Bürokratie, die die Projektleitung und die Projektmitglieder unnötig belasten. Die unmittelbare Konsequenz ist eine Verlängerung der Projektdauer und damit auch eine Erhöhung der Projektkosten. Eine sich nach und nach einstellende Haltung führt schließlich dazu, dass die Methoden nicht konsequent umgesetzt werden, ihren Zweck damit nicht erfüllen.

1.4 Probleme des Projektmanagements

Probleme im Projektmanagement äußern sich großteils in Kosten- und Terminüberschreitungen, sowohl bei privaten als auch bei öffentlichen Vorhaben. Als klassisches Beispiel lässt sich der Bau der Akropolis anführen, zu deren Fertigstellung die Kriegskasse des attischen Bundes geplündert wurde. Dass sich an dieser Problematik nicht viel geändert hat, zeigen viele Großvorhaben aus heutiger Zeit: Beim Bau des Wiener AKH wurden die Projektkosten um das 7,5-fache überzogen, aus geplanten 10 Jahren Bauzeit wurden letztendlich 40 Jahre (die Angabe der Kostenüberschreitung ist nicht inflationsbereinigt). Der Bau des Eurotunnels verschlang statt den projektierten 12,5 Milliarden Mark 25 Milliarden Mark.

Eine Studie der Standish Group, der sogenannte *Chaos Report* aus dem Jahre 1994 [6], zeigt eine fast unglaubliche Statistik:

a) Nur 16.2% aller Projekte erreichten ihr Projektziel (d.h. kosten u. termingerechter Abschluss, spezifizierte Anforderungen werden umgesetzt).

b) 52.7% aller Projekte wurden abgeschlossen, aber deren Projektziel wurde verfehlt (d.h. Kosten und/oder Termine werden nicht eingehalten und/oder nicht alle Anforderungen werden erfüllt.

c) 31.1% aller Projekte wurden während ihrer Entwicklung gestoppt.

Optimistisch stimmt der Vergleich der Standish Group mit dem Jahr 1998 [7]:

	Succeeded	Challenged	Failed
1998	26%	46%	28%
1996	27%	33%	40%
1994	16%	53%	31%

Wobei die Kategorisierung der Projekte in *succeeded*, *challanged* und *failed* denen in obiger Aufzählung (a,b,c) entspricht. Aus dieser Tabelle geht hervor, dass im Vergleich zum Jahr 1994 im Jahr 1998 um 10% mehr Projekte erfolgreich abgeschlossen wurden. Als Grund für diesen Wandel führt die Standish Group kleinere Projekte, besseres Management und die Verwendung von standard Infrastruktur an. Freilich erreichten auch 1998 74% aller IT-Projekte ihr Projektziel nicht. Die grundlegenden Ursachen dieser Misere sind, geordnet nach Wichtigkeit [6]:

1. Fehlender User Input
2. Unvollständige Anforderungen/Spezifikationen
3. Wechselnde Anforderungen/Spezifikationen
4. Fehlende Unterstützung des Managements
5. Technologische Inkompetenz
6. Fehlende Resourcen
7. Unrealistische Erwartungen
8. Unklare Zielvorgaben
9. Unrealistische Zeitvorgaben
10. Neue Technologien

Es stellt sich nun die Frage, warum sich gerade bei IT-Projekten und bei Software Projekten im Speziellen ein derartiges Bild ergibt: Die Informatik ist eine relativ junge Disziplin. In der Anfangszeit bestand Software aus einzelnen, unabhängigen und überschaubaren Programmen. Die Entwicklung von Software wurde als künstlerische Tätigkeit empfunden, die Programmierer hatten entsprechende Freiheiten um ihre Kreativität nicht zu schmälern. Die Anforderungen an die Software wurden bald umfangreicher, die Programme daher komplexer und von Einzelpersonen schwerer durchschaubar. Doch die Herangehensweise an die Entwicklung änderte sich nur marginal. In den 70ger Jahren setzte sich die Erkenntnis durch, das komplexe Software nur durch eine systematische Vorgehensweise beherrschbar ist, wie es in anderen Ingenieursdisziplinen schon längst praktiziert wurde. Es entstand die Disziplin des *Software Engineering*, dessen Ziele die

- Einhaltung der projektierten Kosten
- Einhaltung der vereinbarten Termine
- Erfüllung der vereinbarten Spezifikationen/Anforderungen
- Verbesserung der Qualität

sind. Wie die Studie der Standish Group zeigt, ist man von diesen Zielen noch weit entfernt. Der Grund dafür ist zum einen die Tatsache, dass Software-Entwicklung eine hoch komplexe Aufgabe ist und zum anderen werden die Erkenntnisse des Softwareengineerings ignoriert, beziehungsweise nicht konsequent umgesetzt.

Zum Ende dieser Einleitung noch ein Zitat aus dem Chaos Report 1994:

> All success is rooted either in luck or failure. If you begin with luck, you learn nothing but arrogance. However, if you begin with failure and learn to evaluate it, you also learn to succeed. Failure begets knowledge. Out of knowledge you gain wisdom, and it is with wisdom that you can become truly successful.

Kapitel 2

Der Lebenszyklus eines Projekts

Wie in der Einleitung ausführlich dargelegt wurde, ist ein Projekt ein Vorhaben mit definiertem Anfang und Ende. Diese Definition erlaubt eine klare Bestimmung der Projektdauer und der Projektkosten. Der Lebenszyklus (*project life cycle*) eines Projekts beschreibt den Ablauf desselben von Anfang bis zum Ende.

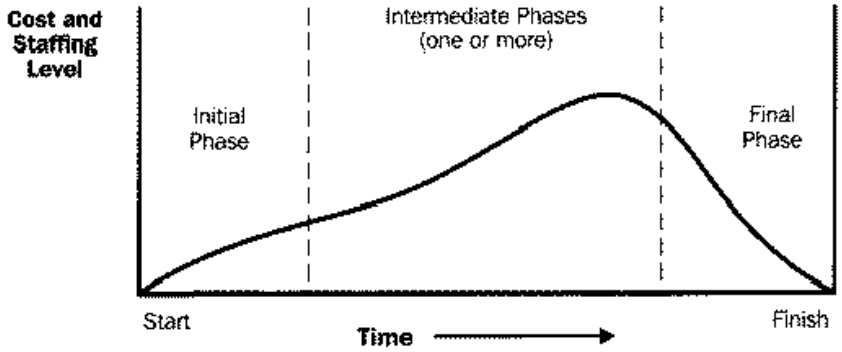

Abbildung 2.1: Allgemeiner Lebenszyklus eines Projekts (Quelle [2])

Der Lebenszyklus eines Projekts wird in sogenannte Projektphasen unterteilt. Abbildung 2.1 zeigt ein sehr allgemeines Phasenmodell (auch Vorgehensmodell genannt) eines Projekts. Ein Phasenmodell beschreibt modellhaft, d.h. idealisiert und abstrahierend, den Entwicklungs-, Betriebs- und Pflegeprozess eines zu entwickelnden Produkts. Die initiale Phase besteht meist aus einem Projektplan, der ersten Definition von (Kunden-) Anforderungen, Machbarkeitsstudien (Durchführbarkeitsanalyse) und erste Kosten-, Risiko- und Zeitanalysen. Die initiale Phase sollte genug Aufschluss darüber geben, ob ein Projekt durchgeführt werden sollte. Darauf folgen Vorgehensmodell abhängig mehrere Phasen, deren Ergebnis das zu entwickelnde Produkt ist. Die abschließende Phase kann z.B. die Übernahme des Produkts durch den Auftraggeber sein.

2.1 Allgemeine Phasenmodelle

Das Ziel von Phasenmodellen ist eine übersichtliche und strukturierte Entwicklung eines Projekts. Ein neues Projekt stellt für eine Firma ein durchaus nicht zu unterschätzendes Risiko dar. Zur Erinnerung: Ein Projekt ist ein einmaliges, zeitlich begrenztes und komplexes Vorhaben. Die Einteilung in Phasen soll der Projektleitung ermöglichen, nach dem Abschluss einer jeden Phase eine go/no-go Eintscheidung zu treffen, d.h. zu entscheiden, ob das Projekt fortgeführt werden soll oder nicht. Allgemein lässt sich ein Projekt in folgende Phasen unterteilen:

1. *Konzeptformulierung*: Untersuchung der Machbarkeit von Kundenwünschen und der Einbettung des zu entwickelnden Produkts in seinem gedachten Umfeld. Zudem werden in dieser Phase erste Zeit- und Kostenpläne und ein Managementkonzept erstellt. Erforderliche Qualifikation der Mitarbeiter in dieser Phase: Allroundwissen, Kenntnis der notwendigen Technologien, Ideenreichtum und die Fähigkeit diese Ideen zu kommunizieren.

2. *Projektdefinition*: Detaillierte Beschreibung und technische Gliederung des Projekts. Baugruppen des Projekts werden genau beschrieben und deren Entwicklung geplant. Des weiteren wird untersucht, ob bereits am Markt befindliche Komponenten verwendet werden können. Das Ergebnis dieser Phase ist ein Projektplan mit abgeschlossener Spezifikation des Projekts, ein verbindlicher Kosten- und Zeitplan und ein ausgereiftes Managementkonzepte. Qualifikation der Mitarbeiter: Es handelt sich hierbei um jene Mitarbeiter, die schon in der Konzeptformulierungsphase aktiv waren. Zusätzlich werden Mitarbeiter aus der Forschungs- und Entwicklungsphase hinzugezogen.

3. *Forschung und Entwicklung (FuE)*: Erstellung des Produkts zu den in den vorangegangen Phasen definierten Randbedingungen (Spezifikation, Kosten und Zeit).

4. *Produktion und Beschaffung*: Fertigungs-, Integrations- und Testarbeiten zu den vorgegebenen Randbedingungen. Diese Phase endet mit der Ablieferung des Produkts zu den vereinbarten Konditionen.

Es ist zwischen dem Lebenszyklus eines Projekts und dem eines Produkts zu unterscheiden: Ein Produktzyklus besitzt zusätzlich die Phasen Betrieb/Wartung und Aussonderung (außer Dienst stellen). Diese Phasen werden häufig vom Käufer eines Produkts übernommen. Generell gibt es einen Unterschied zwischen den Phasenmodellen von Firmen, die rein auf die Entwicklung von Produkten fokusiert sind und Firmen, die auch die Wartung und den Betrieb eines Produkts übernehmen. Im letzten Fall gehören die Phasen Wartung/Betrieb und Aussonderung ebenfalls zu den Projektphasen (diese 3 Phasen werden zusammengefasst auch als Folgephase bezeichnet). Vorkehrungen für die Folgephasen müssen in jedem Fall getroffen werden (z.B. Betriebs- und Wartungspläne, Werkzeuge zum Betrieb/Wartung).

Die Kosten für Konzeptformulierung und Projektdefinition sind im Allgemeinen viel niedriger als für die nachfolgenden Phasen. B. Madauss beziffert die Kostenverhältnisse wie in Tabelle 2.1 aufgelistet, bezogen auf die Kosten der Forschung und Entwicklung:

A - Konzeptformulierung	<= 0.03 * C
B - Projektdefinition	<= 0.1 * C
C - Forschung und Entwicklung (FuE)	1
D - Produktion u. Beschaffung	bis zu 3 * C
E - Betrieb u. Wartung / Aussonderung	bis zu 6 * C

Tabelle 2.1: Relative Kosten der einzelnen Projektphasen (Quelle [1])

Diese Verhältnisse sind natürlich abhängig von der Art des Projektes. Zu berücksichtigen ist weiters, dass Madauss auf den Bereich Luft und Raumfahrt spezialisiert ist.

Softwareprojekte können grob in folgende Phasen eingeteilt werden:

1. Konzeptformulierung
2. Softwaredefinition
3. Entwurf und Implementierung
4. Softwarevalidierung
5. Abnahme
6. Wartung und Weiterentwicklung

Für Softwareprojekte ergibt sich ein etwas anderes Kostenbild, wie Tabelle 2.2 zeigt.

Projektdefinition	ca. 15%
Entwurf (Planung)	ca. 25%
Entwicklung	ca. 20%
Systemintegration u. Test	ca. 40%

Tabelle 2.2: Relative Kosten der einzelnen Projektphasen (Quelle [1])

Überraschend ist hierbei der hohe Anteil an Systemintegration und Test im Vergleich zur Entwicklung. Dieser Prozentsatz kann bei sehr kritischen Anwendungen durchaus 50% oder mehr betragen. Diese Zahlen hängen natürlich auch vom verwendeten Phasenmodell ab (siehe Abschnitt 2.3). Für die Kosten der Weiterentwicklung/Wartung von Software habe ich leider keine sicheren Zahlen finden können. Die Problematik dabei ist, dass viele Produzenten die Weiterentwicklung als ein neues Produkt betrachten, die Weiterentwicklungskosten also nicht als solche ausgewiesen werden. Die Weiterentwicklungskosten von Software, die viele Jahre im Einsatz ist, kann jedoch das Drei- oder Vierfache der Entwicklungskosten betragen.

Auch wenn die oben genannten Zahlen von Projekt zu Projekt divergieren, kann man sehen, dass die Kosten der ersten Phasen deutlich niedriger sind als die Kosten späterer Phasen und dass das Risiko zur Freigabe der Konzept u. Definitionsphase deutlich geringer ist als die Freigabe der Folgephasen.

Das Ergebnis einer jeden Projektphase sind klar definierte Produkte. Als Produkt sind in diesem Zusammenhang auch Dokumente zu verstehen. Zu diesen Produkten gehört auch der Abschlussbericht einer jeden Phase, der Einsicht in eventuelle Probleme und Risiken gibt. Aufgrund dieses Berichts kann die Projektleitung frühzeitig auf Probleme reagieren oder das Projekt abbrechen. Abbildung 2.2 zeigt, wie sich die Kosten einer Fehlerbeseitigung in Programmen in Abhängigkeit der Phase, in der der Fehler entdeckt wird, verhalten. Die Bezugsphase ist die Spezifikationsphase: Die Korrektur eines Fehlers in einem bereits eingesetzten Programm kann 40-1000 Mal mehr kosten, als die Korrektur desselben Fehlers in der Spezifikationsphase.

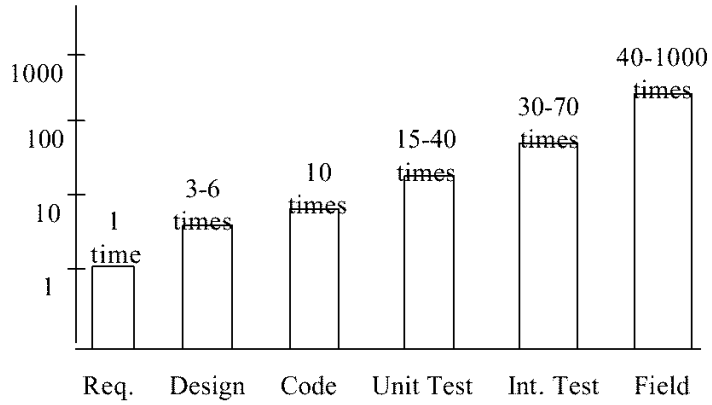

Abbildung 2.2: Relative Fehlerbehebungskosten nach Boehm

2.2 Milestones

Für jede Projektphase wird ein Zeitplan festgelegt, dessen Ende durch einen *Milestone* festgelegt wird. Der Milestoneplan ist Bestandteil eines jeden Projektplans. Als Beispiel seien die Milestones des US-Verteidigungsministeriums angeführt: In ihrer Vorschrift zur Bearbeitung großer Systeme sind folgende Milestones beschrieben:

- *Meilenstein 0*: Projektinitiierung - Beginn Konzeptionsphase
  - Festlegung der Projekterfordernisse
  - Budgetfreigabe (Konzeptphase)
  - Entscheidung zur Durchführung der Konzeptphase
- *Meilenstein 1*: Freigabe der Definitionsphase - Beginn der Definitionsphase
  - Demonstration der untersuchten Konzepte (Konzeptauswahl)
  - Budgetfreigabe (Definitionsphase)
  - Entscheidung zur Durchführung der Definitionsphase

- *Meilenstein 2*: Freigabe der Entwicklungsphase
  - Abschluss der Definitionsphase
  - Budgetfreigabe (Entwicklungsphase)
  - Entscheidung zur Durchführung der Entwicklungsphase
- *Meilenstein 3*: Freigabe der Folgephasen
  - Abschluss der Entwicklungsphase
  - Budgetfreigabe (Folgephasen)
  - Entscheidung zur Durchführung der Folgephasen.

Selbstverständlich können in der Projektabwicklung engere Milestones gesetzt werden. Zu beachten ist, dass die Voraussetzungen um einen Milestone zu erreichen in irgendeiner Form messbar sind oder mit hinreichender Sicherheit überprüft werden können. So ist zum Beispiel die Voraussetzung "Programm zu 90% fertig" ungeeignet, da man diese nur schätzen kann. Mögliche Zustände sind z.b "Anforderungsdokument liegt vor" (durch ein Review überprüfbar) oder "Komponente xyz hat Test bestanden" (durch Auswertung des Testprotokolls überprüfbar).

Abbildung 2.3 zeigt das Phasenkonzept in der Raumfahrt mit den Hauptmeilensteinen.

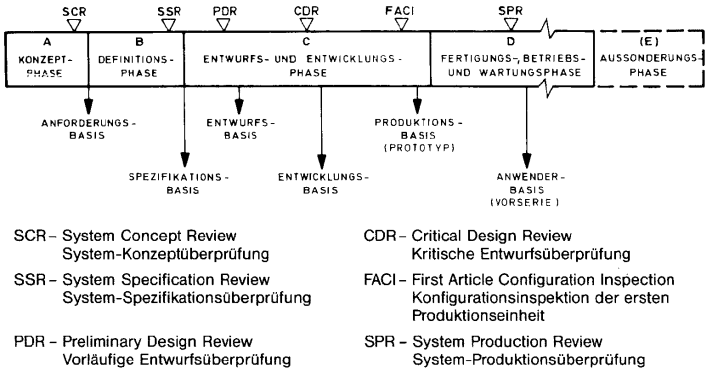

Abbildung 2.3: Phasenkonzept der Raumfahrt

2.2.1 Fast Tracking

Wird die nächste Phase im Projekt vor dem Abschluss der gegenwärtigen Phase in Angriff genommen, so spricht man von *Fast Tracking*. Diese Phasenüberschneidung muss sorgfältig geplant werden, da im Allgemeinen Änderungen in der gegenwärtigen Phase die nachfolgenden Phasen betreffen. Typischerweise werden Teile der Beschaffungsphase vorgezogen, um Güter mit langen Lieferzeiten zu ordern. Schlecht, wenn diese Güter dann doch nicht gebraucht werden.

2.3 Vorgehensmodelle in der Softwareentwicklung

Mittlerweile gibt es eine wahre Flut von Phasenmodellen für die Softwareentwicklung. Die hier dargestellten zeichnen sich durch ihre Einfachheit und große Bekanntheit aus. Viele Software Firmen haben ihr eigenes Vorgehensmodell entwickelt.

Eines der namentlich weniger bekannten, aber von jedem Softwareprogrammierer schon einmal verwendete Modell, und damit vermutlich das am weitesten verbreitete, ist das sogenannte Code-and-Fix Modell: Es besteht aus 2 Phasen:

1. Schreibe Code
2. Behebe Fehler im Code

Die Folgen dieser Vorgehensweise sind leicht absehbar:

a Nach einer gewissen Anzahl von Fixes (Bug oder Anforderungen) wird der Code unübersichtlich und sehr schwer wartbar.

b Ohne Anforderungsphase geht die Software meistens an den Anforderung der User vorbei (=>teure Neuentwicklung).

c Der Code ist schwer auszubessern, da keine Testfälle spezifiziert wurden.

Die nachfolgenden Modelle berücksichtigen diese Schwächen, jedes auf seine Weise.

2.3.1 Das Wasserfall-Modell

Das Wasserfall-Modell ist das erste veröffentlichte Phasenmodell zur Softwareentwicklung (Royce, 1970). Abgeleitet wurde es von Phasenmodellen anderer Ingenieursdisziplinen wie man leicht durch einen Vergleich mit dem in Abschnitt 2.1 beschriebenen Modell von Madauss erkennt.

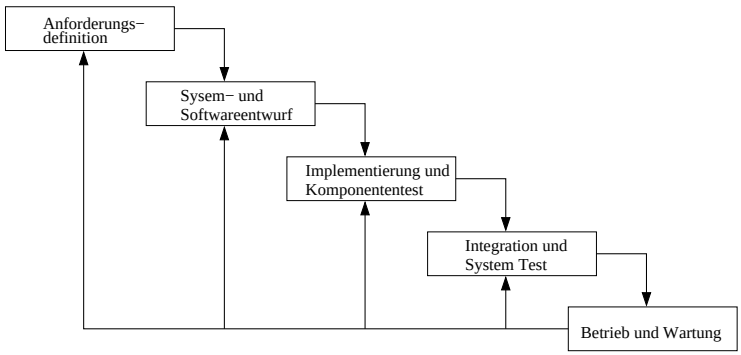

Abbildung 2.4: Wasserfallmodell

Die wichtigsten Phasen dieses Modells sind:

1. *Analyse und Definition der Anforderungen*: Diese Phase wird häufig in 2 Phasen aufgeteilt. In der 1. Phase werden die Kundenanforderungen ermittelt (User Requirements). Es wird das Projekt in "kundengerechter" Sprache und unter Einbeziehung des Kunden spezifiziert. In der 2. Phase werden die Anforderungen an die Software (Software Requirements) aus technischer Sicht beschrieben. Das Ergebnis dieser Phase ist die Projektspezifikation (Systemspezifikation) bestehend aus einem Anforderungsdokument (engl. User Requirements Dokument, URD) und gegebenenfalls einem Software Requirements Dokument (SRD).

2. *System und Software Entwurf*: Die Anforderungen an die Hard- und Softwaresysteme werden gegebenfalls ermittelt. Als nächstes wird eine allgemeine Systemarchitektur festgelegt - also das System in Module zerlegt. Die Anforderungen an die Module und deren Interaktion werden definiert (Architectural Design). Danach werden die einzelnen Module entworfen (Detailed Design). Folgende Dokumente werden produziert: Das Architectural Design Document (ADD) und das Detailed Design Document (DDD).

3. *Implementierung und Komponententest*: Die einzelnen Module der Software werden entworfen und jedes für sich getestet. Wichtig: Es darf nicht nur getestet werden, ob die Module keine Fehler produzieren. Es muss auch sichergestellt werden, dass die Spezifikation erfüllt wird. Die Testfälle müssen dementsprechend spezifiziert werden.

4. *Integration und Systemtest*: In dieser Phase werden die einzelnen Module zusammengesetzt (integriert) und das System wird als Ganzes getestet. War der Test erfolgreich (das heisst fehlerlose Zusammenarbeit der Module und Einhaltung der Spezifikation), wird das System dem Kunden übergeben (das über diese Phase Geschriebene stellt durchaus den Idealfall dar, die Praxis sieht häufig anders aus).

5. *Betrieb und Wartung*: Das System wird installiert und zum Gebrauch freigegeben (bei Standard Software entfällt dieser Schritt häufig. Zur Wartung gehören die Korrektur von nicht entdeckten Fehlern, Verbesserung des System und Erweiterung des Systems, falls neue Anforderungen bekannt/gefordert werden.

Nach jeder Phase werden die produzierten Dokumente einem Review unterzogen. Der Ausgang des Reviews entscheidet, ob mit der nächsten Phase begonnen werden kann oder ob Abänderungen in dieser und eventuell auch in früheren Phasen stattfinden müssen. Es sei noch angemerkt, dass natürlich Fehler/Abänderungen (Spezifikation, Design, Code) nicht nur während des Reviews, sondern zu jeder Zeit im Entwicklungsprozess entdeckt werden können. Wie mit derartigen Fehlern umgegangen wird (d.h. wer entscheidet, ob und wann die Abänderung gemacht wird) muss im Managementkonzept (also im Projektplan) festgelegt sein. Das V-Modell 97 ([11]) gibt hier genaue Richtlinien vor. In der Praxis sind einige Iterationen notwendig, bevor das System in Betrieb gehen kann. Hier sei auch noch das Fast Tracking (siehe Abschnitt 2.2.1) erwähnt: Gehen wir davon aus, dass das System in viele Softwaremodule zerlegbar ist und an jedem Modul eine gewisse Anzahl an Mitarbeitern arbeitet. Wenn nun

alle Module bis auf sehr wenige den Review zur Entwurfsphase bestanden haben, so wird man mit der Implementierung dieser fertig geplanten Module sofort beginnen, und nicht auf die wenigen unfertigen Module warten. Hier müssen die Faktoren Zeit und Kosten mit dem Risiko des Fast Trackings verglichen werden.

Beim Wasserfallmodell spricht man auch von einem ergebnisorientierten Phasenmodell, da die Phasen sequentiell durchlaufen werden und in jeder Phase ein Teil der zu liefernden Ergebnisse erarbeitet werden. Als Nachteile des Wasserfall-Modells gelten:

- Aufgrund der hohen Kosten bei der Herstellung und Abänderung von Dokumenten wird das Ergebnis gewisser Phasen nach wenigen kleineren Iterationen eingefroren (z.B. Spezifikation, ADD). Das verfrühte Einfrieren von Spezifikation könnte bedeuten, dass das Programm nicht das tun wird, was der Benutzer verlangt.

- Um große Iterationen zu vermeiden, müssen in frühen Phasen der Entwicklung verbindliche Annahmen gemacht werden. Das Modell ist also relativ unflexibel hinsichtlich neuer Anforderungen. Für einige Klassen von Software (z.B. Compiler, Protokolle) ändern sich die Anforderungen kaum. Für Software, die stark mit Personen interagiert, ändern sich die Anforderungen ständig.

Es gibt viele dem Wasserfall-Modell ähnliche Modelle, die sich aber meist nur in der Bezeichnung und der Anzahl der einzelnen Phasen unterscheiden. Das Wasserfall-Modell eignet sich besonders für die Entwicklung von Software, deren Anforderungen genau spezifiziert sind. Man sollte bedenken, dass Kunden selten genau wissen, was sie wollen, da ein Verständnis der Informatik nur in seltenen Fällen vorhanden ist (Nach dem Motto: “Ich kann dir jetzt nicht sagen, was ich will, aber ich weiß es, wenn ich es sehe”).

2.3.2 Das V-Modell

Im V-Modell werden den konstruktiven Aktivitäten prüfende Aktivitäten gegenübergestellt. Diese Zuordnung von Phasen kann bildhaft als V dargestellt werden (siehe Abbildung 2.5). Man beginnt mit dem Systementwurf im Groben und verfeinert dann schrittweise (vom Systemdurchführbarkeitskonzept bis zum Moduleentwurf / Code) – dies stellt die linke Achse des V-Modells dar. Ist man an der Spitze des Vs angelangt, so validiert man sämtliche Phase, diesmal vom Detail (Einzeltest der Module) zum Ganzheitlichen (Betrieb).

Die zugrundeliegenden Gedanken der Organisation der Testphasen sind folgende:

- Fehler können am ehesten in der Abstraktionsstufe gefunden werden, in der sie begangen wurden. Es sollten auf den Phasen der rechten Seite nur diejenigen Fehler gesucht werden, die in den entsprechenden Phasen auf der linken Seite begangen wurden.

- Je länger die Verweilzeit eines Fehlers im System, desto aufwendiger seine Behebung (z.B. gleich nach der Implementierung eines Moduls auch dessen Test).

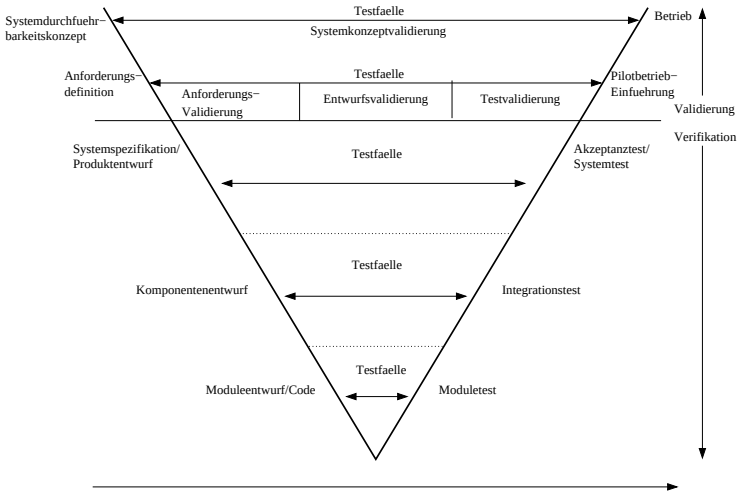

Abbildung 2.5: V-Modell

Die zu entwickelnden Testfälle umfassen die Anforderungsvalidierung, Entwurfsvalidierung und die Testvalidierung. Um das Risiko zu vermeiden, in den letzten Testphasen (z.B. Anwenderforderung) gravierende Fehler beheben zu müssen, kann zusätzlich die Technik des Prototyping verwendet werden (siehe Abschnitt 2.3.7). Das V-Modell könnte man durchaus als eine Ausprägung des Wasserfallmodells sehen.

2.3.3 Evolutionäre Entwicklung

Das evolutionäre Modell (auch Wachstumsmodell, inkrementelles Modell, Abbildung 2.6) steht ganz im Gegensatz zum Wasserfall-Modell. Das Prinzip ist folgendes: Ausgehend von einer groben Systembeschreibung wird die erste Version des Systems erstellt (mit den Phasen Spezifikation, Entwicklung, Validierung). Die erste Version des Programms stellt ein funktionsfähiges System dar, das vom Kunden eingesetzt werden kann. Die Rückmeldungen des Kunden verändern die Spezifikation, eine neue Version entsteht, die wiederum vom Kunden eingesetzt werden kann (auch wenn der Kunde keine Teillieferungen wünscht, kann dieses Modell eingesetzt werden, die Lieferung erfolgt dann intern). Die Teilprojekte sind in der Regel nicht zu umfangreich, es kann zu deren Entwicklung also durchaus das Wasserfall-Modell verwendet werden. Der große Unterschied zum Wasserfall-Modell besteht darin, dass das evolutionäre Modell von den Iterationen lebt, im Wasserfall-Modell hingegen Iterationen einen Fehlerfall darstellen.

Vorteile der evolutionären Entwicklung:

- Sie modellieren das natürliche Verhalten langlebiger Programme.
- Eine erste Version ist schnell fertig (steigert die Motivation der Entwickler, Auftraggeber erhält wichtige Teile schneller).
- Da die einzelnen Iterationen kurz gehalten werden sollten, fördert die evolutionäre Entwicklung die Kontrolle durch das Projektmanagement: Jede

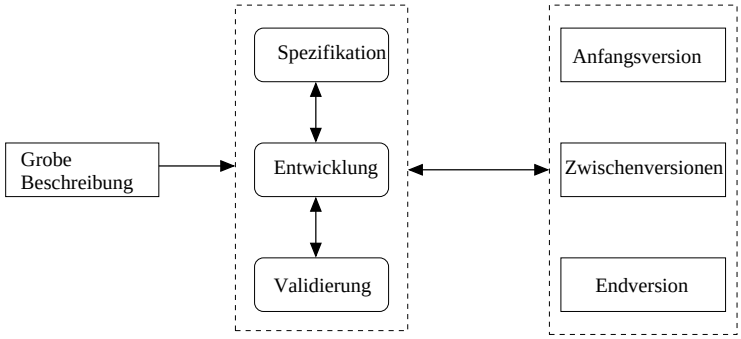

Abbildung 2.6: Evolutionäres Modell

Teillieferung stellt einen Meilenstein dar.

- Das System kann unter Umständen beim Auftraggeber schrittweise eingeführt werden.

Nachteile:

- Es besteht die Gefahr des Abgleitens ins Code-and-Fix Modell.
- Die Integration neuer Anforderungen kann die ursprüngliche Architektur der Software verwässern. Dies kann eine komplette Neuentwicklung der Software zur Folge haben (flexibler Erstentwurf von Vorteil).

Die kontinuierliche Integration von neuen Programmteilen, hervorgerufen durch neue Anforderungen, erfordert genauso wie die anderen Modelle ein gut durchdachtes Verifizierungsmodell. Es muss sichergestellt werden, dass durch die Integration von neuen Programmteilen bereits integrierte Programmteile nicht beeinträchtigt werden. Es ist ein gut funktionierendes Konfigurationsmanagement erforderlich, um auf sämtliche Programmversionen zugreifen zu können, da unter Umständen mehrere Versionen gewartet werden müssen. Sind die zu integrierenden Änderungen nicht zu umfangreich, kann bei guter Organisation auch ein *Daily Build* erzeugt werden. Der Vorteil eines Daily Builds ist der extrem kurze Rückkoppelungszyklus, der frühe Fehlerkorrektur ermöglicht. Der Nachteil eines Daily Builds ist das Unterbleiben von größeren Änderungen, da sie sich als Daily Build nicht realisieren lassen. Bleiben aber diese notwendigen Änderungen aus, so kann eine komplette Neuentwicklung der Software die Folge sein.

Die Entwicklung nach dem Wachstumsmodell kann empfohlen werden, wenn die Entwicklungszeit sehr lange dauert (mehrere Jahre), nicht die gesamte Funktionalität gleichzeitig zur Verfügung stehen muss oder wenn man die Anforderungen nicht genau kennt.

2.3.4 Formale Systementwicklung

Die formale Systementwicklung ist ähnlich dem Wasserfall-Modell, nur dass die Anwenderforderungen in eine formale Spezifikation überführt werden. Der Entwicklungsprozess besteht in einer Umwandlung der formalen Spezifikation in ein

ausführbares Programm. Die Korrektheit kann durch diesen formalen Ansatz bewiesen werden. Es gibt keine Fehlertests (wohl aber Integrations- und Systemtests). Verwendet wird die formale Entwicklung bei sehr kritischen Systemen (z.B. Steuerung von Eisenbahnsignalanlagen). Die formale Systementwicklung erfordert hoch qualifiziertes Personal, das mit den formalen Methoden der Spezifikation vertraut ist.

2.3.5 Agile Softwareentwicklung

2.3.6 Das Spiralmodell

Das Spiralmodell Abbildung 2.7 wurde von Boehm 1988 als Reaktion auf die Problematik des Wasserfallmodells entwickelt.

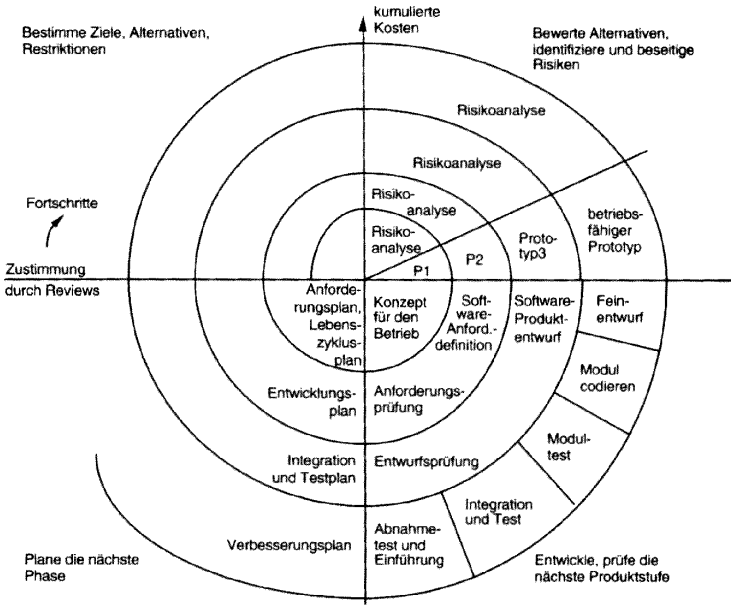

Abbildung 2.7: Spiralmodell

Die Spirale wird 4 Mal durchlaufen. Der 1. Durchlauf entspricht der Konzeptionsphase, der 2. Durchlauf der Definition des Projekts, der 3. Durchlauf der Entwurfsphase (Architectural Design), der 4. Durchlauf schließlich der Fein-Entwurfsphase (entspricht den letzten Phasen des Wasserfall-Modells). Es müssen allerdings nicht alle Phasen des Spiralmodells durchlaufen werden (hängt stark vom Resultat der Risikoanalyse ab).

Jeder Spiraldurchgang beginnt mit der Festlegung folgender Ziele für den aktuellen Durchlauf:

- Ziele (z.B. Performance, Funktionalität, Flexibilität).
- Alternativen (Design, Wiederverwendung, Einkauf)

- Einschränkungen hinsichtlich der Alternativen (Kosten, Zeit, Interface)

Als nächstes werden die Risiken bezüglich der Ziele und Einschränkungen analysiert und die Alternativen ausgewertet (z.B. Erstellung von Prototypen, Simulation, analytische Modellierung, Benchmarking).

Der nächste Schritt hängt nun von dem Ergebnis der Risikoanlayse ab: Können User-Interface Risiken ausgeschlossen werden und sind die User Anforderungen bekannt, kann man mit dem Wasserfall Modell fortfahren (SW-Anforderungen, Entwicklungsplan, Produktentwurf etc). Die Optionen Prototyping, Simulationen usw. können entfallen. Die Risikoanalyse bleibt aber auch hier fester Bestandteil des Prozesses. Sollten hingegen grobe Risiken hinsichtlich Kundenanforderungen bestehen, so fährt man mit der Spirale fort. Es kann auch z.B. mit einem evolutionären Modell fortgefahren werden, d.h. es werden solange Prototypen entwickelt, bis die Kundenanforderungen klar sind. Die Definition der Softwarerequirements wird so gesehen zwar erfüllt, aber nicht wie im Spiralmodell angegeben, ausgeführt (nämlich bevor der 2. Prototyp entworfen wird).

Das Spiralmodell ist stark Risiko-orientiert, das Ergebnis der Risikoanalyse bestimmt die weitere Phasenplanung, es gibt also keine festen Phasen wie z.B. beim Wasserfall-Modell.

2.3.7 Prototypen

Ein Prototyp ist ein Teil Software oder Hardware, der kritische oder schwer planbare Teile eines zu entwickelnden Systems vorab realisiert. Prototypen stellen kein Vorgehensmodell dar, sie können aber einige Phasen von Vorgehensmodellen unterstützen, insbesondere die Spezifikations- und Entwurfsphase. Aus den Erfahrungen der Entwicklung des Prototypen können dann Anforderungen oder Designentscheidungen konkretisiert werden. Prototypen können zusätzlich folgende Vorteile bringen:

- Durch das Vorführen von Prototypen können Missverständnisse zwischen Entwicklern und Benutzern aufgezeigt werden.
- Prototypen können für Schulungszwecke verwendet werden.
- Prototypen können dem Management bei der Aquisition von Aufträgen hilfreich sein.
- Der Prototyp kann als Spezifikationsgrundlage des zu entwickelnden Systems dienen.

Prototypen erhöhen zwar die Entwicklungskosten des Systems in den ersten Phasen, können aber durchaus zur Verringerung der Gesamtentwicklungskosten führen, da man mit Prototypen Fehler in der Spezifikation oder im Design schnell erkennt. Der Entwurfsprozess eines Prototypen sieht im Allgemeinen folgendermaßen aus:

1. Ziel des Prototypen festlegen (z.B. Konkretisieren der Anwenderforderungen).
2. Die Funktionalität des Prototypen wird festgelegt.

3. Entwicklung des Prototypen.

4. Auswertung des Prototypen (z.B. Welche neuen Anforderungen haben sich ergeben? Ist das System performant? Funktioniert das Design?).

Es gibt grundsätzlich 2 Arten von Prototypen: *Wegwerf-Prototypen* (Throw-Away Prototyping) und *evolutionäre Prototypen*. Das Vorgehensmodell für den evolutionären Prototypen ist die evolutionäre Entwicklung (siehe Abschnitt 2.3.3). Das evolutionäre Prototyping wird häufig in E-Commerce und Web-Site Entwicklung verwendet. Evolutionäre Prototypen müssen im Vergleich zu Throw-Away Prototypen sorgfältig geplant und dokumentiert werden, da sie immer weiter entwickelt werden und auch als Produkt verkauft werden. Die 3 Hauptprobleme des evolutionären Prototypings sind:

1. Managementprobleme: In der Regel kommt für große Systeme ein Vorgehensmodell zum Einsatz, das gut dokumentierte Teilergebnisse liefert. Die Dokumentation wird bei evolutionärem Prototyping gern vernachlässigt, da neue Prototypen sehr schnell entstehen.

2. Wartungsprobleme: Ständige Veränderungen führen zu einer Verwässerung des architektonischen Konzepts des Prototypen. Die Wartung und Erweiterung wird zunehmend schwieriger. Das System ist oft nur für die direkt beteiligten Entwickler durchschaubar.

3. Vertragliche Probleme: Verträge entstehen meist auf Basis von genauen Spezifikationen, die für beide Seiten (Auftraggeber, Auftragnehmer) bindend und einklagbar sind. Das Fehlen einer derartigen Spezifikation erschwert die Vertragsaushandlung. Ein Auftraggeber bezahlt ungern die Entwickler auf Tagesbasis (Budgetüberschreitung, Verschleppung des Projekts). Umgekehrt wird der Auftragnehmer ungern eine Pauschale akzeptieren, wenn keine genaue Spezifikation vorliegt.

Throw-Away Prototypen werden vorwiegend benutzt um Anwenderforderungen und Designs zu überprüfen. Diese Art von Prototypen unterliegen nicht den Qualitätsanforderungen des Projekts (z.B. weniger Dokumentation oder kein Testfallentwurf). Der Prototyp kann sich auf einen kleinen Teil des Projekts beziehen. Ungeeignet ist diese Art von Prototyp zur Feststellung von Zuverlässigkeit, Stabilität und Sicherheit der zu entwickelnden Software. Typische Wegwerfprototypen sind z.B. Demonstratoren für stark interaktive Systeme, die den Fokus auf das User Interface legen und die Funktionalität vernachlässigen oder gar nicht implementieren. Eine andere Art von Throw-Away Prototypen sind experimentelle Prototypen. Sie werden verwendet, um ein Design oder alternative Designs zu überprüfen und zu bewerten. Eine Problematik von Throw-Away Prototypen besteht in der Forderung des Managements, den zur einmaligen Verwendung bestimmten Prototypen als Produkt zu verkaufen (Hauptursache dafür: Terminprobleme).

2.3.8 Bemerkungen zu den Modellen

Bei den hier gezeigten Vorgehensmodellen handelt es sich wie an deren Bezeichnung erkennbar um Modelle. Modelle haben die Eigenschaft die Wirklichkeit

abstrakt und vereinfacht darzustellen, indem sie sich auf einen bestimmten (vermeintlich) wesentlichen Teil des Problem beziehen. Die Modelle sollten eine grobe Anleitung für das Projektteam schaffen, die dann projektspezifisch im Projektplan genauer definiert wird. Die Anzahl der Phasen und insbesondere die Planung einer jeden Phase ist unbedingt der Größe des Projekts anzupassen, um unnötigen Verwaltungsaufwand zu vermeiden. So zum Beispiel wird sich die Entwicklung eines Middleware Systems durchaus von der Entwicklung einer speziellen Anwendung unterscheiden. Zu beachten ist auch die bevorzugte Arbeitsweise des Teams. Ich vertrete die Ansicht, dass man bei der Planung des Vorgehensmodells nur die Aktivitäten, Werkzeuge und Dokumente vorsehen soll, die als unbedingt notwendig erachtet werden. Sollte sich später herausstellen, dass ein Werkzeug fehlt, kann man es immer noch integrieren. Umgekehrt alle unter Umständen benötigte Werkzeuge im Voraus zu beschaffen und zu integrieren kann viel Zeit und Geld vernichten, wenn sich herausstellt, dass man diese gar nicht verwendet. Ähnlich verhält es sich mit dem Verfassen von Dokumenten. Es sollte nur das Notwendigste dokumentiert werden. Je mehr Dokumente verfaßt werden, umso schwieriger ist die Gewährleistung deren Konsistenz. Hinzu kommt, das es sich bei vielen Softwareentwicklern nicht unbedingt um brillante Schriftsteller handelt, deren Dokumente häufig schwer verständlich sind. Damit die Dokumentation ihren Zweck erfüllt, müssen diese aber präzise und unmissverständlich formuliert sein. Für viele Softwareentwickler ist das Verfassen von Dokumenten geradezu eine Qual.

Jedes der vorgestellten Vorgehensmodelle hat projektabhängig seine Vor- und Nachteile. Ein beliebter Fehler (oder Zwang) ist allerdings, eine genaue Spezifikation und Planung erst gar nicht zu versuchen, da man möglichst schnell ein lauffähiges Produkt erzeugen will. Ein anderer Fehler ist es allerdings zu spezifizieren, was man noch gar nicht wissen kann, weil der Kunde es eben auch noch nicht weiß. Prototyping kann hier helfen und auch in den Vorgehensweisen eingesetzt werden, in denen es nicht explizit Teil des Vorgehens ist (z.B. Wasserfall- und V-Modell).

Ungenaue Anwenderforderungen können bei der Planung des Systems berücksichtigt werden, indem man die Software in Hinblick auf Erweiterung und Flexibilität plant. Die Planungsphase verlängert sich dadurch, die insgesamte Projektlaufzeit kann sich dafür unter Umständen deutlich verkürzen. Der Nachteil einer längeren Planungsphase ist offensichtlich: Der Kunde “sieht” erst später Teile des Produkts, die erstellten Planungsdokumente sind dem Kunden nicht immer verständlich, der Verlauf der Entwicklung also nicht nachvollziehbar. Grundsätzlich sollten die Konsequenzen des geplanten Vorgehens (z.B. lange Planungsphase) mit dem Kunden besprochen und wenn nötig ausverhandelt werden. Des Weiteren sollten lange Planungsphasen nur dann in Kauf genommen werden, wenn die Anforderungen feststehen.

Kapitel 3

Projekt-Planung

3.1 Management Prozesse

Die zentralen Aufgaben des Projektmanagers sind die Initiierung, Planung, Kontrolle und Ausführung des Projekts. Das Ergebnis der Planung eines Projekts ist der Projektplan. Ohne einen Projektplan ist die Kontrolle und Ausführung des Projektablaufs kaum möglich, da aus dem Projektplan ersichtlich ist, was kontrolliert werden muss und welche Aktivitäten ausgeführt werden müssen. Abbildung 3.1 zeigt das Zusammenspiel dieser Management Prozesse. Zu Beginn steht die Projektinitiierung, die die Freigabe zur Planung des Projekts zur Folge hat. Nach der Planung erfolgt die Ausführung der im Projektplan definierten Aktivitäten. Es erfolgt schließlich, wie im Projektplan definiert, die Kontrolle der ausgeführten Aktivität. Die Kontrolle kann sich wieder auf die Planung und die Ausführung des Projekts auswirken. So zu Beispiel kann die Planung des Projekts beeinflusst werden, wenn gewisse Aktivitäten nicht zu den im Projektplan vorgesehenen Kosten abgeschlossen werden können. In diesem Fall muss der Projektplan modifiziert werden. Die Kontrolle kann auch ergeben, dass eine Aktivität (z.B. Entwicklung eines Moduls) nicht die im Projektplan geforderte Funktionalität liefert. In diesem Fall ergibt die Kontrolle, das diese Aktivität noch nicht abgeschlossen werden kann, beeinflusst also dessen Ausführung.

Wichtig hierbei ist, dass Kontrolle und Planung keine einmaligen Aufgaben im Projektverlauf sind, sondern diesen bis zum Projektabschluss begleiten.

Der Hauptteil der Projektplanung passiert in der Projektdefinitionsphase (siehe Abschnitt 2.1). Abbildung 3.2 zeigt den Zusammenhang der Kernplanungsprozesse. Die Beziehungen zwischen diesen Prozessen und deren Reihenfolge treffen auf die meisten Projekte zu.

Zu Beginn des Planungsprozesses wird die Projektdefinition durchgeführt. Nach Abschluss dieser Aktivität findet die Kosten-, Zeit- und Risikoplanung statt. Bei der Risikoplanung handelt es sich nicht um die Identifikation der Risiken, sondern wie diese stattfinden soll (wann findet eine Risikoüberprüfung statt und wer ist dafür verantwortlich, wie werden Risiken dokumentiert, welche Tools werden verwendet etc.).

Teile der Kosten- und Zeitplanung können auch parallel ausgeführt werden. Die Zeitplanung beginnt mit einer Definition der zu planenden Aktivitäten, die Teil des Projekts sind (Grundlage dafür ist der Projektstrukturplan, dieser be-

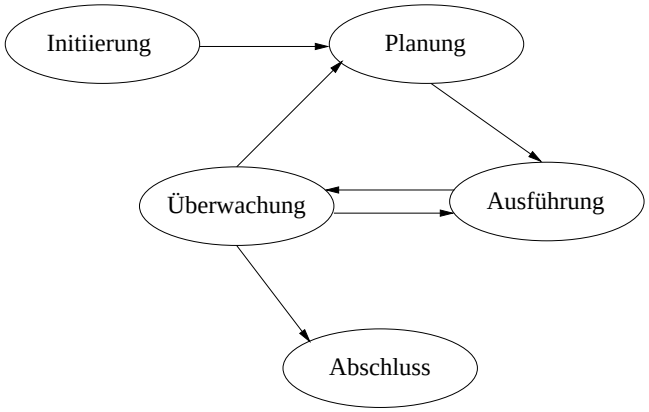

Abbildung 3.1: Zusammenspiel der Management Prozesse (Quelle [2])

stimmt die Unterteilung des Projekts in handliche Arbeitspakete, siehe auch Abschnitt 4.6). Danach wird abgeschätzt, wie viel Zeit jede dieser Aktivitäten benötigt und in welcher Reihenfolge diese ausgeführt werden (Netzplan Technik, Balkenplan...). Der Kostenplan beginnt mit einer Ressource-Planung (Bestimmung wie viele Mitarbeiter, Ausrüstung etc. benötigt werden) und nachfolgend werden die Kosten der Resourcen und Aktivitäten geschätzt (Grundlage ist wieder der Projektstrukturplan). Zu beachten ist, dass die Ressource-Planung unter Umständen die Zeitplanung (die Dauer der Aktivitäten) beeinflusst. Je mehr Mitarbeiter zur Verfügung stehen, umso schneller können gewisse Aktivitäten ausgeführt werden. Umgekehrt kann die Zeitplanung die Kosten beeinflussen (z.B. im Fall von Kreditfinanzierung wäre die Zinsbelastung zu beachten). Das Ergebnis der Kostenschätzung ist die Etataufstellung. Diese gibt nicht nur die gesamten Projektkosten wieder, sondern auch den Kostenverlauf über den gesamten Projektzyklus. Der Kosten- und Zeitplan sowie der Risikoplan werden in den Projektplan integriert.

Zusätzlich zu diesen Prozessen gehören in der Planungsphase noch:

- Die Planung der Qualitätssicherung (QS)
- Die Planung des Konfigurationsmanagements (KM)
- Risiko-Identifikation und Risikoanalyse, Planung der Reaktionen auf ein Risiko
- Organisatorische Planungen (z.B. benötigte Mitarbeiter, Rollenverteilung, Dokumentation)
- Planung der Vergabe/Beschaffung (z.B. wenn Teile des Projekts fremd vergeben werden)
- Kommunikationsplanung

Diese Aktivitäten sind zeitlich nicht so streng einzuordnen wie die Aktivitäten in Abbildung 3.2. So kann eine Risikoanalyse bei all den Kernprozessen stattfinden (z.B. technische, finanzielle, zeitliche Risikoanalyse). Die Risiken

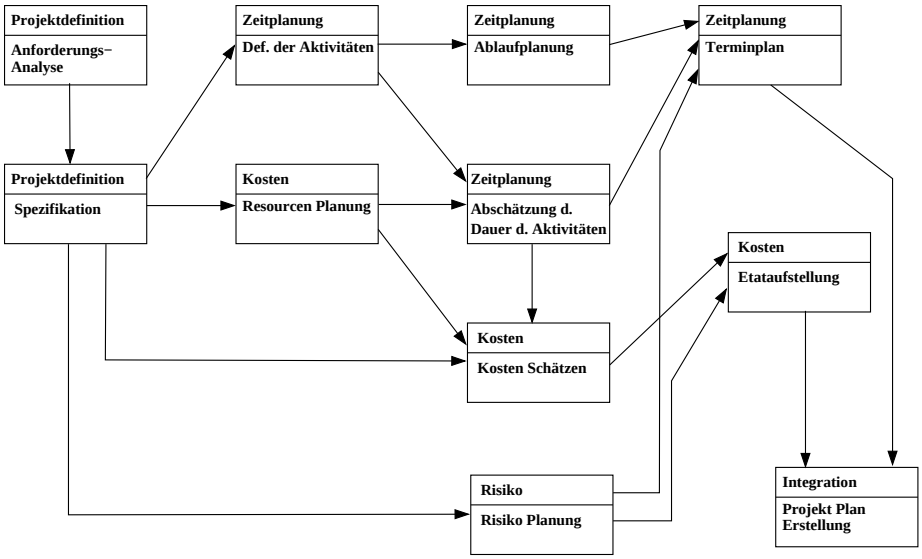

Abbildung 3.2: Beziehungen der Planungsprozesse (Quelle [2])

könnten zum Beispiel vor der Erstellung des Kosten- und Zeitplans oder des QS-Plans nicht bekannt gewesen sein.

Sämtliche in diesem Kapitel vorgestellten Prozesse sind iterativ, das heißt ihre Durchführung kann wiederholt werden müssen. So könnten sich im Laufe der Planung neue Anforderungen ergeben (z.B. aus der QS). Es könnten auch Anforderungen wegfallen (z.B. wegen zu hoher Kosten).

3.2 Der Projektplan

Nach einer groben Besprechung der Planungsaktivitäten in Abschnitt 3.1 sind die Teile des Projektplans identifiziert. Es gibt viele Möglichkeiten, einen Projektplan zu strukturieren. Die Genauigkeit des Projektplans sollte dessen Publikum angepasst sein. So kann unter Umständen der Projektplan, der an das Management der durchführenden Firma/Organisation verteilt wird, weniger Information enthalten als der Projektplan, der an den Auftraggeber des Projekts oder an dessen Entwickler verteilt wird. Es folgen nun mögliche Punkte, die der Projektplan abhandeln sollte:

- **Einleitung**
  - Projektziele: Kurze Beschreibung der Projektziele.
  - Projektbeschreibung: Kurze Beschreibung des Projekts.
  - Projektphasen: Kurze Beschreibung der Projektphasen.
  - Hauptmeilensteine: Wiedergabe der Hauptmeilensteine (welche Produkte beinhalten sie und wann werden sie erreicht).
  - Wenn erforderlich, Referenzen auf die untergeordneten Managementpläne (textuell, Baumstruktur). Untergeordnete Managementpläne

sind z.B. QS-Plan, KM-Plan. Bei kleineren Projekten können diese Dokumente auch direkt im Projektplan als eigene Kapitel eingebunden werden.

- Übersicht über die nachfolgenden Kapitel.

- **Projektorganisation** Mögliche Unterpunkte sind:
  - Aufbauorganisation: Hier folgt eine Beschreibung der Projektorganisation. Es wird erklärt, wie sich das Projektteam (aus welchen Stellen, Organisationen, Firmen) zusammensetzt.
  - Aufgaben und Verantwortlichkeiten: Verteilung der Aufgaben auf das Projektteam und der oben genannten Organisationen (z.B Projektleitung, KM-Verantwortlicher, QS-Verantwortlicher, Entwickler..).
  - Externe Schnittstellen: Definition der Kommunikation- und Verfahrensschnittstellen mit firmenexternen Stellen (Auftraggeber, Unterauftragnehmer, Partner, Gremien, Ausschüsse, Zulassungsstellen). Wichtig ist auch die Benennung eines Ansprechpartners der externen Stellen.
  - Berichtswesen: Kommunikation- und Verfahrensschnittstellen mit firmentinternen Stellen.
  - Beteiligung des Auftraggebers: Der Aufgabenbereich zwischen Auftraggeber und Auftragnehmer wird festgelegt (z.B. Beteiligung des Auftraggebers an QS-Aktivitäten).
  - Standards und Richtlinien: Z.B. Kodierungsstandards, Dokumentenmuster, Namenskonventionen... (kann auch ein Verweis auf entsprechende Dokumente sein).

- **Projektstrukturplan**: Der Projektstrukturplan sollte hier in angemessener Genauigkeit wiedergegeben werden. Die Genauigkeit sollte den Kontrollfunktionen der Zielgruppe entsprechen.

- **Projektphasen**: Beschreibung des Vorgehensmodells (auch die Aktivitäten, die vor und nach einer Phase durchgeführt werden müssen). Soweit bekannt, die Entwicklungsstrategie der einzelnen Teile des Projekts (siehe Projektstrukturplan), welche Teile des Produkts im Rahmen des Projekts entwickelt werden, welche zugekauft werden können bzw. von ähnlichen Projekten wiederverwendet werden können.

- **Auswahl von Methoden und Werkzeugen**: Sofern die Auswahl von bestimmten Methoden und Werkzeugen vorgeschrieben ist (z.B. durch die Qualitätssicherung) erfolgt hier die Auflistung und Zuordnung von Methoden und Werkzeugen zu den einzelnen Aktivitäten (z.B. Aktivität: Anforderungsanalyse Methode: Funktionale Dekomposition; Werkzeug: Rational RequisitePro).

- **Aufwands- und Terminplanung:**
  - Ablaufplan: Beschreibt den zeitlichen Ablauf (jedoch ohne Datumsangabe) der im Projektstrukturplan enthaltenen Module und deren Abhängigkeiten.

  - Aufwandsplan: Für alle Aktivitäten (auch für die Aktivität Projektmanagement) wird hier der Aufwand ermittelt. Hier sollte auch eine Gegenüberstellung vom Ist- und Soll-Zustand wiedergegeben werden.
  - Terminplan: Aus dem Ablaufplan und Aufwandsplan wird der Terminplan konstruiert. Auch hier sollte eine Gegenüberstellung von Ist- und Soll-Werten stattfinden.
  - Milestone Plan: Entscheidende Termine werden zu Milestones ernannt (Ist- und Soll- Werte).
  - Zu- und Auslieferтermine: In diesem Kapitel werden alle Termine gesondert festgehalten, die auch projektexterne Stellen betreffen wie zum Beispiel Auslieferung von Produkten an den Auftraggeber, Zulieferungen von Unterauftragnehmern, gemeinsame Aktivitäten mehrerer Auftragnehmer (z.B. Integrations/Systemtest).

- **Einsatzmittelplan**

  - Personal: Hier erfolgt die Zuordnung von Projektmitarbeitern zu den einzelnen Aktivitäten.
  - Resourcen: Es wird festgelegt, welche Resourcen wann und in welchem Umfang benötigt werden (in Abstimmung mit dem Termin- und Aufwandsplan).

- **Risikomanagement** : Risikomanagementplan, Risikoabschätzung, Maßnahmen zur Minderung und Vermeidung von Risiken.

- **QS-Plan**: Der QS-Plan enthält die für ein Projekt gültigen generellen Festlegungen bezüglich der Prävention und der Nachweisführung (Arbeitsweise, Hilfsmittel, Abläufe). Geplant werden konstruktive Maßnahmen zur Erreichung der Qualitätsziele, präventive Maßnahmen zur Vermeidung von Qualitätsrisiken und analytische Maßnahmen zum Nachweis der Erfüllung von Qualitätsforderungen.

- **Prüfplan**: Der Prüfplan definiert die Prüfgegenstände und die Aufgaben und Verantwortlichkeiten bei den Prüfungen, die zeitliche Planung sowie die für die Durchführung erforderlichen Ressourcen. Im Prüfplan ist festgelegt, welche Produkte und Aktivitäten in welchem Zustand wann, von wem und womit zu prüfen sind.

- **KM-Plan**: Die Regeln des Konfigurationsmanagements werden festgelegt (welche Produktbibliothek wird verwendet, wie werden die Produkte identifiziert). Weiters wird festgelegt, welche Produkte vom Konfigurationsmanagement verwaltet werden.

Der Umfang der einzelnen Pläne variiert natürlich je nach Projektgröße. Umfangreiche Teilpläne (KM-Plan, QS-Plan ...) werden häufig in eigene Dokumente ausgelagert, der Projektplan enthält dann entsprechende Verweise und gegebenfalls nur eine Zusammenfassung des jeweiligen Plans. Wesentlich ist, dass der Projektplan dem Leser eine möglichst schnelle Entscheidungsgrundlage bietet. Unnötige Details sollten ausgelagert werden.

Der Projektplan unterliegt durchwegs Änderungen im Laufe des Projekts. Die erste Fassung des Plans enthält meist nur grobe Abschätzungen (Projektstrukturplan, Aufwandsabschätzung...). Diese groben Abschätzungen sind notwendig, um über die Durchführbarkeit des Projekts in den ersten Phasen (Konzeptionsphase) zu entscheiden. Der Projektplan wird dann mit zunehmendem Kenntnisstand verfeinert und korrigiert. Besonders in den Frühphasen des Projekts unterliegt der Projektplan häufigen Änderungen.

Es gibt viele Abhängigkeiten im Projektplan (siehe auch Abbildung 3.2). Eine Änderung im Projektstrukturplan hat auf Grund seiner vielseitigen Abhängigkeiten zu anderen Plänen (Ablauf-, Termin-, Aufwand- und Kostenplan, Ressourceplanung) unter Umständen weitreichende Konsequenzen für den gesamten Projektplan. Auch zeitliche Verschiebungen wirken sich auf mehrere Teile aus: Eine Terminüberschreitung eines Arbeitspaketes wird sich auf davon abhängige Arbeitspakete auswirken. Es ist also erforderlich, den Ablauf-, Termin- und Kostenplan entsprechend anzupassen. Hier ist für größere Projekte (viele Arbeitspakete im Projektstrukturplan) der Einsatz von Projektmanagement Software eine große Hilfe. Beginnend mit dem Projektstrukturplan sollte die Software die Entwicklung von allen davon abgeleiteten Plänen unterstützen.

Kapitel 4

Konzept- und Spezifikationsphase

In Kapitel 2 wurden die Phasen eines Projekts grob erklärt. Dieses Kapitel beschäftigt sich nun mit der Spezifikationsphase eines Projekts. Das Ergebnis dieser Phase ist ein Pflichtenheft/Anforderungsdokument (aus Sicht des Auftraggebers auch Lastenheft genannt), bestehend aus den Benutzeranforderungen (User Requirements) und Systemanforderungen (bezogen auf Software sind das die Software Requirements). Die Vorteile dieses Dokuments wurden teilweise schon in Kapitel 2 herausgearbeitet. Die Unverzichtbarkeit des Pflichtenhefts offenbart sich, wenn man dessen Zielgruppen (also die Leser des Dokuments) genauer betrachtet:

- **Auftraggeber/Kunde**: Dieser legt die Anforderungen an das Produkt fest. Aufgrund des Pflichtenheftes kann er überprüfen, ob alle seine Anforderungen berücksichtigt wurden. Desweiteren kann der Auftraggeber die im Projektplan angenommenen Zeit-, Kosten- und Risikoabschätzungen nachvollziehen. Für den Auftraggeber ist das Pflichtenheft ein wesentlicher Beitrag zu den Vertragsverhandlungen.
- **Management**: Das Management (Auftragnehmer) der Firma, die das Projekt leitet, benötigt das Pflichtenheft unter anderem, um die Sinnhaftigkeit/Notwendigkeit des Projekts zu überprüfen und den im Projektplan festgelegten Zeit- und Kostenplan nachzuvollziehen.
- **Projektleitung**: Diese benötigt das Pflichtenheft zur Planung des Systementwicklungsprozesses, zur Kontrolle des Projektablaufs und zur Angebotserstellung.
- **Benutzer**: Das Pflichtenheft erlaubt dem Benutzer des Systems sich eine ganzheitliche Sicht desselben anzueignen. Dies fördert das allgemeine Verständnis und daher auch die Bedienbarkeit des Systems. Im Allgemeinen ist der für den Benutzer relevante Teil des Pflichtenhefts Teil des Benutzerhandbuchs.
- **Systementwickler**: Sie benutzen die Anforderungen, um eine ganzheitliche Sicht von dem zu entwickelnden Produkt zu erhalten und um die Beziehungen der Bestandteile des Systems besser zu verstehen. Dies fördert

die Motivation, da sich die Entwickler eher mit dem System identifizieren. Ein Überblick über das System fördert auch die Sensibilität der Entwickler für Fehler in den zu entwickelnden Bestandteilen.

- **Systemtester**: Der Systemtester verwendet die Anforderungen, um die Funktionalität des Systems zu validieren.
- **Systemwarter**: Der Überblick über das System fördert das Verständnis desselben (siehe auch Systementwickler).

Die Probleme, die ein Projektteam zu bewältigen hat, sind oft kompliziert und daher ist ein gemeinsames Verständnis des Problems und eine exakte Beschreibung der gewünschten Eigenschaften nicht leicht zu erreichen. Was ein System leisten (und nicht leisten) soll, wird in den Benutzer- und Systemanforderungen beschrieben. Wichtig dabei ist, dass die Anforderungen nicht die Lösung des Problems vorgeben, sondern eben nur die Beschreibung des Problems. Dies ist vor allem dann von Vorteil, wenn eine Firma/Organisation eine Ausschreibung basierend auf einer Systemanforderung startet. So werden unter Umständen mehrere Anbieter mit verschiedenen Lösungsansätzen auf die Ausschreibung reagieren.

Die Anforderungsanalyse beschäftigt sich mit den Herausfinden und Analysieren von Anforderungen. Die Anforderungspezifikation verfeinert die Anforderungen (Softwarerequirements). Das Anforderungsmanagement beschreibt, wie bei Änderungen von Anforderungen vorgegangen werden muss. Abbildung 4.1 zeigt die Prozesse und ein mögliches Vorgehensmodell zur Spezifikation eines Projekts.

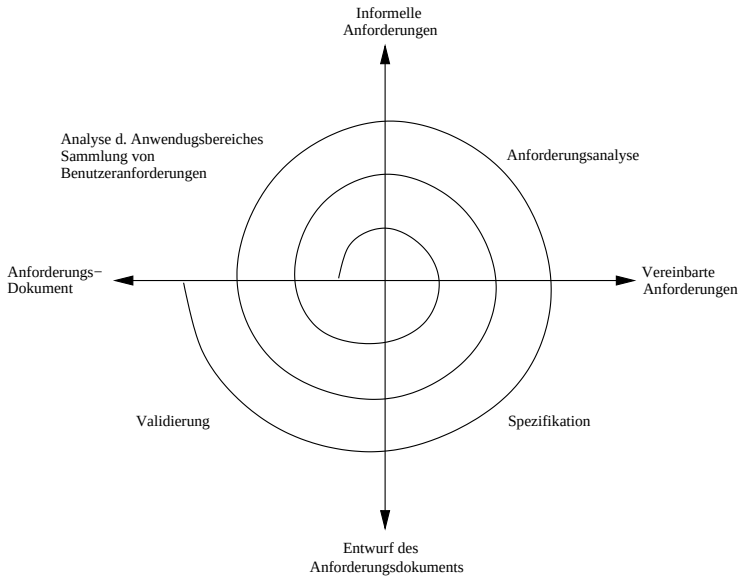

Abbildung 4.1: Spiralmodell der Spezifikationsprozesse

Ein Spiralumlauf beinhaltet folgende Prozesse:

1. Zuerst muss sich der Analytiker in das Anwendungsgebiet einarbeiten und Benutzeranforderungen sammeln. Wird zum Beispiel ein Programm zur Buchhaltung benötigt, muss der Analytiker die Vorgänge der Buchhaltung verstehen lernen. Gängige Techniken dazu sind die Befragung von Benutzern und Ethnografie (Beobachtung der Benutzer bei der Arbeit).

2. Es folgt die Gliederung und Analyse der Anforderungen. Auch eine Reihung der Anforderung nach ihrer Wichtigkeit sollte vorgenommen werden. Diese werden mit dem Auftraggeber und den zukünftigen Nutzern des Systems abgesprochen. Nun steht ein erster Satz an Benutzeranforderungen fest, der von der Auftraggeber- als auch von der Auftragnehmer-Seite akzeptiert ist.

3. Die Benutzeranforderungen werden in eine Systemspezifikation übergeführt. Es entsteht die erste Version des Anforderungsdokuments (auch Pflichten/Lasten Heft genannt).

4. Das Anforderungsdokument wird einer Überprüfung unterzogen (Kosten- und Aufwandsschätzung). Abhängig von der Validierung wird das Dokument akzeptiert oder es wird festgestellt, dass es abgeändert werden muss. Letzteres führt zu einem neuen Spiraldurchlauf.

Ein Nachteil dieses Modells ist die späte Validierung der Anforderungen (erst am Ende eines Spiraldurchlaufs). Empfehlenswert ist eine Validierung auch nach der Anforderungsanalyse, da Fehler in der Anforderungsanalyse unweigerlich auch Fehler in der Spezifikation zur Folge haben.

Der gerade beschriebenen Spezifikationsphase geht die Konzeptionsphase voraus. In der Konzeptionsphase wird eine Durchführbarkeitsstudie gemacht. Voraussetzung für eine derartige Studie ist eine grobe Systembeschreibung und wie das System eingesetzt werden soll. Folgende Fragen sollten von dieser Studie beantwortet werden:

- Folgt das Projekt der Unternehmensphilosophie (z.B. Markführerschaft)?
- Lohnt sich das Projekt wirtschaftlich für das Unternehmen (Return on Investement)?
- Sind neue Technologien erforderlich?
- Ist das Projekt mit den zur Verfügung stehenden Resourcen (Personal, Kosten, Zeit) durchführbar?
- Wie würde sich ein Scheitern des Projekts auf das Unternehmen auswirken?
- Was muss vom System unterstützt werden (z.B. Marktanalyse)?

Abhängig von dieser Studie wird das Management entscheiden, ob das Projekt durchgeführt wird (also mit der Spezifikationsphase begonnen wird).

4.1 Benutzer- und Systemanforderungen

Die Benutzeranforderungen beschreiben die Dienste, die das System leisten soll und die Randbedingungen, unter denen es betrieben wird. Diese Beschreibung sollte möglichst wenige Kenntnisse im Entwicklungsbereich erfordern. So zum Beispiel stehen beim Beschreiben eines Bankomatsystems in dieser Phase die Abläufe (was passiert wenn...), die Bedienung des Bankomaten und die Sicherheitsanforderungen ans System im Vordergrund. Daraus leiten sich Anforderungen an das System ab, die durch die Systemanforderungen beschrieben werden. Die Systemanforderungen (i.e. Systemspezifikation) und die Benutzeranforderungen dienen meist als Grundlage für Verträge zwischen Auftragnehmer und Auftraggeber.

Benutzeranforderungen sind die Beschreibung des Systems in natürlicher Sprache und gegebenenfalls Diagrammen, die allgemein verständlich sein sollten. Bei den Systemanforderungen handelt es sich um eine detaillierte, technische Beschreibung des Systems, abgeleitet von den Benutzeranforderungen. Die Systemanforderungen dienen den Entwicklern als Grundlage. Ein Beispiel aus [9], wie der Unterschied zwischen Benutzeranforderungen und Systemanforderungen aussehen könnte, zeigt Tabelle 4.1.

Benutzeranforderung	Systemanforderung
1. Die Software muss über Mittel zur Darstellung externer, von anderen Werkzeugen erzeugter Dateien verfügen und auf sie zugreifen können.	1. Der Benutzer sollte über Möglichkeiten zur Definition externer Dateitypen verfügen. 2. Jeder externe Dateityp kann eine damit verknüpfte Anwendung besitzen, mit der die Datei bearbeitet wird. 3. Jeder externe Dateityp kann als bestimmtes Symbol auf den Bildschirm des Anwenders dargestellt werden. 4. Es sollten Möglichkeiten zur Definition des Symbols für externe Dateitypen durch den Benutzer bereitgestellt werden. 5. Wenn ein Benutzer ein Symbol auswählt, das eine externe Datei repräsentiert, so soll die durch dieses Symbol dargestellte Datei mit der entsprechenden Anwendung geöffnet werden.

Tabelle 4.1: Von der Benutzeranforderung zur Systemanforderung

Dieses Beispiel ist mit Vorsicht zu genießen. Es zeigt, was aus einer Benutzeranforderung alles abgeleitet werden kann. So geht aus der Benutzeranforde-

rung nicht unbedingt hervor, dass es eine Möglichkeit zur Definition externer Dateitypen geben muss. So sollte auch bei der Systemspezifikation eine Rücksprache mit dem Auftraggeber stattfinden, da die Benutzeranforderungen im Allgemeinen verschieden interpretiert werden können.

4.1.1 Funktionale und nichtfunktionale Anforderungen

Funktionale Anforderungen beziehen sich auf eine spezielle Funktion (im Sinne von Aufgabe) des Systems wie z.B. die Anforderungen in Tabelle 4.1. Nichtfunktionale Anforderungen beziehen sich auf das System im Ganzen, diese Anforderungen wirken sich auf mehrere/alle Funktionen des Systems aus. Wird eine funktionale Anforderung nicht erfüllt, so lässt sich diese meist leichter korrigieren, da diese auf eine Funktion des Systems beschränkt ist. Nichtfunktionale Anforderungen sind meist nur mit viel Aufwand zu korrigieren. Ein Beispiel für eine nichtfunktionale Anforderung wäre eine Sicherheitsanforderung. Wird diese bei der Systemerstellung nicht beachtet, müssen sämtliche davon betroffene Funktionen umgearbeitet werden. Es folgt eine Art Checkliste für nichtfunktionale Anforderungen aus [9], wobei man bei manchen durchaus diskutieren könnte, ob diese nun funktional oder nichtfunktional sind (hängt auch vom zu entwickelnden System ab):

- Produktanforderungen:
  - Benutzbarkeitsanforderungen
  - Effizienzanforderungen (Leistungsanforderung, Speicherplatzanforderung)
  - Zuverlässigkeitsanforderungen
  - Portierbarkeitsanforderungen
- Unternehmensanforderungen:
  - Lieferanforderungen
  - Umsetzungsanforderungen
  - Vorgehensanforderungen
- Externe Anforderungen:
  - Kompatibilitätsanforderungen
  - Ethische Anforderungen
  - Rechtliche Anforderungen (Datenschutzanforderungen, Sicherheitsanforderungen)

4.2 Anforderungsanalyse

Die Anforderungsanalyse beschäftigt sich mit dem Sammeln von Anforderungen, dem Gruppieren von verwandten und abhängigen Anforderungen, mit der Zuweisung von Prioritäten zu den Anforderungen und mit der Auflösung von sich widersprechenden Anforderungen. Dieses Kapitel beschäftigt sich mit der Viewpoint-orientierten und Use-Case orientierten Bestimmung von Anforderungen. Des Weiteren wird eine diese Methoden begleitende Technik, die Ethnografie, erklärt.

4.2.1 Viewpoint-orientierte Bestimmung

Diese Methode versucht, die Anforderungen aus dem Blickpunkt der Projektbeteiligten (Project Stakeholders) zu ermitteln. Ein Beispiel für Projektbeteiligte an einem Bankomatsystem einer Bank sind (diese Liste erhebt keinen Anspruch auf Vollständigkeit):

1. Bankkunden, die den Bankomat verwenden
2. Vertreter anderer Banken, die Vereinbarungen mit der Bank geschlossen haben
3. Manager von Zweigstellen, die Verwaltungsinformation vom System erhalten
4. Schalterpersonal, die am täglichen Betrieb des Systems beteiligt sind (Kundenhilfe, Kundenbeschwerden)
5. Datenbankadministratoren für die Integration des Systems in die Kundendatenbank
6. Sicherheitsbeauftragte der Bank, die die Sicherheit des Systems garantieren müssen
7. Die Marketingabteilung der Bank, die die Vorteile des Systems an den Kunden weitergibt
8. Hard- und Softwareentwickler, die das System entwickeln
9. Kartenhersteller
10. Helpdesk der Bank (z.B. um Karten zu sperren)

Hat man die Viewpoints ermittelt (z.B. durch Brainstorming), versucht man die Anforderungen aus Sicht der Beteiligten zu ermitteln (persönliches Gespräch, Ethnografie). Die Anforderungen werden dann strukturiert und den verschiedenen Diensten/Funktionen zugeordnet (z.B. Bargeldabhebung, Kontoinformation), identische Anforderungen und widersprüchliche Anforderungen werden identifiziert. Im nächsten Schritt werden die Anforderungen und Dienste verfeinert (z.B. Bargeldabhebung: Karte einführen, Pin-Eingabe ...). Das Ergebnis sollten die Benutzeranforderungen sein. Man fährt dann wie in Abbildung 4.1 beschrieben mit dem Spezifikationsprozess fort.

4.2.2 Use-Cases

(FIXXME: Unterschied zu Benutzeranforderungen [by Karl Blümlinger (kbluem@iicm.edu)]) ⇐

Use-Cases dienen dazu, die Dienste/Funktionen eines Systems anhand der beteiligten Akteure zu beschreiben. Darüber hinaus weisen Use-Cases auf neue Anforderungen hin. Für Use-Cases gibt es verschiedene Notationen. In UML [10] ist eine graphische Notation für Use-Cases spezifiziert, zugeschnitten auf objektorientierte Entwicklung. Ein Beispiel eines Use-Cases basierend auf die UML Notation zeigt Abbildung 4.2. Es zeigt die agierenden Personen und die

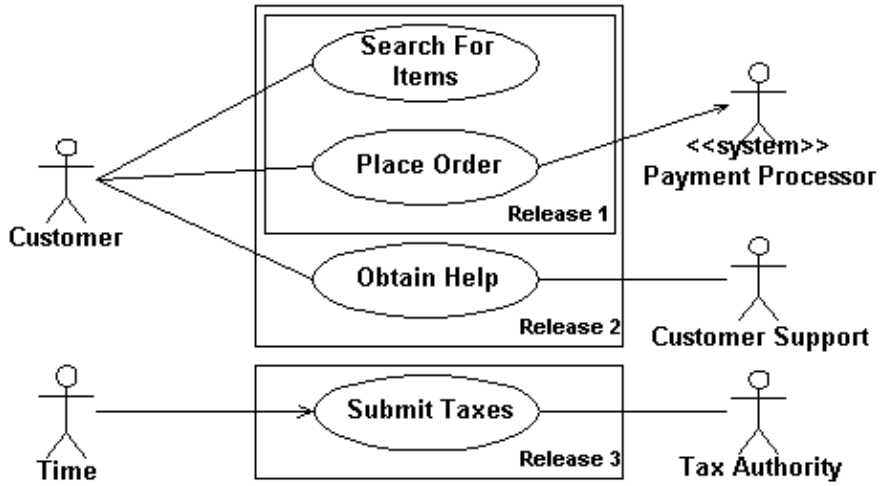

Abbildung 4.2: Beispiel eines Use-Cases in UML Notation

von diesen Personen in Anspruch genommenen Dienste. Die einzelnen Dienste werden dann wiederum mit UML Diagrammen verfeinert dargestellt.

Die UML Notation ist nur eine der Möglichkeiten, Use-Cases grafisch darzustellen. Abbildung 4.3 zeigt ein nicht standardisiertes Zustandsdiagramm eines Bankomaten.

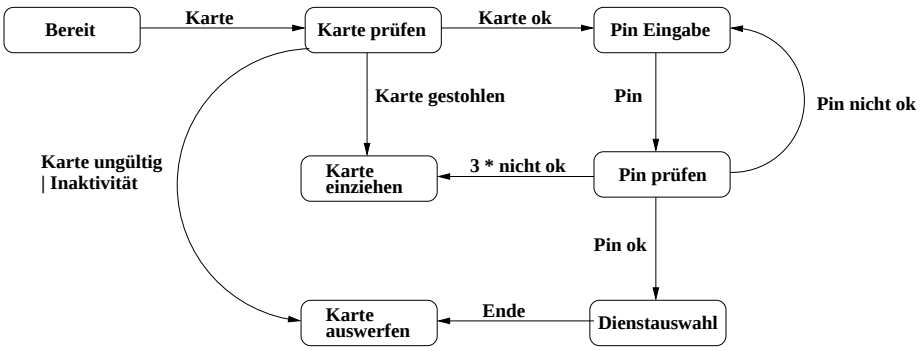

Abbildung 4.3: Zustandsdiagramm eines Bankomaten

Use-Cases können natürlich auch rein textuell beschrieben werden. Das Spezifikationsteam sollte entscheiden, welche Notation verwendet wird. Hier ein Beispiel, in dem beschrieben wird wie ein Kunde am Bankomat Geld abhebt:

1. Der Kunde sieht am Display eine Nachricht, die ihn zum Einführen einer Karte auffordert.
2. Der Kunde schiebt seine Smartcard in die Einzugsvorrichtung.
3. Ist die Karte ungültig, wird sie mit entsprechender Meldung wieder ausgegeben.

4. Nach 3 minütiger Inaktivität wird die Karte ausgeworfen.
5. Ist die Karte als gestohlen gemeldet, wird sie sofort eingezogen.
6. Der Kunde wird aufgefordert, seinen PIN-Code einzugeben.
7. Ist der eingegebene PIN-Code korrekt (siehe Use-Case Benutzerüberprüfung), fährt man mit Schritt 10 fort.
8. Im Falle einer falschen Eingabe wird der Kunde maximal 3 Mal aufgefordert, den korrekten PIN einzugeben (Schritte 6,7).
9. Wurde der PIN 3 Mal falsch eingegeben, wird die Karte eingezogen.
10. Nach erfolgreiche PIN-Eingabe folgt Dienstauswahl (siehe entsprechende Use-Cases).
11. Nach Beendigung des gewählten Dienstes wird die Karte wieder ausgegeben.

Es ist durchaus denkbar, grafische und textuelle Beschreibungen von Use-Cases in einem Anforderungsdokument zu verwenden, je nachdem welche Methode geeigneter erscheint. Nicht zu empfehlen ist allerdings eine Vermischung von verschiedenen grafischen Notationen (z.B. In Zustandsdiagrammen wird einmal ein Kreis, dann wieder ein Rechteck zur Kennzeichnung eines Zustandes verwendet).

4.2.3 Ethnografie

Ethnografie nennt man die Wissenschaft, die sich mit der Erforschung von gesellschaftlichen Vorgängen beschäftigt. Die wichtigste Methode der Ethnografie ist die Beobachtung: Ein Analytiker taucht dabei in eine Gesellschaft ein und beobachtet deren Verhalten und Strukturen. Software Programme sind in der Regel Systeme, die mit Menschen interagieren um deren Arbeit zu erleichtern. Für Anforderungsanalytiker kann die Ethnografie eine wichtige und gewinnbringende Methode sein, um Anforderungen an ein Softwaresystem zu erforschen. Die Ethnografie eignet sich besonders für folgende Arten von Anforderungen:

1. Anforderungen, die sich nicht aus einem vorgeschriebenen Arbeitsablauf ergeben.
2. Anforderungen, die sich aus der Zusammenarbeit mit anderen Menschen ergeben.
3. Anforderungen, die von den Benutzern für unwesentlich gehalten werden und daher nicht erwähnt werden.
4. Anforderungen, die dem Benutzer ganz einfach nicht in den Sinn kommen.

Ethnografie kann auch mit Prototypen kombiniert werden. Eine erste Anforderungsanalyse (z.B. Ethnografie, Use-Cases, Viewpoint Methode) fließt in den Prototypen ein. Die Benutzer werden dann bei der Verwendung des Prototypen beobachtet. Zu bedenken ist dabei, dass die Benutzer unter Umständen aufgrund der Anwesenheit eines Beobachters ihre Arbeitsweise ändern könnten.

4.3 Notationen für die Systemspezifikation

Systemanforderungen sollten ausdrücken, was das System tun soll und nicht wie das System zu implementieren ist (wobei natürlich das *Was* die Implementierung beeinflussen wird).

4.3.1 Beschreibung durch natürliche Sprache

Bei der Beschreibung in natürlicher Sprache wird zweckmäßiger Weise ein Standardformular/Standardlayout gewählt, aus dem zumindest der Name der spezifizierten Aufgabe, eine Beschreibung und Abhängigkeiten zu anderen Aufgaben hervorgeht. Bei der Wahl einer natürlichen Sprache für die Spezifikation sollte man folgendes beachten:

- Die Interpretation oder das Verständnis der Sprache hängt von den Lesern und Autoren der Spezifikation ab, da die natürliche Sprache mehrdeutig ist, ein und derselbe Begriff mehrere Bedeutungen haben kann. Eine klare Begriffsdefinition ist daher notwendig.
- Ähnliche oder voneinander abhängige Aufgaben sollten gruppiert werden. Dazu kann vorab aus den Benutzeranforderungen eine grobe Architektur des Systems erstellt werden.
- Die Spezifikation in natürlicher Sprache ist sehr flexibel, gleiche Sachverhalte können unterschiedlich ausgedrückt werden. Es sollten nicht mehrere Anforderungen als eine Anforderung ausgedrückt werden.

4.3.2 Beschreibung durch eine Program Description Language (PDL)

Eine PDL ist einer Programmiersprache sehr ähnlich, mit dem Unterschied, dass die einzelnen Funktionen nicht ausprogrammiert werden. Ein kleines Beispiel, das die Pineingabe eines Bankomaten spezifiziert:

```
pin = keyPad.readPin()
while ( !card.Pin.equals(pin) & attempts < 4 )
{
  pin = keyPad.readPin();
  attempts = attempts + 1;
}
if (!card.Pin.equals(pin))
  throw InvalidPinException();
```

Die Nachteile dieser Art von Spezifikation sind:

1. Die Spezifikation ist nur für Leute verständlich, die mit der verwendeten PDL vertraut sind.
2. Es besteht die Gefahr, dass man die Problemlösung vorwegnimmt.

4.3.3 Grafische Notation

Nur der Vollständigkeit wegen hier erwähnt. Beispiele für grafische Spezifikation sind Petri-Netze, UML, Zustandsdiagramme ...

4.3.4 Mathematische Notation

Dies sind Notationen, die auf mathematischen Konzepten aufbauen. Anforderungen in mathematischer Notation sind klar definiert und verhindern somit Streitpunkte zwischen Auftraggeber und Auftragnehmer. Ein Auftragnehmer, der mit dieser Art von Notation nicht vertraut ist, wird eine mathematische Notation unter Umständen nicht als Vertragsgrundlage akzeptieren.

4.4 Validierung von Anforderungen

Sinn der Anforderungsvalidierung ist es zu zeigen, dass die Anforderungen das System definieren, das der Kunde erwartet (Zur Erinnerung: Je früher ein Fehler erkannt wird, desto billiger ist seine Beseitigung). Die Validierung ist der Anforderungsanalyse nicht unähnlich, da beide Prozesse Probleme mit Anforderungen aufdecken und lösen. Der Unterschied zwischen den Prozessen liegt darin, dass die Analyse den Fokus auf das Herausfinden von Anforderungen legt, die Validierung auf die Überprüfung des gesamten Pflichtenhefts (Abschnitt 4.7).

Die Validierung sollte den Schwerpunkt auf folgende Art von Prüfungen legen ([9]):

- *Gültigkeits- und Konsistenzprüfungen*: Systeme haben verschiedene Benutzer mit unterschiedlichen und unter Umständen gegensätzlichen Anforderungen. Sämtliche Widersprüche in den Anforderungen müssen aufgelöst werden. Des weiteren sollte es keine unterschiedlichen Beschreibungen für dieselbe Systemfunktion geben.
- *Vollständigkeitsüberprüfungen*: Das Anforderungsdokument sollte alle durch den Anwender/Kunden erwarteten Funktionen und Einschränkungen beschreiben.
- *Realisierbarkeitsprüfungen*: Es muss überprüft werden, ob sich alle Anforderungen mit den vorhandenen Resourcen (Zeit, Geld) realisieren lassen.
- *Verifizierbarkeitsprüfungen*: Es sollte festgestellt werden, ob und wie gezeigt werden kann, dass die Anforderungen im fertigen System erfüllt wurden. Wichtig ist diese Art der Validierung vor allem bei der Abnahme des Systems durch den Kunden, um nachweisen zu können, dass die Anforderungen umgesetzt wurden.
- Überprüfung auf *Verständlichkeit*
- Überprüfung auf *Nachvollziehbarkeit*: Der Ursprung und Grund einer Anforderung sollte dokumentiert werden.

Techniken zur Validierung sind zum Beispiel Reviews, Prototypen und Testfallerzeugung.

4.5 Anforderungsmanagement

Die Anforderungen an ein System und in weiterer Folge die Spezifikation eines Systems unterliegen leider ständigen Änderungen, ein Anforderungsdokument ist naturgemäß unvollständig: Während der Systementwicklung bekommt man ein tieferes Verständnis für das System (also für die Probleme, zu deren Lösung das System entwickelt wird), neue Anforderungen werden bekannt. Ist das System in Betrieb, stellen sich manche Anforderungen als unzulänglich oder schlichtweg falsch heraus, einige Funktionen fehlen. Aus wirtschaftlichen Gründen wurden Anforderungen gestrichen, die sich letztlich als notwendig erwiesen haben. Im Laufe der Jahre kann sich das Umfeld, in dem die Software eingesetzt wird, verändern (technologisch, organisatorisch).

Das Anforderungsmanagement ist ein Prozess, der Anforderungsänderungen in bestehende Anforderungen einbindet. Ein Anforderungsmanagementplan legt fest, wie dieser Prozess durchzuführen ist. Bei der Planung werden vor allem zwei Bereiche definiert:

1. Formale Kriterien, um Änderungen leichter durchführen zu können. So wird zweckmäßiger Weise gefordert, dass jede Anforderung mit einer ID versehen wird, um Anforderungen einfach referenzieren zu können. Auch Abhängigkeiten zwischen Anforderungen und die Autoren der Anforderungen sollten dokumentiert werden.

2. Der bei einer Änderung durchzuführende Prozess wird definiert.

Der Änderungsprozess sollte wie folgt durchgeführt werden, nachdem ein Problem erkannt wurde:

1. Problemanalyse: gibt Auskunft darüber, was geändert werden muss.

2. Änderungsanalyse und Aufwandsschätzung: Hier werden Abhängigkeiten zu anderen Anforderungen und die Notwendigkeit der Änderung analysiert und der Aufwand der Änderung abgeschätzt.

3. Änderungsauftrag: Basierend auf dem vorangegangenen Schritt wird entschieden, ob die Änderung durchgeführt wird oder nicht.

4. Wird die Änderung beauftragt, folgt die Implementierung der Änderung gemäß dem verwendeten Vorgehensmodell.

Noch eine Bemerkung zum Änderungsprozess: Es sollte im Vorgehensmodell festgelegt werden, wann dieser Prozess in Kraft treten muss. Jede Änderung über diesen Prozess laufen zu lassen ist nicht immer zielführend.

4.6 Der Projektstrukturplan

Sind die Benutzeranforderungen bekannt und wurde das System gemäß den Anforderungen spezifiziert, ist man nun in der Lage, das Projekt in Teilprojekte (Arbeitspakete) zu zerlegen. Die entsprechende Zerlegung nennt man *Projektstrukturplan*. Der Projektstrukturplan ist Teil des Projektplans. Es gibt verschiedene Möglichkeiten, den Projektstrukturplan darzustellen. Abbildung 4.4 zeigt eine hierarchische Struktur in 4 Ebenen.

Es gibt verschiedene Arten von Projektstrukturplänen:

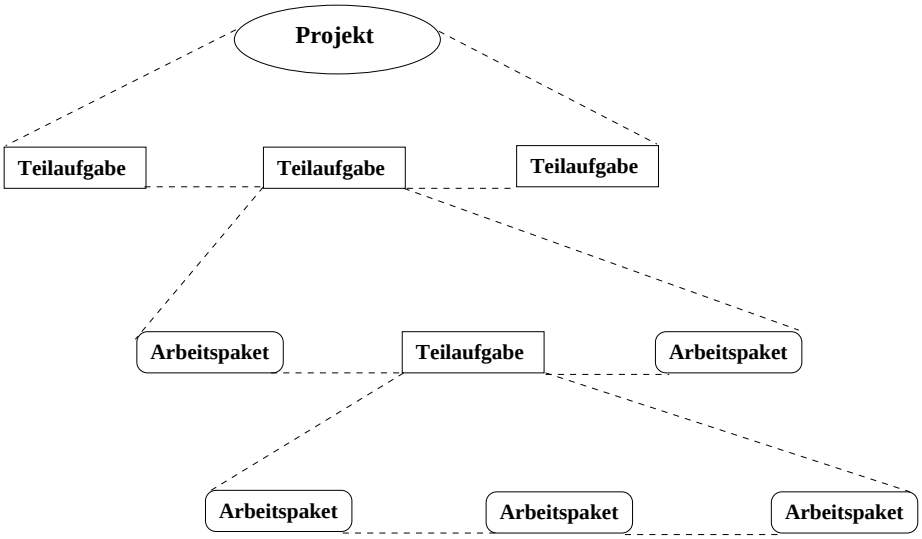

Abbildung 4.4: Möglicher Aufbau eines Projektstrukturplans

- Der Projektstrukturplan berücksichtigt nur die Arbeitspakete des zu entwickelnden Produkts (z.B. Personenkraftwagen: Chassis, Antrieb, Fahrgestell usw.). Derartige Pläne werden auch Produktstrukturplan genannt, da die Baugruppen/Bauelemente/Komponenten des Produktes unterteilt werden.
- Der Projektstrukturplan wird funktionalorientiert gegliedert, d.h. die zur Projektrealisierung erforderlichen Funktionen/Methoden werden untergliedert. Diese Art von Strukturplan ist der Beschreibung des Vorgehensmodells ähnlich.
- Der Projektstrukturplan wird sowohl funktionell als auch aufbauorientiert (Produktstrukturplan) gegliedert. Das Projekt wird wie beim Produktstrukturplan zuerst in seine Komponenten zerlegt. Jeder Komponente werden dann die zur Realisierung notwendigen Methoden zugeordnet (z.B. Grobentwurf, Feinentwurf, Produktion ...).

Egal für welche Form man sich entscheidet, eine Zerteilung des Produkts in Komponenten und die Zuordnung von Methoden zu jeder Komponente wird im Projektplan vorkommen müssen, allerdings nicht notwendigerweise im Projektstrukturplan, da dieser bei größeren Projekten sehr umfangreich und unübersichtlich werden kann. Ein Vorschlag ist, im Projektstrukturplan nur die Komponenten des Produkts zu berücksichtigen, die Zuordnung der entsprechenden Methoden gegebenenfalls getrennt zu behandeln. Auch wird der Projektstrukturplan nicht unbedingt vollständig sein, da man nicht alle Komponenten und zur Realisierung notwendigen Methoden in dieser Planungsphase kennen wird. Der Projektstrukturplan kann auch als sehr grobes Architectural Design verstanden werden (d.h. ohne Schnittstellenbeschreibung). Wichtig ist es, die Arbeitspakete so zu definieren, dass möglichst wenig Abhängigkeiten zwischen

den Arbeitspaketen bestehen, sodass jedes Arbeitspaket weitgehend unabhängig von den anderen bearbeitet werden kann.

Der Projektstrukturplan ist Ausgangspunkt für viele untergeordnete Projektpläne (z.B. Ablaufplan, Kostenplan, siehe Abschnitt 3.1).

4.7 Struktur eines Anforderungsdokuments

Wie für den Projektplan (Abschnitt 3.2) gilt auch für das Anforderungsdokument, dass hier nur beschrieben wird, welche Punkte das Dokument abhandeln kann. Abhängig vom Projekt können gewisse Punkte entfallen. Die hier gezeigte Struktur kann auch als Checkliste dienen. Die Gliederung entspricht den Aktivitäten des V-Modells zur Anforderungsanalyse ([11]). Die Gliederung findet man auch unter http://www.informatik.uni-bremen.de/gdpa/home.htm.

- **Ist-Aufnahme und Ist-Analyse**

  Dieser Punkt wird relevant, wenn bereits ein System dieser Art existiert und durch ein neues ersetzt werden soll oder das zu entwickelnde System in existierende ablauforganisatorische oder technische Gegebenheiten eingebunden werden soll.

- **Fachliche Anforderungen**

  - Grobe Systembeschreibung: Die grobe Systembeschreibung hat hier in der Form eines Gesamthorizonts insbesondere folgenden Kriterien zu genügen:
    * Es muss klar werden, welche fachlichen Aufgaben durch das System unterstützt werden sollen. Daher muss die Einordnung des Systems in die Organisation bzw. sein Integrationsumfeld beschrieben werden.
    * Sie muss erkennbar machen, welche Gesamtfunktionalität im Endausbau abgedeckt werden soll und welche Prioritäten dabei im weiteren Verlauf erwartet werden.
    * Sie muss das Einsatzkonzept soweit beinhalten, dass daraus die technische und organisatorische Einsatzumgebung und die Randbedingungen ableitbar sind.
    * Sie muss quantitative Abschätzungen enthalten, die es ermöglichen, grundsätzliche technische Entscheidungen in der richtigen Größenordnung zu treffen (Netzauslegung, Rechnerkapazitäten, usw.)
    * Sie muss so präzise erfolgen, dass im weiteren Verlauf jederzeit entscheidbar ist, wann dieser Gesamthorizont verlassen wird und dementsprechend technische Entscheidungen neu überdacht werden müssen.
    * Die Verwendung vordefinierter fachlicher Bausteine (Standardthemenbereiche) ist zu berücksichtigen. Mögliche Kandidaten zur Wiederverwendung auf der Ebene der Anwenderforderungen im Rahmen der beabsichtigten Entwicklung sind zu benennen.

  - Organisatorische Einbettung: Dieses Kapitel enthält eine Beschreibung des organisatorischen Einsatzbereiches des Systems: die Aufbau- und Ablauforganisation beim Anwender, Nutzerklassen, Verantwortlichkeiten und Zuständigkeiten beim Einsatz sowie die weitere Ausrüstung des Anwenders.
  - Nutzung: Hier sind Anforderungen an die Bedienung und Nutzung des Systems festzuhalten, wie z.B. an mobilen oder stationären Einsatz, Einsatzzeiten, an die Kommunikation von Nutzer und System, an den Umfang der vom System automatisch bereitzustellenden Dienste, usw.
  - Kritikalität des Systems: Die Kritikalität des Systems wird festgelegt (evtl. auch definiert) und auf der Basis der Kritikalitätsdefinition im QS-Plan begründet.
  - Externe Schnittstellen: Anforderungen an die externen Systemschnittstellen zu Nachbarsystemen und im speziellen an die Mensch-Maschine-Schnittstelle sind zu formulieren. Anforderungen hinsichtlich der Ergonomie sind hier zu berücksichtigen. Die technische Ausgestaltung der Nutzeroberfläche wird im Rahmen der technischen Anforderungen definiert.
  - Beschreibung der Funktionalität: Dieses Kapitel nimmt die fachliche Strukturierung der Funktionalität des Systems aus Anwendersicht auf:
    * Zur Festlegung der geforderten Systemfunktionalität werden Geschäftsprozesse (Use-Cases) definiert.
    * Fachliche Strukturierung des Systems in Bereiche (Funktionen). Für jede der Bereiche/Funktionen werden die Anforderungen definiert.
  - Qualitätsforderungen: Es werden Anforderungen hinsichtlich der Qualitätsmerkmale formuliert:
    * Zuverlässigkeit
    * Nutzerfreundlichkeit
    * Effizienz
    * Wartbarkeit
    * Übertragbarkeit
    * Wiederverwendbarkeit
    * statische und dynamische Leistungsdaten, usw.

- **IT-Sicherheitsziel**

  In diesem Kapitel wird gegebenenfalls vorgegeben, welche Anforderungen an Verfügbarkeit, Integrität oder Vertraulichkeit von bestimmten Funktionen oder Informationen bestehen.

- **Bedrohungs- und Risikoanalyse**

  Die für das System relevanten Bedrohungen sind zu ermitteln und die damit verbundenen Risiken unter Berücksichtigung von Eintrittswahrscheinlichkeiten und zu erwartenden Schäden zu bewerten.

- **IT-Sicherheit**

  Die IT-Sicherheit eines Systems wird gewährleistet durch Maßnahmen in der Umgebung des Systems (Zutrittssicherung, Sicherung gegen Abstrahlung, sichere Handhabung von Datenträgern, usw.) oder durch Maßnahmen im System selbst (Passwortschutz, Mitlaufen von Protokollen, usw.).

  In den Anforderungen an die IT-Sicherheit wird festgelegt, welche Wirkung von der Gesamtheit solcher Maßnahmen erwartet wird. Die Bedrohungen aus der Bedrohungs- und Risikoanalyse müssen den Anforderungen zugeordnet werden. Es muss klargestellt sein, dass die Maßnahmen zur IT-Sicherheit den Bedrohungen entgegenwirken.

- **Randbedingungen**

  - Technische Randbedingungen: Auf der Basis einer Betrachtung der technischen Einsatzumgebung können z. B. Randbedingungen hinsichtlich folgender Punkte fixiert werden:
    - Technische Anforderungen an Schnittstellen
    - Vorgaben für die technische Realisierung von Schnittstellen
    - Hinweise und Vorgaben für den technischen Ablauf
    - Vorgabe von Fertigprodukten
    - Einhaltung von technischen Standards
    - Vorgaben aus der technischen Entwicklungs- oder Einsatzumgebung.

    Zusätzlich können hier auch technische Anforderungen an die Umgebung vermerkt werden (z. B. für Hardware: erforderliche Klimatisierung, Vibrationsdämpfung, usw.).
  - Organisatorische Randbedingungen: Auf der Basis einer Betrachtung der organisatorischen Einsatzumgebung können z. B. Randbedingungen hinsichtlich folgender Punkte fixiert werden (es handelt sich hier nicht um Projektorganisation, sondern um die Organisation die das Produkt bei seinem Einsatz verlangt):
    - Bedarf an Kommunikation, Kooperation und Koordinierung zwischen Stellen innerhalb der Ablauforganisation beim Anwender
    - Informationsbedarf bei der Abwicklung von Geschäftsprozessen
    - Ausführungsrhythmus und Wiederholfrequenz der Geschäftsprozesse
    - geographische Verteilung der Anwender innerhalb des Einsatzgebietes
    - Vorgaben aus der Organisationsentwicklung zum Personalstand, Stellenpläne, Arbeitsplatzbeschreibungen
    - Vorgaben aus Personalgesetzgebung (Beamtengesetz, Betriebsverfassungsgesetz, Mitbestimmung, Tarifverträge)
    - Zusammenarbeit mit externen Dienststellen und Behörden (überbehördliche Kooperation, Amtshilfe)
  - Sonstige Randbedingungen: Randbedingungen, die weder technischer noch organisatorischer Natur sind, sind zu fixieren (z. B. zeitliche und wirtschaftliche Randbedingungen)

Kapitel 5

Zeit-Management

Die bisherigen Themen dieses Manuskripts behandelten primär die Fragen des "Was ist im Rahmen des Projekts zu verwirklichen" (Benutzeranforderungen und Spezifikation) und "Wie" im groben bei der Entwicklung vorzugehen ist (Vorgehensmodelle). Dieses und die folgenden Kapiteln handeln von quantitativen Problemen, nämlich wie viel Zeit benötigt das Projekt, wieviel kostet es und wie hoch sind die Risiken des Projekts. In einem wirtschaftlichen Umfeld entscheiden die quantitativen Parameter Zeit und Kosten wesentlich den Erfolg eines Projekts, oftmals mehr als die qualitativen Parameter (wie wird entwickelt).

Zweck des Zeit-Managements ist es, Voraussagen über die Dauer und zeitliche Gliederung der einzelnen Aktivitäten zu treffen.

Unter Zeit-Management fallen jene Aktivitäten, die eine termingerechte Fertigstellung des Projekts ermöglichen. Dazu gehören (siehe auch Abbildung 3.2):

1. **Definition aller Aktivitäten**(Activity Definition), die Teil des Projektablaufs sind. Primäre Quelle für die Auswahl der Aktivitäten ist der Projektstrukturplan.

2. **Ablaufplanung der Aktivitäten** (Activity Sequencing): Es gilt, Abhängigkeiten zwischen den einzelnen Aktivitäten zu finden und die Aktivitäten dann den Ablauf entsprechend zu planen (voneinander unabhängige Aktivitäten können parallel ausgeführt werden).

3. **Zeitliche Aufwandsabschätzung** (Activity Duration Estimating) der einzelnen Aktivitäten: Dies ist wohl der schwierigste Teil des Zeitmanagements. Üblicherweise wird der Aufwand in Mann-Tagen / Mann-Monaten geschätzt (diese Aufwandsabschätzung entspricht nicht der tatsächlichen Dauer der Aktivitäten).

4. **Zeitplanung** (Schedule Development): Ausgangspunkt für die Zeitplanung ist die Aufwandsabschätzung und die zur Verfügung stehenden Ressourcen (Personen, Budget, Arbeitsmittel, Zeitvorgaben).

5. **Controlling**: Die Einhaltung des Zeitplans muss während der Projektabwicklung ständig kontrolliert werden. Bei Verzug müssen die Ursachen analysiert werden und Gegenmaßnahmen ergriffen werden.

5.1 Definition der Aktivitäten

Aufgrund des Projekt Strukturplans (Abschnitt 4.6 und des verwendeten Vorgehensmodells (Kapitel 2) ist man in der Lage, die für das Projekt relevanten Aktivitäten zu definieren. Angenommen, der Projektstrukturplan besteht in seiner ersten Ebene aus 3 Komponenten, die ich hier einfach als Komponente_1, Komponente_2 und Komponente_3 bezeichne. Wir gehen weiter davon aus, dass Projekt sei nicht zu umfangreich und birgt kaum Risiken hinsichtlich seiner Entwicklung. Daher verwenden wir das Wasserfallmodell wie in Abbildung 2.4. Dann könnten wir die Aktivitäten wie in Tabelle 5.1 dargestellt ableiten.

Aktivitäten	
• Anforderungsdefinition • Architectural Design • Entwurf der Komponente_1 • Entwurf der Komponente_2 • Entwurf der Komponente_3 • Implementierung der Komponente_1 • Implementierung der Komponente_2 • Implementierung der Komponente_3	• Test der Komponente_1 • Test der Komponente_2 • Test der Komponente_3 • Integrationstest • Systemtest • Abnahme durch den Kunden

Tabelle 5.1: Aktivitäten entsprechend d. Projektstrukturplan u. d. Vorgehensmodell

Nicht berücksichtigt sind hier die Kontrollaktivitäten und QM-Aktivitäten (z.B. Reviews, Testfallauswertung ...). Auch wurden die Aktivitäten Benutzeranforderungen und Systemspezifikation zu einer Aktivität zusammengefasst.

5.2 Ablaufplanung der Aktivitäten

Hat man alle notwendigen Aktivitäten definiert, gilt es, Abhängigkeiten zwischen den einzelnen Aktivitäten zu finden und den Ablauf entsprechend zu planen. Voneinander unabhängige Aktivitäten können parallel ausgeführt werden. Hier zeigt sich, wie wichtig es ist, die Arbeitspakete so zu definieren, dass sie möglichst unabhängig voneinander ausgeführt werden können. Bei der Beschreibung der Abhängigkeiten hat sich eine graphische Darstellung bewährt, da diese eine schnelle Übersicht ermöglicht. Man unterscheidet prinzipiell drei Arten von Diagrammen:

- Activity on Arrow (AOA)
- Activity on Node (AON)

Wir beschäftigen uns nur mit Precedence Diagrammen, eine Erweiterung von AON Diagrammen. Abbildung 5.1 zeigt das Beispiel aus Abschnitt 5.1, dargestellt über ein Precedence Diagramm, genauer durch ein Gantt Diagramm. Die Aktivitäten werden durch Balken (sind in diesem Fall die Nodes) dargestellt, die Abhängigkeiten durch Pfeile zwischen den entsprechenden Balken. Dieses Diagramm wurde mit dem Open Source Project Management Tool MrProject Version 0.91 erstellt. Dieses Programm ist für kleinere Projekte durchaus geeignet und wird ständig weiterentwickelt.

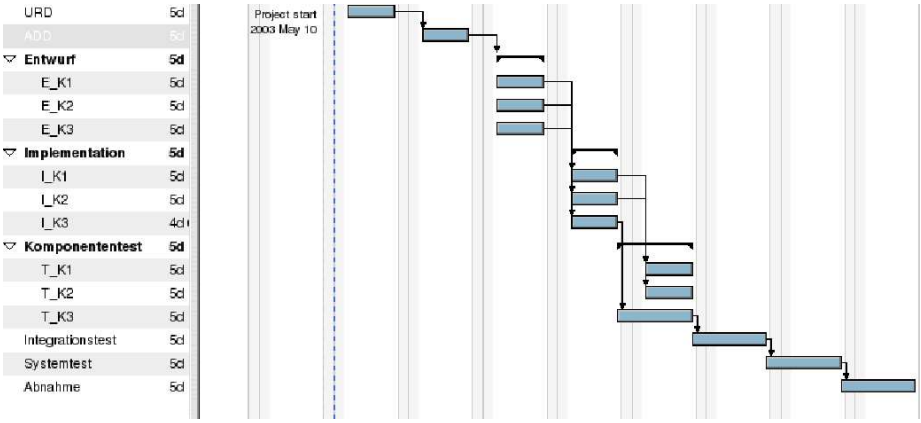

Abbildung 5.1: Beispiel eines Gantt Diagramm

In PD Diagrammen gibt es folgende Arten von Relationen:

- *Finish-to-Start*(FS): Diese Relation drückt aus, dass der Beginn der nächsten Aktivität vom Abschluss der vorangegangenen Aktivität abhängt (dies ist die am meisten verwendete Relation in Precedence Diagrammen).
- *Finish-to-Finish*(FF): Der Abschluss des Nachfolgers hängt vom Abschluss des Vorgängers ab.
- *Start-to-Start*(SS): Der Beginn des Nachfolgers hängt vom Beginn des Vorgängers ab.
- *Start-to-Finish*(SF): Der Abschluss des Nachfolgers hängt vom Beginn des Vorgängers ab.

Abbildung 5.2 zeigt die verschiedenen Typen von Relationen, wie sie üblicherweise in Precedence Diagrammen dargestellt werden. Die Finish-to-Start Relation bedeutet, dass B erst dann beginnen darf, wenn A beendet ist. Die Finish-to-Finish Relation in Abbildung 5.2 drückt aus, dass D frühestens n-Tage nach C fertig sein darf. Die Start-to-Start Relation besagt, dass F frühestens n-Tage nach Beginn von E begonnen werden darf (vergleiche diese Relation mit Fast Tracking aus Abschnitt 2.2.1). Die Start-to-Finish Relation schließlich bedeutet, dass H frühestens n-Tage nach Beginn von G abgeschlossen werden darf. Die Zeitdifferenz, die die Relationen vorgeben, werden auch als *Lag* bezeichnet. Finish-to-Finish und Start-to-Finish Relationen werden sehr selten gemeinsam benötigt und wenn sollten sie sehr umsichtig verwendet werden. Abbildung 5.3

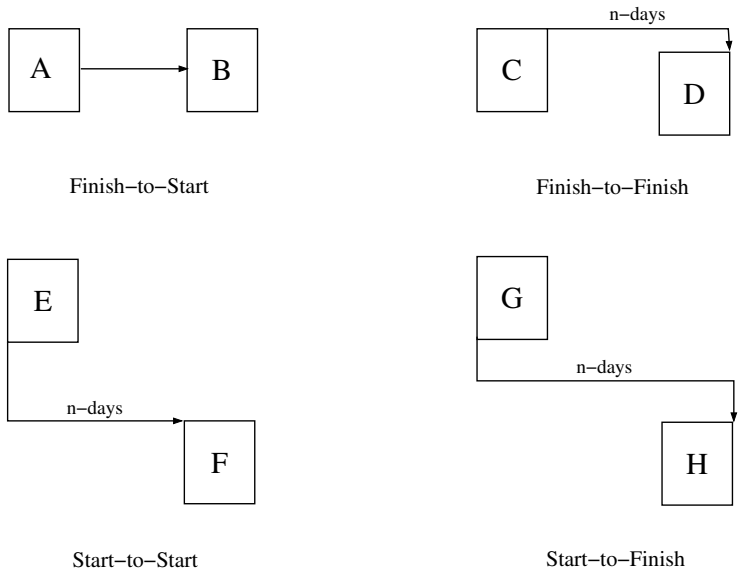

Abbildung 5.2: Typen von Relationen in Precedence Diagrammen

zeigt ein interessantes Beispiel, was bei einer gemeinsamen Anwendung einer FF und SS Relation passieren kann. Angenommen, die Aktivitäten A,B und C benötigen jeweils 15 Tage. Der Lag der FF und SS Relationen betrage jeweils 5 Tage. Die Gesamtdauer des Projekts wäre 25 Tage. Verkürzt man B, so verlängert sich die Gesamtdauer des Projekts. Angenommen, B würde um 5 Tage verkürzt, wieviele Tage würde das Projekt dauern? Was würde passieren, wenn man B um 5 Tage verlängert?

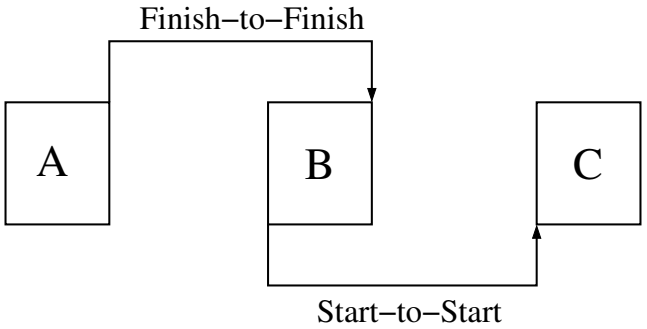

Abbildung 5.3: Eine Verkürzung von B verlängert die Dauer des Projekts

Bei einer iterativen Vorgehensweise dürfen keine Zyklen entstehen, die Iterationen müssen anders berücksichtigt werden (z.B man sieht die einzelnen Iteration durch eigene Aktivitäten vor).

Zum Abschluss dieses Kapitels noch zwei von Studenten erdachten Beispiele von Finish-to-Finish Relationen:

Der Chemiker Die Entwicklung eines flüchtigen Impfstoffs, der aus einer exo-

tischen Pflanze gewonnen wird, soll geplant werden. Aufgrund der Flüchtigkeit des Impfstoffs werden Bakterien gezüchtet, die diesen aufnehmen. Nach der Züchtung der Bakterien dauert es n Tage, bis sich diese in der erforderlichen Menge reproduziert haben. Daraus folgt, dass die Züchtung der Bakterien n Tage vor Fertigstellung des Impfstoffs beendet sein muss. Anders ausgedrückt: Die Herstellung des Impfstoffs darf nicht früher als n Tage nach der Züchtung der Bakterien beendet sein, da in diesem Fall zu wenige Bakterien zur Aufnahme des Impfstoffs zur Verfügung stehen würden.

Der beinharte Projektleiter Ein Chemielabor forscht an gefährlichen Substanzen und entwickelt auch die entsprechenden Behälter, um diese aufzubewahren. Ein Behälter muss aus Sicherheitsgründen n Tage vor der entsprechenden Substanz fertiggestellt sein. Die Frage, die sich hier aufdrängt ist, was passiert mit der Substanz, wenn der Behälter nicht rechtzeitig fertiggestellt werden kann ? Hier gilt es abzuwägen, ob nicht der Sicherheit wegen eine Verlängerung der Projektlaufzeit in Kauf genommen werden sollte (die Entwicklung der gefährlichen Substanz wird erst dann in Angriff genommen, wenn der Behälter fertig entwickelt und geprüft wurde - also eine Finish-to-Start Relation).

5.3 Zeitliche Aufwandsabschätzung

(FIXXME: Personen Monat genauer erklären [by Karl Blümlinger (kbluem@iicm.edu)]) ⇐ Nachdem die Abhängigkeiten zwischen den Aktivitäten ermittelt wurden, wird versucht, den zeitlichen Aufwand des Projekts zu bestimmen. Der zeitliche Aufwand, angegeben in Personenmonaten (Feiertage, Urlaub sind nicht inkludiert !), entspricht nicht der tatsächlichen Dauer des Projekts. Diese hängt von der Parallelität der Aktivitäten und von den zur Verfügung stehenden Ressourcen ab. Für die Softwareentwicklung gibt es folgende empfehlenswerte Methoden, um den Aufwand zu ermitteln (all diese Methoden können auch zur Kostenschätzung herangezogen werden).

- Experten-Schätzung (Expert Judgement): Experten schätzen den Aufwand des Projekts. Sicherlich die beste Methode, wenn Experten zur Verfügung stehen.
- Schätzung durch Vergleich (Analogous Estimating): Man vergleicht das Projekt mit vorher durchgeführten Projekten. Voraussetzung für die Anwendung dieser Methode ist erstens, dass die vorangegangenen Projekte dem Projekt ähnlich sind und zweitens, dass die vorangegangenen Projekte nach Projektabschluss analysiert wurden (Dokumentation von Kennzahlen, Analyse der kostentreibenden Faktoren). Diese Methode ist eine Art Experten Schätzung.
- Algorithmische Modelle: Basierend auf Kennzahlen wird der Aufwand mathematisch ermittelt. Die Schwierigkeit bei diesem Modell besteht im Quantifizieren der Kennzahlen. Ein weit verbreitetes algorithmisches Modell ist das COCOMO Modell.

Bei Vergleichen zwischen Schätzmodellen stößt man häufig auf die Begriffe *top-down* und *bottom-up* Modelle. Zu Beginn eines Projekts wird man gezwungenermaßen auf top-down Modelle zurückgreifen müssen (z.B. mit expert judgement od. analogous estimating). Algorithmische Modelle verlangen genauere Kenntnis der Spezifikation des Projekts und der notwendigen Komponenten. Es handelt sich bei algorithmischen Modellen daher in der Regel um bottom-up Modelle.

Bei der Experten-Schätzung greift man gerne auf die sogenannte *Delphi Technik* zurück. Diese besteht aus den folgenden Schritten:

1. Ein Koordinator übergibt jedem Experten die Spezifikation des Projekts.
2. Der Koordinator ruft ein Treffen aller Experten ein, in dem die für die Schätzung relevanten Belange besprochen werden.
3. Jeder Experte erstellt anonym eine erste Schätzung.
4. Der Koordinator fasst die Ergebnisse in einem Bericht zusammen und verteilt diesen an die Experten.
5. Der Koordinator ruft ein Treffen aller Experten ein, in dem die strittigen Punkte besprochen werden (also jene Punkte, wo die Schätzungen der Experten zu weit auseinander liegen).
6. Jeder Experte erstellt wiederum eine anonyme Schätzung und Punkte 4-6 werden solange wiederholt, bis ein zufriedenstellendes Ergebnis erreicht wird.

Der Erfolg dieser Technik liegt in der Anonymität der Schätzung und der Diskussion zwischen den Experten.

5.3.1 Schätzung und Vorhersage

Schätzung ist eine Vorhersage basierend auf einer wahrscheinlichen Annahme. Tom DeMarco schlägt in [12] folgende Definition vor:

> Eine Schätzung ist eine Vorhersage, deren Wahrscheinlichkeit, dass sie eintritt, gleichermaßen über oder unter dem tatsächlichen Ergebnis liegt.

Der Duden schreibt dazu:

> Schätzen: ohne exaktes Wissen, nur auf Erfahrung basierend, näherungsweise bestimmen.

Das heißt, auf Schätzung ist man dann angewiesen, wenn kein Rückgriff auf exaktere Methoden (z.B. Messen) möglich ist. Es gibt gewisse Bereiche, in denen die Zeit/Kosten eines Projekts sehr genau vorhergesagt werden können, zum Beispiel in der Bauwirtschaft. Dort existieren Tabellen, sogenannte Standardleistungsverzeichnisse, die es der Projektleitung ermöglichen, den Aufwand eines Projekts vorherzusagen (man schätzt nicht). Dort wird häufig gar nichts geschätzt, sondern der Aufwand aufgrund zuverlässiger Daten berechnet. Dies trifft allerdings auch in der Bauwirtschaft nur dann zu, wenn es sich um ein

Projekt handelt, das keine neuen Technologien erfordert oder dessen Umfeld hinreichend bekannt ist. Beispiele dafür sind der Bau des olympischen Zeltdachs in München oder den Bau des Eurotunnels. Grundlage der Vorhersage sind Kennzahlen, auch Metriken genannt, die es in der Bauwirtschaft für viele Fälle gibt (z.B. die Länge, Breite und Höhe einer Wand). Die Metriken zusammen mit Erfahrungswerten ermöglichen gute Vorhersagen.

Um den Aufwand eines Projekts zu kontrollieren, wird dieser im Laufe des Projekts immer wieder neu ermittelt, meistens über Schätzung. Abbildung 5.4 zeigt die Unsicherheit der Schätzung in den verschiedenen Phasen des Projekts. Zu Beginn des Projekts liegen häufig noch zu wenig Daten über das Projekt vor, sodass eine Schätzung mit großer Unsicherheit behaftet ist. Je länger das Projekt andauert, umso mehr Daten liegen vor, umso genauer kann geschätzt werden. Nach Auslieferung sollte bei guter Projektleitung keine Schätzung mehr vorgenommen werden müssen, um den Gesamtaufwand des Projekts zu ermitteln (es sei denn Wartung und Weiterentwicklung sind Teil des Projekts).

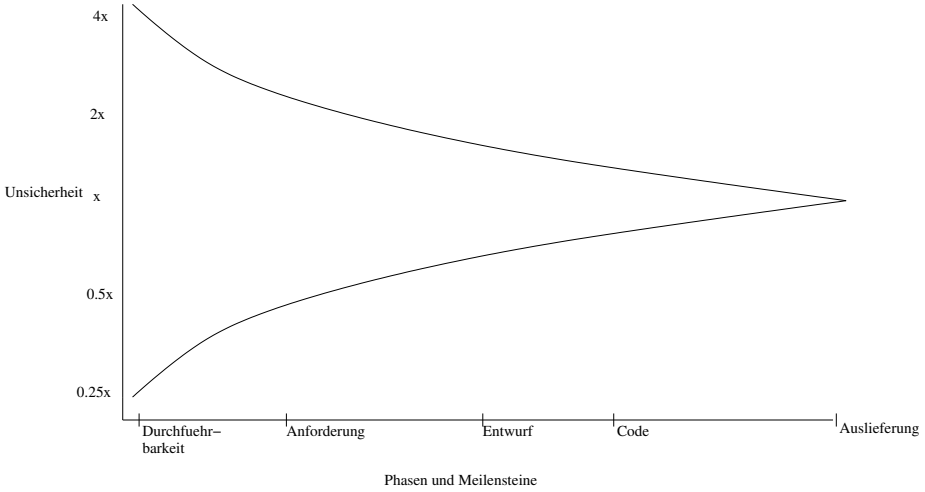

Abbildung 5.4: Unsicherheit der Schätzung im Projektverlauf [14]

5.3.2 Probleme des Schätzens

Mit den zur Verfügung stehenden Daten eines Projekts wird der Aufwand geschätzt. Wie Abbildung 5.4 zeigt, sind Schätzungen in der Frühphase eines Projekts mit großer Unsicherheit behaftet. Dennoch muss schon häufig eine Schätzung zu diesem Zeitpunkt vorgenommen werden. Im Folgenden werden beliebte Fallen beim Schätzen aufgezeigt. Eine ausführlichere Abhandlung des Themas findet man in ([12]).

Wurde einmal eine erste Schätzung erstellt, so neigt man aus verschiedenen Gründen dazu, diese Schätzung als verbindlich zu betrachten und im weiteren Projektverlauf keine neuen Schätzungen mehr vorzunehmen (Tom DeMarco bezeichnet dies als *nächste Schätzung = letzte Schätzung*). Ein Grund für dieses Verhalten liegt darin, dass man ungern Fehler zugibt und eher hofft, einen eventuellen Verzug wieder aufholen zu können. Weiters kann die erste

Schätzung zu einer Zielvorgabe entarten (hier könnte man auch sagen: *Ursprüngliche Schätzung = Zielvorgabe*).

Hat man sich zur Revision einer neuen Schätzung entschieden sollte man nicht versucht sein, nach den Erwartungen derer zu schätzen, denen man Rechenschaft schuldig ist (Vorgesetzte, Kunden) (nach DeMarco: *neue Schätzung = letzte Schätzung + erlaubte Verzögerung*). Ein ähnliches Problem liegt vor, wenn man die erste Schätzung in Hinblick der Erwartungen des Kunden/Vorgesetzten vornimmt.

Ist eine Projektphase im Verzug, so neigt man dazu, folgende "Abschätzungen" um genau diesen Verzug weniger zu erstellen (DeMarco: *Vergangene Überschreitung = künftige negative Überschreitung*). Derartige Voraussagen erfüllen sich leider nur in sehr wenigen Fällen. Um die oben geschilderten Probleme zu vermeiden, sollten Schätzungen von Leuten durchgeführt werden, die nicht am Projekt beteiligt sind.

Nun könnte man einwenden, dass es in der Software Branche durchaus üblich ist, sich dem Aufwand betreffend nach dem Kunden zu richten. Nur hat der Vorgang des Schätzens damit überhaupt nichts zu tun. Es geht schlicht darum festzustellen, wie hoch der Aufwand des Projekts ist. Aufgrund dieser Schätzung sollte man entscheiden können, ob die Vorstellungen des Kunden machbar sind.

Schätzungen werden auch gern zu Motivationszwecken durchgeführt. Das heißt, es wird ein aller Voraussicht nach zu geringer Aufwand geschätzt, um die Mitarbeiter zu schnellerem Arbeiten anzuregen. Auch dieses Vorgehen hat mit Schätzen nichts gemein. Hinzu kommt, dass eine derartige Motivation die Qualität des zu entwickelnden Produkts mindern kann.

5.3.3 Software Kennzahlen

In der Bauwirtschaft stützt man sich auf empirische Aufwandsvorhersage (vergleiche analogous estimating), das heißt, gestützt auf über Jahre gesammelte Daten wird der Zeit- und Kostenaufwand ermittelt. Ohne entsprechende Kennzahlen ist eine Schätzung auch hier nicht möglich. In der Softwareindustrie ist man seit Jahren bemüht, geeignete Kennzahlen zu entwickeln, um die Unsicherheit und Risiken einer Schätzung zu mindern. Eine Kennzahl sollte folgende Eigenschaften besitzen:

- Sie muss messbar sein.
- Sie sollte auch in frühen Projektphasen messbar sein.
- Sie muss mit dem resultierenden Arbeits- und Kostenaufwand korrelieren.
- Sie sollte möglichst unabhängig sein (d.h. von den Projektmitarbeitern nicht korrumpiert werden können).

Anzahl der Codezeilen

Eine sehr umstrittene Kennzahl ist die Anzahl der Codezeilen (LOC) und dafür gibt es mehrere Gründe:

- Die Komplexität des Codes hängt vom zu implementierenden Algorithmus ab.

- Die Anzahl der Codezeilen ist schwer abschätzbar, die Kennzahl kann erst in späteren Phasen vernünftig geschätzt werden.
- Sie ist abhängig von der verwendeten Programmiersprache.
- Abhängig von der Art, wie die Codezeilen gezählt werden.

Der Problematik der LOC sollte man sich in jedem Fall bewusst sein. Die Anwendung der LOC als Maßeinheit ist vor allem für Produktivitätsmessungen sehr problematisch, wenn man das Vorgehensmodell außer Acht lässt: So ist eine Erkenntnis der Softwareentwicklung, dass ein gutes Design einfacher zu implementieren ist als ein schlechtes Design, ein gutes Design letzten Endes in weniger LOC resultiert. Die Verwendung der LOC als Kennzahl für Produktivität (z.b Produktivität P = LOC/PM) würde also schlechtes Design belohnen (generell würden alle Aktivitäten bestraft, die keinen unmittelbaren Code erzeugen).

Für andere Anwendungsgebiete mag LOC eher Verwendung finden, so zum Beispiel zur Ermittlung der Fehlerrate (engl. defect rate). Hier sollte man sich allerdings auf einen Standard einigen, wie die Codezeilen gezählt werden. Man unterscheidet zwei Methoden: *physical LOC counting logical LOC counting*, wobei logical LOC counting die Zusammenfassung mehrere Anweisungen in einer Zeile berücksichtigt. Physical LOC counting ist daher einfacher, dafür aber vom Coding Standard abhängig. Nicht mitgezählt werden bei den Methoden Leerzeilen und Kommentare (keinen direkten Einfluss auf Fehlerrate und Komplexität des Codes). Gezählt werden meist Anweisungen inklusive Deklarationen und Definitionen. Wenn man Vergleiche von Software auf Basis von LOC vornimmt, sollte man immer die Programmiersprache und den zeitlichen Aufwand berücksichtigen. So ist im Allgemeinen die Produktivität in LOC/Monat bei der Verwendung von Assembler höher als bei höheren Programmiersprachen. Jedoch ist die Anzahl der erforderlichen Codezeilen bei Verwendung einer höheren Programmiersprache wesentlich geringer und infolgedessen auch der erforderliche Aufwand.

Untersuchungen zeigten weiters, dass viele Manager den Aufwand eines Projekts besser abschätzen konnten als die erforderlichen Codezeilen.

Function Points

Eine alternative Kennzahl zu Codezeilen sind die *Function Points*. Sie wurden 1979 von Albrecht [15] entwickelt, um die Produktivität der Entwicklung zu messen. Eine Funktion ist eine Ansammlung von Statements, die eine bestimmte Aufgabe erfüllen. Zu einer Funktion gehören selbstverständlich auch die Deklaration der formalen Parameter und der lokalen Variablen. Eine mögliche Kennzahl für den Umfang des Softwareprojekts ist die Anzahl der Funktionen, die für das Projekt zu implementieren sind. Leider ist die Anzahl der Funktionen im Vorfeld (auch nach der Spezifikationsphase) nur sehr schwer zu bestimmen. Die Idee von Albrecht war nun nicht, die Funktionen zu zählen, sondern die wichtigsten Komponenten des Softwareprojekts. So gesehen ist der Name *Function Point* nicht sehr glücklich gewählt. Die Vorteile der Function Points gegenüber LOC sind:

- Plattform unabhängig
- Schon in der Anforderungsphase verfügbar

- Vom Benutzer/Kunden nachvollziehbar
- Gültigkeit während des gesamten Entwicklungszyklusses (sofern sich die Anforderungen nicht ändern)

Function Points haben, wie aus obiger Aufzählung ersichtlich, den Vorteil, dass der Kunde die Anzahl der Function Points aufgrund der Spezifikation nachvollziehen kann und so auch Berechnungen verstehen kann, die die Function Points als Grundlage verwenden (Zeit- und Kostenberechnungen).

Albrecht geht davon aus, dass Software aus folgenden fünf Hauptkomponenten (Function Point Types) besteht (Abbildung 5.5):

- *Internal Logical Files (ILF)*
- *External Logical Files (ELF)*
- *External Inputs (EI)*
- *External Outputs (EO)*
- *External Inquiries (EQ)*

Funktionen, die zu den External Inputs, External Outputs und External Inquiries gehören, zählen zur Gruppe der *Transactional Functions*, und Funktionen, die Internal Logical Files und External Logical Files betreffen, gehören zur Gruppe der *Data Functions*. Betrachten wir die einzelnen Typen von Funktionen etwas näher:

Internal Logical Files: Dies sind all jene Daten, die der Benutzer der Software erstellt und manipuliert. Ein Logical File könnte zum Beispiel eine Relationale Datenbank sein.

External Logical Files: Dies sind all jene Daten, die der Benutzer selbst nicht erstellt oder verändert, also von einer externen Stelle zur Verfügung gestellt werden. Ein Beispiel wäre ein Programm zur Anzeige des Wetters, wobei die Datenbank mit den Wetterdaten von einer externen Stelle zur Verfügung gestellt wird. Auf diese Datenbank wird nur lesend zugegriffen.

Zu den Transactional Functions gehören all jene Funktionen, die auf diese Daten zugreifen (lesend, schreibend).

External Inputs: Darunter versteht man alle Benutzereingaben, die die Daten verändern (neue Daten hinzufügen, ändern und löschen bestehender Daten). Ein einfaches Beispiel wäre das Hinzufügen eines neuen Datensatzes in einer Datenbank. Daten, die eine gemeinsame Verarbeitung verlangen zählen als ein External Input.

External Outputs: Unter External Outputs fallen all die Funktionen, die dem Benutzer oder anderen Applikationen Output in jedweder Form bieten. Ein Beispiel wäre die Darstellung eines Statusreports am Bildschirm oder der Ausdruck desselben am Drucker.

External Inquiries: External Inquiries (Anfragen) ermöglichen dem Benutzer (oder anderen Applikationen) die gezielte Abfrage von Daten, wobei die die Abfrage definierenden Parameter vom Benutzer zur Verfügung gestellt werden. Der Unterschied zu den EO's besteht darin, dass EQ's parametrisiert sind und dass keine weiteren Berechnungen/Transformationen durchgeführt werden.

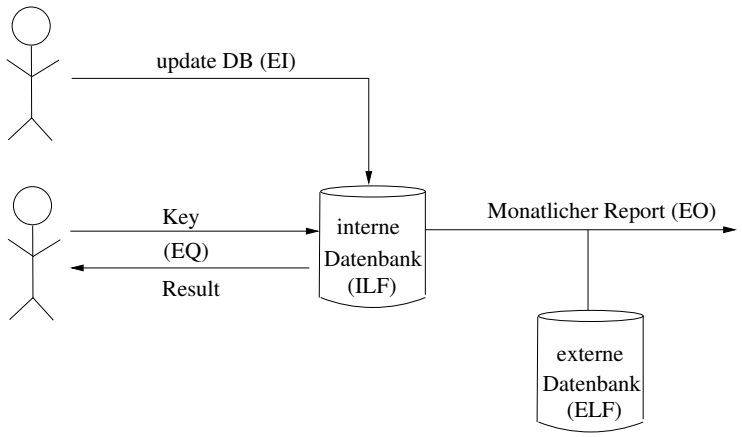

Abbildung 5.5: Mögliche Arten von Function Types

Im ersten Schritt wird die Anzahl der Function Points je Typ ermittelt. Im nächsten Schritt werden die Function Points nach ihrer Komplexität gewichtet. Tabelle 5.2 zeigt die Gewichtung der Function Points in Abhängigkeit des Typs und die Komplexitäts-Matrizen der einzelnen Typen. Die Komplexität wird in Low, High und Average eingeteilt. Die Bestimmung der Komplexität erfolgt mit Hilfe der Kompexitäts-Matrizen der Function Point Typen. Die Komplexität von ILF und EIF ist abhängig von der Anzahl der Gruppen von Daten (Record Element Types - RET) und der Anzahl der verschiedenen Datentypen (Data Element Types - DET). Die Komplexität der Transactional Functions hängt zum einen von der Anzahl der verschiedenen Typen von referenzierten Files ab (File Types Referenced - FTR dies können auch Datenstrukturen, Relationale Datenbanken ... sein), zum anderen von der Anzahl der verschiedenen Datentypen.

Die Formel zur Ermittlung der Anzahl der sogenannten *unberichtigten (unadjusted) Function Points* lautet:

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} fp_{ij} * g_{ij}$$

Wobei fp_{ij} die Anzahl des jeweiligen Function Points ist (es gibt 5 Typen, daher $i = 1..5$) und g_{ij} seine entsprechende Gewichtung.

Der Grund, warum man hier noch von unadjusted Function Points spricht liegt darin, dass es noch eine Reihe weiterer Faktoren gibt, die das System in ihrer Komplexität/Größe beeinflussen:

Gewichtung der Function Points			
Type	Low	Avg	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10
ILF und EIF Komplexitäts Matrix			
RETs	1-19	DETs 20-50	DETs 51+ DETs
1	Low	Low	Avg
2-5	Low	Avg	High
6+	Avg	High	High
EI Komplexitäts Matrix			
FTRs	1-4 DETs	5-15 DETs	16+ DETs
0-1	Low	Low	Avg
2	Low	Avg	High
3+	Avg	High	High
EQ und EI Komplexitäts Matrix			
FTRs	1-5 DETs	6-19 DETs	20+DETs
0-1	Low	Low	Avg
2-3	Low	Avg	High
4+	Avg	High	High

Tabelle 5.2: Gewichtung und Komplexität von Function Points

1. Daten Kommunikation (Daten- und Kontrollinformationen)
2. Verteilte Funktionen
3. Performance (Antwortzeit, Datendurchsatz)
4. Konfigurierbarkeit
5. Transaktionsrate (z.B. geforderte Transaktionen pro Sekunde)
6. Online Dateneingabe
7. End-user Effizienz (z.B. intuitive grafische Oberfläche)
8. Online Update (z.B. für ILF)
9. Komplexe Informationsverarbeitung
10. Wiederverwendbarkeit des Codes
11. Einfache Installation
12. Einfache Wartung (z.B. Backup/Restore, Konfiguration)
13. Verteilte Instanzen von Applikationen (z.B. in mehreren Firmen)
14. Einfache Adaptierung des Systems (Änderungen, Erweiterungen)

Um nun zu berichtigten (adjusted) Function Points zu gelangen, werden diesen Faktoren Werte zwischen 0 (unwichtiger Faktor) und 5 (sehr wichtiger Faktor) zugewiesen und nach folgender Formel ein Value Adjustement Factor (VAF) errechnet:

$$VAF = 0.65 + 0.01 \sum_{i=1}^{14} s_i$$

s_i ist der Wert des entsprechenden Faktors. Diese Formel wurde durch Regression ermittelt, das heißt an abgeschlossenen Projekten wurde das Verhalten der Projekte in Abhängigkeit der oben aufgelisteten Faktoren untersucht und mit Hilfe dieser Formel nachgebildet.

Schließlich wird die Anzahl der Function Points (FP) ermittelt:

$$FP = UFP * VAF$$

Man könnte bei einer unkritischen Betrachtungsweise des hier Gezeigten geneigt sein zu glauben, dieses Vorgehen zur Ermittlung der Function Points sei ein exaktes Vorgehen. Dem ist leider nicht so. Die Bestimmung der Unadjusted Function Points verlangt einiges an Erfahrung, da es keine exakten Vorgaben gibt, wie man einen Function Point genau eingrenzt. Genaues Zählen erfordert Spezialisten. Des weiteren gibt es mehrere Methoden, wie Function Points gezählt werden und wie diese kategorisiert werden (z.B. 3D Function Points und Feature Function Points). 1986 wurde die IFPUG (International Function Point User Group) ins Leben gerufen, die sich mit dem Thema Function Points beschäftigt. Diese gibt auch ein ständig aktualisiertes Handbuch zum Thema Function Points heraus [17].

Die hier beschriebene Vorgehensweise könnte auch zur Vermutung führen, Function Points wären nur für Software im Bereich Informationssysteme geeignet. Die Methode des Function Point-Zählens lässt sich durchaus auf andere Systeme (z.B. Echtzeit Systeme übertragen) [18].

Wer mehr über Function Points wissen möchte, dem können als Startpunkt die Function Points FAQ, herausgegeben von der Software Composition Technologies Inc, empfohlen werden (http://ourworld.compuserve.com/homepages/softcomp/fpfaq.htm).

Object Points

Eine Alternative zu Function Points stellen die Object Points dar. Diese können verwendet werden, wenn Programmiersprachen der 4. Generation (z.B. eine Datenbankprogrammiersprache) verwendet werden. Object Points sollten nicht mit Objekten von objektorientierten Programmiersprachen, wiewohl beide korrelieren (je mehr Object Points die Spezifikation der Software beinhaltet, desto mehr Objekte werden zur Implementierung der Software notwendig sein). Die hier vorgestellte Zählweise entspricht der COCOMO 2 Zählweise [14]. Ähnlich wie bei Function Points werden auch hier die vermuteten Komponenten der Software gezählt, nur sind bei Object Points folgende Komponenten in ihrer Anzahl wesentlich:

- Die Anzahl der angezeigten Bildschirmmasken (Type Screen).

- Die Anzahl der erzeugten Berichte (Type Report).
- Die Anzahl an 3GL Module (Module implementiert in einer Sprache der 3 Generation, also in einer höheren Programmiersprache), die zusätzlich zu den 4GL Modulen gebraucht werden (Type 3GL-Component).

Wie bei Function Points werden auch die Object Points nach ihrer Komplexität gewichtet. Tabelle 5.3 zeigt die Komplexitätsabschätzungen der einzelnen Typen und deren Gewichtung in Abhängigkeit der Komplexität. Eine 3GL Komponente hat immer die Gewichtung 10. *srvr* bezeichnet die Anzahl der Server Datenbanktabellen, die pro Screen/Report verwendet werden, *clnt* bezeichnet die Anzahl der Client seitigen Datenbanktabellen, die pro Screen/Report verwendet werden.

Gewichtung der Object Points			
Type	low	avg	high
Screen	1	2	3
Report	2	5	8
3GL-Component			10
Komplexität für Screens			
# der Views	# und Quelle der Daten		
	Total <4 (<¡ 2 srvr <2 clnt)	Total <8 (2/3 srvr 3-5 clnt)	Total 8+ (>3 srvr >5 clnt)
<3	low	low	Avg
3-5	low	Avg	High
8+	Avg	High	High
Komplexität für Reports			
# der Abschnitte	# und Quelle der Daten		
	Total <4 (<¡ 2 srvr <2 clnt)	Total <8 (2/3 srvr 3-5 clnt)	Total 8+ (>3 srvr >5 clnt)
0/1	low	low	Avg
2-3	low	Avg	High
4+	Avg	High	High

Tabelle 5.3: Gewichtung und Komplexität von Object Points

Die Formel zur Berechnung der Anzahl von Object Points ist ähnlich wie die zur Berechnung der UFP (Abschnitt *Function Points*):

$$object\_points = \sum_{i=1}^{3} \sum_{j=1}^{3} fp_{ij} * g_{ij}$$

Wobei fp_{ij} die Anzahl des jeweiligen Typs von Object Points ist und g_{ij} seine entsprechende Gewichtung. In die Zahl der Unadjusted Object Points fließt nun auch die Wiederverwendbarkeit von Modulen, die man im Projekt erreicht, ein und man erhält schließlich die Anzahl an Object Points (NOP):

$$NOP = (object\_points) * \frac{(100 - \%reuse)}{100}$$

Worin liegt nun der Vorteil in der Verwendung von Object Points gegenüber Function Points ? Der Einfluss auf die Schätzgenaugikeit ist nach Untersuchungen vernachlässigbar, doch sind Object Points schneller und leichter zu zählen (etwa 47% Zeitersparnis gegenüber Function Points) [19].

5.3.4 COCOMO 2

Allgemein lässt sich der Aufwand für ein Softwareprojekt mit folgender Formel berechnen:

$$Aufwand = A * Groesse^B * M$$

Bei A handelt es sich um einen konstanten Faktor, der gewisse Eigenheiten der Organisation wiederspiegelt, in der die Software entwickelt wird (eine zugegeben etwas dubiose Erklärung für das hier gesagte aber nicht weiter wichtig). Die *Größe* kann in Lines of Code, Function oder Object Points angegeben werden. Der Multiplikator M quantisiert verschiedene Prozess-, Produkt- und Entwicklungsfaktoren. Der Exponent B (scaling factor) drückt aus, dass sich der Aufwand nicht linear zur Größe des Projekts verhält.

Beim COCOMO Modell handelt es sich um ein algorithmisches Modell, wobei die Faktoren A, B und M auf empirischem Wege ermittelt wurden, d.h. es wurde eine Vielzahl von Softwareprojekten untersucht und daraufhin die Faktoren quantisiert. Das COCOMO Modell ist frei verfügbar, gut dokumentiert und es gibt sowohl kommerzielle als auch frei verfügbare Werkzeuge. Beim hier beschriebenen COCOMO 2 Modell handelt es sich um eine Weiterentwicklung des COCOMO Modells (dieses wird jetzt als COCOMO 81 bezeichnet). Es bietet Abschätzungen in Anlehnung an drei Entwicklungsstufen:

1. Die frühe Prototypenstufe: Diese beruht auf der Annahme, das bei Prototypen eine bereits erhältliche Software (meist 4GL Module) verwendet werden kann und basiert auf die Zählung von Object Points. Dieses Modell kann auch dann verwendet werden, wenn Software aus 4GL Modulen zusammengesetzt wird (Application Composition Model).

2. Die frühe Entwicklungsstufe (Early Design Model): Dieses Modell findet dann Anwendung, wenn die Spezifikation und eine grobe Systemarchitektur vorliegt. Die Größe des Projekts wird mit Hilfe der Function Points ermittelt. Die Schätzung verwendet eine kleine Anzahl an Faktoren (M), die den Aufwand etwas genauer wiedergeben (Cost Drivers).

3. Die Stufe nach dem Architektur Design (Post-Architecture Model): In dieser Phase sollte das Architektur Design vorliegen, folglich kann präziser geschätzt werden. Das COCOMO 2 Modell verwendet in dieser Stufe 17 multiplikative Kostenfaktoren (M) und 5 Faktoren, die den Exponenten B bestimmen. Als Maß für die Größe des Projekts können entweder Function Points oder LOC verwendet werden.

Als Einheit für den Aufwand werden Personen-Monate verwendet, wobei ein Personenmonat 152 Arbeitsstunden beinhaltet. Im den folgenden zwei Unterkapiteln werden die ersten 2 Modelle beschrieben.

Das Application Compositon Model

Grundlage für dieses Modell ist die Zahl der Object Points (Abschnitt *Object Points*). Der Aufwand in Personenmonaten (PM) errechnet sich wie folgt:

$$PM = \frac{(object\_points) * \frac{(100-\%reuse)}{100}}{PROD}$$

Bei *PROD* handelt es sich um die Produktivität der beteiligten Mitarbeiter (siehe Tabelle 5.4).

Fähigkeit der Entwickler	sehr gering	gering	normal	hoch	sehr hoch
PROD	4	7	14	25	50

Tabelle 5.4: Produktivität der Entwickler in NOP pro Monat

Das Early Design Model

Das Early Design Model und das Post-Architectural Model beruhen beide auf folgender Formel:

$$Aufwand = (\prod_i EM_i) * A * (Groesse)^B$$

Der Unterschied in beiden Modellen besteht in der Anzahl der Faktoren EM_i (Effort Cost Multiplier) und die Berechnung des Exponenten B. Die Größe des Projekts kann entweder in Function Points oder SLOC (Source Lines of Code) angegeben werden, wobei bei der Verwendung von Function Points (hier unadjusted Function Points) eine Umrechnung in SLOC stattfindet (siehe Tabelle 5.5). Eine ausführlichere Umrechnungstabelle mit Produktivitätsangabe findet man in [20]. A sollte nach Boehm den Wert 2.54 annehmen.

Der Exponent B (Project Scale Factor) errechnet sich aus folgender Formel:

$$B = 1.01 + \sum_{i=1}^{5} w_i$$

w_i sind die einzelnen Skalierungsfaktoren:

- *Precedence:* Spiegelt die Erfahrung der Organisation mit ähnlichen Projekten wieder. Sehr gering bedeutet, dass die Organisation über keine Erfahrung verfügt, besonders hoch bedeutet, dass der Organisation das Gebiet des Projekts völlig vertraut ist.
- *Development Flexibility:* Gibt den Flexibilitätsgrad im Entwicklungsprozess wieder. Sehr gering bedeutet, dass ein vom Kunden vorgeschriebener Prozess angewandt werden muss, sehr hoch bedeutet, dass der Kunde in dieser Hinsicht nur allgemeine Vorgaben gibt.
- *Architecture / Risk Resolution* : Gibt den Umfang der Risikoanalyse wieder.

Programmiersprache	LOC/UFP
ADA	71
Assembler	320
Assembler(Macro)	214
Basic (kompiliert)	91
Basic (interptretiert)	128
C	128
C++	29
Fortran 77	105
Java	53
Lisb	64
Maschinensprache	640
Modula 2	80
Pascal	91
Perl	21
Prolog	64
Smalltalk	21
Visual Basic 5	29
Visual C++	34

Tabelle 5.5: Umrechnung von UFP in SLOC

- *Team Cohesion:* Sehr gering bedeutet, dass das Team schlecht zusammenarbeitet bzw. die Interaktion sehr schwierig ist, sehr hoch bedeutet, dass die Zusammenarbeit reibungslos funktioniert.
- *Process Maturity:* Bewertet den Entwicklungsprozess nach CMM (Capability Maturity Model). CMM Level 1 bezeichnet einen eher unausgegorenen Entwicklungsprozess, CMM Level 5 bezeichnet einen ausgereiften Entwicklungsprozess [21].

Sämtliche Faktoren werden nach 6 Stufen gewichtet: Very Low (Wert = 5), Low (Wert = 4), Nominal (Wert = 3), High (Wert = 2), Very High (Wert = 1), Extra High (Wert = 0) ([14]). Der Wertebereich von B liegt also zwischen 1.01 und 1.26. Ein kleines Beispiel: Angenommen, alle Faktoren sind 0, dann ergibt sich für ein 100 KSLOC Projekt folgender Aufwand $Aufwand = 100^{1}.01 = 105PM$. Nehmen alle Faktoren den Wert 5 an, so erhält man einen Aufwand von 331 PM ($100^{1}.26 = 331$).

Die 7 Effort Cost Multiplier (EM_i):

- Produktzuverlässigkeit und - komplexität (Reliability and Complexity - RCPX)
- Geforderter Wiederverwendungsgrad (Reuse - RUSE)
- Schwierigkeitsgrad der Plattform (platform difficulty - PDIF)
- Mitarbeiterfähigkeiten (personnel capability - PERS)
- Erfahrung des Personals (personnel experience - PREX)

- Zeitplan (schedule - SCED)
- unterstützende Einrichtungen (facilities - FCIL)

Diese Werte können aus einer sechs Punkte Skala entnommen werden (z.B besonders hohe Komplexität = 6, sehr niedrige Komplexität = 1). Sie können auch aus den Faktoren des Post-Architecture Design abgeleitet werden. Mehr zu diesen Faktoren und dem Post-Architecture Design kann in [14] erfahren oder unter `http://sunset.usc.edu/research/COCOMOII/index.html`.

5.4 Zeitplanung

Die Aufgabe der Zeitplanung ist es, für jede der Aktivitäten ein Start- und End-Datum festzulegen. Der Prozess der Zeitplanung wird im Allgemeinen mehrmals durchlaufen, und zwar immer dann, wenn neue Daten vorliegen. Dabei kann es sich zum Beispiel um neue Anforderungen handeln, die neue Aktivitäten erfordern. Kurz, der Prozess wird immer dann wiederholt werden müssen, wenn sich die Daten der vorlaufenden Aktivitäten ändern (Ablaufplanung, Zeitabschätzung, Ressourcenplanung, siehe auch Abbildung 3.2). Um einen brauchbaren Terminplan zu entwickeln, benötigt man folgende Daten:

- Ablaufplanung der Aktivitäten (siehe Abschnitt 5.2)
- Zeitabschätzung (siehe Abschnitt 5.3)
- Verfügbarkeit der benötigten Ressourcen: Der Zeitplan muss auch den Zeitplan gewisser Mitarbeiter berücksichtigen (z.B. könnte ein Mitarbeiter in mehreren Projekten beteiligt sein oder mehreren parallelen Aktivitäten zugeordnet sein). Des weiteren müssen auch die Urlaubstage der Mitarbeiter berücksichtigt werden.
- Kalender: Arbeitsfreie Tage sollten nicht von vornherein im Zeitplan enthalten sein. Auch die Arbeitszeit der Mitarbeiter ist zu bedenken (z.B. normale Geschäftszeit, Mehrschichtbetrieb).
- Zeitvorgaben: Im Vorfeld festgelegte Termine müssen beachtet werden. Beispiele dafür sind Auslieferung eines Teilprodukts an eine beteiligte Firma, die Präsentation eines Prototypen (an einer Fachmesse) oder mit dem Kunden ausgehandelte Milestones (z.B. um den Projektfortschritt zu belegen). Auch die Lead u. Lag Zeiten sind hier zu berücksichtigen (siehe Abschnitt 5.2).
- Risikoplan

Wie aus der obigen Aufzählung hervorgeht, braucht man, um einen Terminplan zu erstellen, auch genaue Informationen über die vorhandenen Ressourcen und wann man diese wie lange einsetzen kann (Ressourcenkalender).

Für kleinere Projekte ist es durchaus möglich, GANTT Charts (siehe Abschnitt 5.2) zur Entwicklung des Terminplans zu verwenden. Für größere Projekte wird ein GANTT-Chart leicht unübersichtlich

5.5 Der optimale Plan

Eine beliebte Vorgehensweise bei der Entwicklung eines Terminplans ist es zuerst einen optimalen Plan zu entwerfen, der nur von den Aktivitäten, ihrer Abhängigkeiten und ihrer Dauer beeinflusst wird. Alle anderen Faktoren und Einschränkungen werden nicht beachtet (z.B. Ressourcen, vorgeschriebene Zeiten...). Hat man einen optimalen Plan entworfen, so geht man im nächsten Schritt daran, diesen Plan an die Restriktionen anzupassen.

Der optimale Plan wird folgendermaßen entworfen:

1. Ausgehend vom geplanten Beginn des Projekts startet man eine sogenannte Vorwärtsrechnung und erhält somit den frühestmöglichen Beginn und das frühestmögliche Ende einer jeden Aktivität.

2. Ausgehend vom geplanten Ende des Projekts startet man eine sogenannte Rückwärtsrechnung und erhält somit den spätesten Beginn und das späteste Ende einer jeden Aktivität.

Ein Beispiel: Angenommen wir haben 5 Aktivitäten eines Projekts A..E und deren Aufwand in Personentagen ermittelt (A ->5, B ->5, C ->6, D ->4, E ->5), und weiters nehmen wir an, das Projekt startet am 5.5.2003 und muss spätestens am 10.6.2003 abgeschlossen sein. B hängt von A ab (FS - Relation), C und D hängen von B ab (FS) und E hängt von C und D ab (FS) . Abbildung 5.6 zeigt das Ergebnis einer Vorwärtsrechnung. Bei diesem Diagramm handelt es sich um ein Activity on Arrow Diagramm (siehe Abschnitt 5.2), die Bezeichnung über den Pfeil bestimmt die Aktivität, die Knoten weisen die zugewiesenen Zeiten aus. Aktivität A beginnt frühestens am 5.5 (Eearly Start Date - ES) und endet frühestens am 9.5 (Early Finish Date - EF) , da deren Dauer 5 Personentage beträgt. Aktivität B kann infolge dessen frühestens am 12.5 beginnen und am 16.5 enden. Zu beachten ist, dass nur Arbeitstage gezählt werden, Wochenende und Feiertage werden also nicht mit eingerechnet (in diesem Beispiel sind die Wochenenden am 10/11, 17/18, 24/25, 31/1, 7/8 und am 29.5 ist Feiertag).

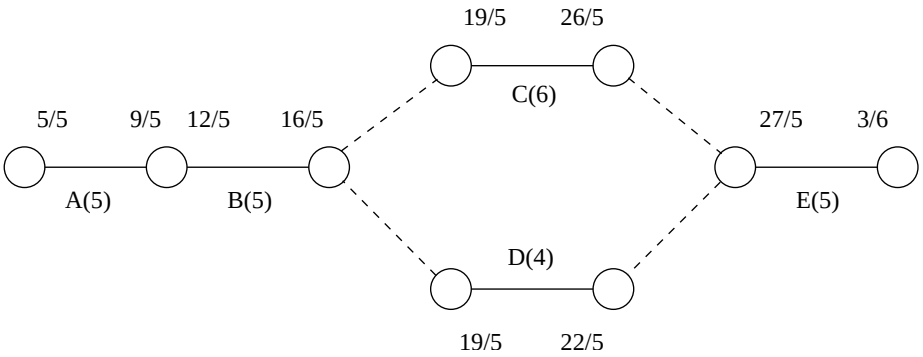

Abbildung 5.6: Beispiel einer Vorwärtsrechnung

Bei der Rückwärtsrechnung geht man vom anvisierten Ende des Projekts aus. Aktivität E dauert 5 Tage, das späteste Ende (Late Finsih Time - LS) ist

der 10.6, der späteste Beginn (Late Start Time - LS) ist der 4.6 (ein Wochenende liegt dazwischen). Später kann nicht begonnen werden, da sonst das Projektenddatum überschritten werden würde. Das späteste Ende der Aktivitäten C u. D ist 3.6. Abbildung 5.7 zeigt die Rückwärtsrechnung kombiniert mit der Vorwärtsrechnung, wobei die obere Zeile über jedem Knoten das Ergebnis der Vorwärtsrechnung enthält, die untere Zeile das Ergebnis der Rückwärtsrechnung. Aus diesem Diagramm kann man nun ablesen, wann eine Aktivität frühestens begonnen werden kann und spätestens begonnen werden muss, ohne das Projekt zu verzögern. Das Diagramm zeigt auch, wann eine Aktivität frühestens abgeschlossen werden kann und wann sie spätestens abgeschlossen werden darf, ohne die Projekt Deadline zu verpassen.

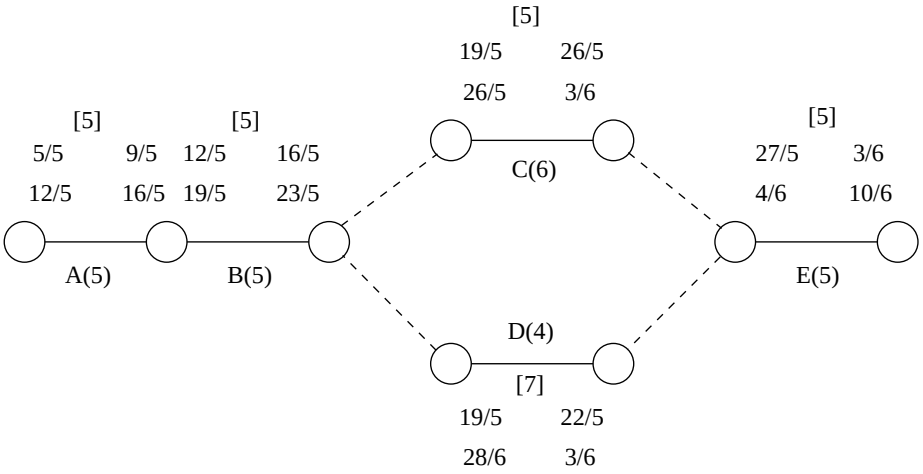

Abbildung 5.7: Beispiel einer Vorwärts-/Rückwärtsrechnung

Die Zahlen in den eckigen Klammern in Abbildung 5.7 bezeichnen den sogenannten *Float* oder *Slack*. Je größer diese Zahl, umso mehr Puffer hat man zwischen dem frühesten Beginn und dem spätesten Beginn einer Aktivität oder zwischen frühesten Beginn und spätestem Ende einer Aktivität. So liegen zwischen dem frühesten Beginn und spätesten Beginn der Aktivität A 5 Werktage (10/11 ist Wochenende) genau soviele Puffer-Werktage liegen zwischen dem frühest möglichen Beginn der Aktivität A und deren spätest möglichem Ende. Summiert man die Werte aller Slacks entlang eines Pfades, so bezeichnet der Pfad mit der niedrigsten Summe der Slacks (*Total Slack*) den *kritischen Pfad*. Ein kritischer Pfad beinhaltet all diejenigen Aktivitäten des Diagramms, deren Verzögerung als erstes auch die Gesamtdauer des Projekts verzögert (also A-B-C-E). Beträgt der gesamte Slack eines Pfades 0, so wirkt sich jede Zeitüberschreitung einer Aktivität gleichermaßen auf das Enddatum des Projekts aus (in dem hier gezeigten Beispiel hätte der gesamte Slack des kritischen Pfades dann Null, wenn die Endzeit des Projekts der 3.6 wäre). Pfade mit einem gesamten Slack von 0 sollten vermieden werden, da die Wahrscheinlichkeit, dass das Projekt zeitgemäß abgeschlossen werden kann, sehr gering ist. Die Ermittlung des kritischen Pfades und des Slacks ist daher für jedes Projekt empfehlenswert und bei umfangreicheren Projekten ohne die Methode der Vorwärts-/Rückwärtsrech-

nung nicht trivial (in diesem Fall sollte die Zeitplanung ohnehin softwaregestützt durchgeführt werden). Abbildung 5.8 zeigt eine alternative Darstellung von Abbildung 5.7.

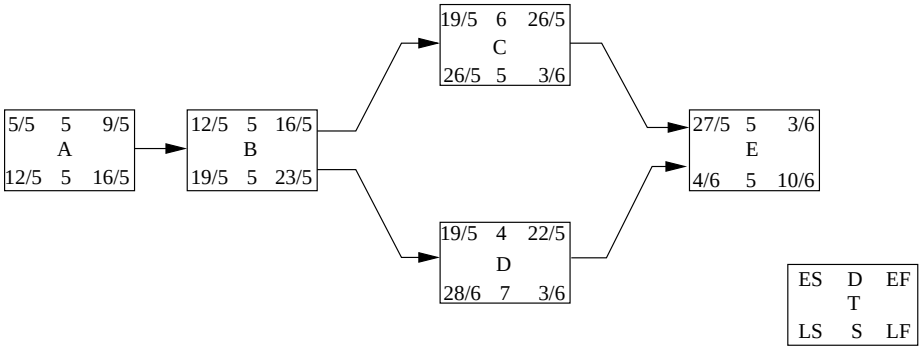

Abbildung 5.8: Eine alternative Darstellung der Vorwärts/Rüchwärtsrechnung

Nach jeder Verzögerung eines Tasks muss die Rechnung erneut durchgeführt werden. Abbildung 5.9 zeigt die Auswirkung einer Zeitüberschreitung von 5 Tagen der Aktivität A. Der gesamte Slack des Pfades A-B-C-D beträgt nun 0, jede Verzögerung entlang dieses Pfades wirkt sich unmittelbar auf den Abschluss des Projekts aus.

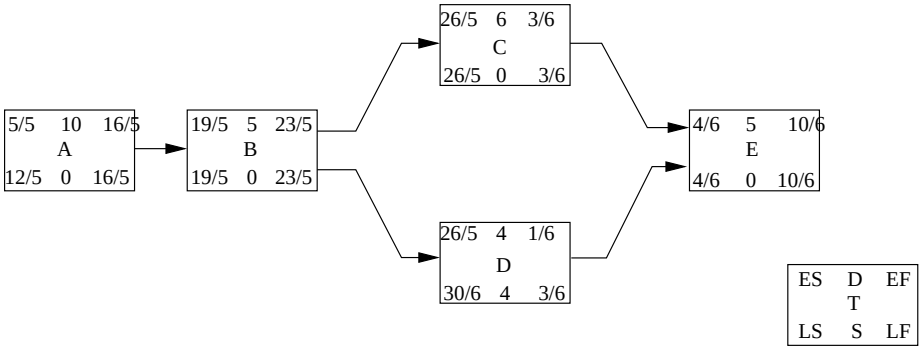

Abbildung 5.9: Ist eine Aktivität im Verzug, so beeinflusst diese den Slack aller von ihr abhängigen Aktivitäten

5.6 Kontrolle des Terminplans

Wie kontrolliert man den Zeitplan eines Projekts? Der erste Schritt ist, dass man einen Zeitplan hat und dieser so gestaltet wird, dass man erkennt, wann eine Aktivität als abgeschlossen gelten kann. Die Abschlussbedingungen sollten quantifizierbar sein. Ist eine Aktivität im Verzug, so muss man sich die Auswirkungen auf den Terminplan vor Augen halten und überlegen, wie man auf die Verzögerung reagiert. Die Gründe für einen Verzug sind vielfältig (falsche

Einschätzung, Krankheit, Mitarbeiterabgang, politische Entscheidung ...). Prinzipiell hat man 2 Möglichkeiten, auf eine Zeitüberschreitung einer Aktivität zu reagieren: Man kann das Personal für diese Aktivität aufstocken oder den Funktionsumfang kürzen. Bei der Erhöhung der Zahl der Mitarbeiter einer Aktivität ist folgendes zu beachten:

- Wird der Verzug spät erkannt, so kann eine Personalaufstockung unter Umständen den gegenteiligen Effekt haben, da man die Mitarbeiter häufig erst anlernen muss. Das Anlernen eines neuen Mitarbeiters bindet zusätzlich personelle Ressourcen der betroffenen Aktivität.

- Je mehr Mitarbeiter an einer Aktivität arbeiten, desto höher der Kommunikationsaufwand (die Produktivität des Einzelnen sinkt mit zunehmender Gruppengröße).

- Die Aktivität muss unter Umständen neu geplant werden, da mehr Mitarbeiter an ihr arbeiten werden. Nicht jede Aktivität lässt sich an mehr Mitarbeiter anpassen.

Ist eine Personalaufstockung nicht praktikabel, so bleibt der Ausweg, den Funktionsumfang zu beschneiden. Hier sollte sehr umsichtig vorgegangen werden, um die Qualität des Produkts nicht zu beeinträchtigen (zum Beispiel auf die geplanten Tests verzichten). Ist weder eine funktionale Reduktion des Produkts möglich, noch die Beteiligung zusätzlicher Mitarbeiter, so wird das Projekt aller Wahrscheinlichkeit nach den ursprünglich geplanten Endtermin überschreiten. Die Hoffnung, den Verzug einer Aktivität bei anderen Aktivitäten kompensieren zu können, wird sich nur selten erfüllen. Welche der Möglichkeiten man in Betracht zieht, hängt entscheidend vom Umfeld (von den Projektbeteiligten) ab. Hat zum Beispiel der Fertigstellungstermin höhere Priorität als der Funktionsumfang oder die Qualität des Produkts, wird man eine Beschneidung des Produkts leichter in Kauf nehmen.

Noch einige Worte zum Verzug einer Aktivität: Tritt ein grobes Problem auf (z.B. eine Gruppe von Mitarbeitern verlässt die Firma), so wird man alle Anstrengungen unternehmen, das Problem in den Griff zu bekommen. Auf eine schleichende Verzögerung (jeden Tag ein bisschen) hingegen reagiert man gern zögerlich, da die Hoffnung lebt, an einem anderen, besseren Tag schneller voranzukommen. Dazu ein Zitat von Frederick P. Brooks [13]:

> How does a project get to be a year late?
> ... One day at a time.

Literaturverzeichnis

[1] Bernd J. Madauss. *Handbuch Projekt Management, 6 Auflage*. Schäffer Poeschel, 2000

[2] Project Management Institute. *A Guide to the Project Management Body of Knowledge*. PMI, 2000 Edition

[3] P. Rinza. *Projekt Management*. Springer-Verlag Berlin Heidelberg, 1998

[4] Projektdefinition nach DIN 69901

[5] Christian H. Tauber. *Kostengünstiges Bauen*. Handelsblatt vom 9./10.7. 1982

[6] The Standish Group International. *The Chaos Report (1994)*. The Standish Group International, 1994

[7] The Standish Group International. *Chaos: A Recipe for Success*. The Standish Group International, 1999

[8] Barry Boehm. *A Spiral Model of Software Development and Enhancement* IEEE Computer, vol.21, #5, May 1988, pp 61-72

[9] Ian Sommerville. *Software Engineering*. Addison-Wesley 2001

[10] The Object Management Group. *Unified Modelling Language*. http://www.omg.org/technology/documents/formal/uml.htm

[11] W. Dröschel. *Das V-Modell 97*. Oldenburg 2000

[12] Tom DeMarco. *Software Projektmanagement*. Wolfram's Fachverlag 1989

[13] Frederick P. Brooks Jr. *The Mythical Man Month*. Addison Wesley 1995

[14] Barry Boehm et. al. *Cost Models for Future Life Cycle Processes: Cocomo 2.0*. Annals of Software Engineering 1995

[15] Albrecht A. J. *Measuring application development productivity*. SHARE/-GUIDE IBM Application Development Symposium 1979 (Kap. 23)

[16] Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison Wesley 2003.

[17] IFPUG. *IFPUG Counting Practices Manual, Release 4.1*, International Function Point User Group 1999.

[18] Jones C. *Software Assessments, Benchmarks and Best Practices*. Addison Wesley 2000.

[19] Kaufmann R. and R. Kumar *An Empirical Validation of Software Cost Estimation Models*. Stern School of Business Report, New York University, Jan. 1993.

[20] Jones C. *Programming Language Levels Release 8.2*. Software Productivity Research, Inc. 1996. Available at `http://www.theadvisors.com/langcomparison.htm`

[21] Paulk, M.C. und Konrad, M. et. al. *Capability Maturity Model, Version 1.1*. IEEE Software, 10(4), 18-27.(Kap. 25, 29)