Diplomarbeit

Universität Hamburg

Fachbereich Informatik

Arbeitsbereich Softwaretechnik

OPEN – eine Vorgehensweise zur flexiblen Softwareentwicklung in der Praxis?

Henrik Ortlepp

Betreuer:

Prof. Dr. Christiane Floyd

Prof. Dr. Bernd Neumann

Hamburg

Februar 2001

Inhalt

Inhalt 3

Abbildungen 5

Danksagungen 7

1. Einleitung 9

1.1. Problemstellung 9

1.2. Zielsetzung 11

1.3. Aufbau 12

1.4. Das begleitende Praxisprojekt 14

2. Flexibles Vorgehen in der Softwareentwicklung 17

1.5. Bestandteile eines Vorgehens in der Softwareentwicklung 17

2.2. Schematische und flexible Vorgehensweisen 25

2.3. Kriterien für eine flexible Vorgehensweise 34

3. Object-oriented Process Environment and Notation – OPEN 37

3.1. Überblick 38

3.2. Aufbau 42

3.2.1. Aktivitäten strukturiert in einem Lebenszyklusmodell 43

3.2.2. Konkrete handlungsleitende Aufgaben in Aktivitäten 52

3.2.3. Techniken zur Ausführung von Aufgaben 53

3.3. Flexibilität in Projekten mit OPEN 55

3.3.1. OPEN-Standardbeispiel 55

3.3.2. OPEN allgemein 57

4. Die Software-Expedition – Flexibilität im Fokus 61

4.1. Die Expeditionssicht 61

4.2. Ablauf einer Software-Expedition 65

4.3. Flexibilität in Software-Expeditionen 72

5. Ein flexibler Einsatz von OPEN in der Praxis 77

5.1. Besondere Einflußfaktoren in der Praxis 78

5.2. Ein Ansatz für eine passende OPEN-Instanz 80

5.2.1. Anforderungen / Herleitung der Prinzipien 80

5.2.2. Grundlegende Prinzipien 86

5.2.3. Das Lebenszyklusmodell 87

5.2.4. Die Aktivitäten des Lebenszyklusmodells 92

5.2.5. Dokumente und Vorlagen 95

5.3. Erfahrungen aus dem Einsatz 96

5.3.1. Erfahrungen mit dem Kunden 97

5.3.2. Erfahrungen mit Entwicklern und Management 98

5.4. Kritische Diskussion 101

6. Schluß 103

6.1. Zusammenfassung 103

6.2. Ausblick 106

Literatur 109

Anhang A – Erklärungen zur OPEN-Terminologie 117

Anhang B – Aufgaben aus OPEN 119

Anhang C – Techniken aus OPEN 121

Abbildungen

Abbildung 2-1 – Einflußfaktoren für das Vorgehen in der Softwareentwicklung 18

Abbildung 2-2 – Typische Bestandteile einer Vorgehensweise 23

Abbildung 2-3 – Ein Wasserfallmodell (nach Floyd, Züllighoven 1999) 24

Abbildung 2-4 – Vorgehensweisen im Spannungsverhältnis 29

Abbildung 2-5 – Instanziierung einer eigenen Vorgehensweise 31

Abbildung 3-1 – Gliederung des Lebenszyklusmodells in OPEN 41

Abbildung 3-2 – OPEN »contract-driven lifecycle model«, CDLM (aus Graham 1997, S. 46) 45

Abbildung 3-3 – Ein einfaches Lebenszyklusmodell für »responsibility-driven design« (aus Henderson-Sellers und Unhelkar 2000, S. 117) 47

Abbildung 3-4 – OPEN »firesmith instantiation« (aus Firesmith et al. 1998, S. 36) 49

Abbildung 3-5 – Lebenszyklusmodell des »use-case-driven approach« (aus Henderson-Sellers und Unhelkar 2000, S. 155) 50

Abbildung 3-6 – Gegenüberstellung von OPEN-Activities zu OPEN-Tasks 52

Abbildung 3-7 – Eine Roadmap zur Erstellung eines »deliverable« (aus Unhelkar 2000) 53

Abbildung 3-8 – Die Software-Expedition zwischen Schema und Flexibilität 60

Abbildung 4-1 – Exemplarisches Vorgehen einer Expedition (aus Mack 2001, S. 148) 62

Abbildung 4-2 – Vorgehensweise einer Software-Expedition 66

Abbildung 4-3 – Territorium der Anwendungsorganisation (aus Mack 2001, S. 143) 68

Abbildung 4-4 – Ablauf einer Software-Expedition (aus Mack 2001, S. 148) 69

Abbildung 4-5 – Die Software-Expedition zwischen Schema und Flexibilität 75

Abbildung 5-1 – Übersicht über das Lebenszyklusmodell 88

Abbildung 5-2 – Früher Entwurf des Lebenszyklusmodells 91

Danksagungen

Der Flexibilität und Unterstützung vieler Menschen, die ich hier noch einmal erwähnen möchte, ist es zu verdanken, daß diese Arbeit zum einen als Diplomarbeit an der *Universität Hamburg* zum anderen aber auch als gestaltender Beitrag für die Firma *CTH Consult Team Hamburg GmbH* entstehen konnte[1]. Als erstes möchte ich hier Julian Mack von der Universität Hamburg und Karen Paul von der CTH herzlich danken, die beide immer wieder zu kontroversen Diskussionen bereit waren und durch umfangreiche Korrekturen zur vorliegenden Form der Arbeit beitrugen.

Frau Prof. Dr. Christiane Floyd und Herrn Prof. Dr. Bernd Neumann gilt mein besonderer Dank für die flexible Betreuung meiner Arbeit sowie der Ermutigung, mein Vorhaben in der geplanten Form fortzusetzen. Sie haben mir weiterhin durch gezielte Kommentare zu wesentliche strukturellen und inhaltliche Ideen verholfen.

Bedanken möchte ich mich auch bei weiteren Kollegen der CTH, die mit mir Erfahrungen und Erlebnisse aus ihrem Projektleben teilten, so daß ich die Arbeit durch Erkenntnisse aus der Praxis erweitern konnte. Hier ist zunächst Mike Radzuweit zu nennen, der durch seine Zustimmung und extrem flexible Unterstützung von Beginn an, diese Arbeit erst möglich gemacht hat. Michael Bosselmann, Dr. Wolf-Peter Zank und Silke Rode danke ich vor allem für die ausgedehnten Diskussionen, die konstruktive Kritik sowie die praktische Umsetzung und das Austesten der Ideen im Projektalltag.

---

[1] Ich werde in der vorliegenden Arbeit die Rechtschreibung gemäß Duden (1986) verwenden.

1. Einleitung

Nach wie vor unterscheiden sich die zur Verfügung stehenden Vorgehensweisen zur Softwareentwicklung z.T. erheblich voneinander. Die vorliegende Arbeit betrachtet vor allem einen dieser Unterschiede im Detail – die Flexibilität – und konkretisiert die Ideen für eine flexible und praktikable Vorgehensweise in einer Anpassung des *Object-oriented Process, Environment and Notation* (OPEN). Dazu sollen zum einen Anregungen aus einem aktuellen Forschungsansatz, bei dem Flexibiltät eine entscheidende Rolle spielt, in die Arbeit einfließen. Zum anderen soll durch ein die Arbeit begleitendes Praxisprojekt die Anwendbarkeit überprüft und letztendlich gewährleistet werden.

1.1. Problemstellung

Spätestens mit Beginn der sog. Softwarekrise in den 60er Jahren wurde klar, daß sich die Entwicklung von Software zu einem so komplexen Prozeß entwickelt hatte, daß Hilfestellungen zum Vorgehen in der Softwareentwicklung und eine durchdachte Struktur dringend erforderlich wurden[2]. Zu den stetig wachsenden Anforderungen an Anwendungssoftware zählen heute Dinge wie[3]:

- Fachliche Relevanz
- Ökonomie
- Wartbarkeit und Skalierbarkeit
- Anpaßbarkeit
- Ergonomie
- Angemessenheit
- Portabilität
- Performanz
- Robustheit
- Zuverlässigkeit
- Sicherheit

Um diese immer wiederkehrenden Forderungen zu erfüllen, entstanden zunächst schematische Modelle, die den Prozeß der Softwareentwicklung strukturieren sollten. Von herausragender Bedeutung ist hier das phasenorientierte Wasserfallmodell[4], in dem ein Schema aus zeitlich aufeinanderfolgenden Phasen aufgestellt wird, das beschreibt, in welcher Art und Weise Software zu entwickeln ist.

---

[2] Vgl. Somerville (1995, S. 46).

[3] Vgl. Somerville (1995, S. 9) sowie Duden Informatik (1993, S. 655f).

[4] Vgl. Royce (1970).

Obwohl schematische Vorgehensmodelle nach wie vor häufig eingesetzt werden, bleibt gerade bei Projekten in dynamischem Umfeld die mangelnde Flexibilität ein harter Kritikpunkt. Es fehlt generell die Möglichkeit, Eigenarten des organisatorischen Umfelds, projektspezifische und veränderliche Rahmenbedingungen sowie die Erfahrungen der mitwirkenden Personen angemessen zu berücksichtigen. In phasenorientierten Vorgehensmodellen baut darüber hinaus jede Phase auf den Ergebnissen der vorherigen auf: Es ist im angestrebten Ablauf nicht vorgesehen, flexibel auf besondere Umstände zu reagieren oder die Ergebnisse einer abgeschlossenen Phase nachträglich zu ändern.

Die aus diesen Mißständen motivierten Alternativen zum Wasserfallmodell propagieren das zyklische Durchlaufen der Phasen in mehreren Iterationen[5]. Die Ergebnisse werden so mit jedem Durchlauf inkrementell weiterentwickelt; Veränderungen können in der entsprechenden Phase der nächsten Iteration eingearbeitet werden. Zwar stellen diese Vorgehensmodelle schon einen großen Fortschritt in Richtung Flexibilität dar, jedoch sind doch noch nicht alle Flexibilitätsanforderungen erfüllt: So liegt z.B. immer noch ein festes Schema zugrunde, das u.U. den Eigenarten des Projektes, der Arbeitsweise der Entwickler, der Dynamik der Anforderungen, dem Einsatzkontext oder der Geschwindigkeit der technischen Entwicklung nicht angemessen sein muß.

Inzwischen wird die Ansicht, daß es kein generelles Modell für alle Projekte geben kann, von einer immer größeren Anzahl von Autoren geteilt[6]. Dies kommt vor allem daher, daß sich Softwareprojekte im Projekt- und Organisationskontext – also in ihrer Organisationspolitik und -philosophie, ihren Mitarbeitern, den verwendeten Softwarewerkzeugen und Technologien etc. – zu sehr voneinander unterscheiden. Da hier unzählige Kombinationen vorstellbar sind, die jeweils eine eigene Berücksichtigung erfordern, erscheint es nicht sinnvoll, nach einem festen Modell zu suchen, das alle existierenden Projektformen angemessen unterstützt. Ein flexibles Vorgehen ist also dringend notwendig.

---

[5] Vgl. z.B. das Spiralmodell von Boehm (1988).

[6] Vgl. z.B. Henderson-Sellers (1998, S. 4): „Yet, it is generally believed that, however hard we seek it, a single, generally applicable process will remain an unattainable and illusionary Holy Grail of software engineering“ oder auch Goldberg (1995, S. 91) und Brooks (1986).

1.2. Zielsetzung

In der vorliegenden Arbeit werde ich zwei generelle Lösungsansätze für ein solches flexibles Vorgehen untersuchen, die den beschriebenen Problemen begegnen. Einen der Ansätze werde ich als Basis betrachten, um Ideen des anderen erweitern und so zumindest Grundgedanken für eine flexible Vorgehensweise entwickeln, die ich in einem Praxisprojekt parallel zur Arbeit einsetze und auf ihre Tauglichkeit untersuche.

Ein genereller Lösungsansatz ist, sich grundsätzlich vom Ansatz der fest vorgegebenen, schematischen Vorgehensmodelle zu lösen und neuartige, flexiblere Vorgehensweisen zu entwickeln. Hier geht es dann nicht mehr darum, den Entwicklungsprozeß in vorgegebene Phasen oder Aktivitäten einzuteilen, in denen genau festgelegt ist was, wann, wie und von wem zu tun ist. Vielmehr werden die Erfahrungen und die Eigenverantwortlichkeit der Teammitglieder stärker in den Vordergrund gerückt und durch Beschreibung genereller Vorstellungen zur Softwareentwicklung unterstützt. Diese Vorstellungen können sich als Leitideen, wie beispielsweise im Ansatz *Software-Technik für Evolutionäre Partizipative Systementwicklung* (STEPS)[7], als Werte und Praktiken, wie im *Extreme Programming* (XP)[8] oder als Sichtweise der Softwareentwicklung, wie in der *Software-Expedition*[9], darstellen.

Ein zweiter Ansatz besteht in dem Versuch, die alten Vorgehensmodelle so anzupassen, daß sie den dynamischen Randbedingungen gerecht werden und flexible Softwareentwicklung ermöglichen. Dazu reicht es allerdings i.A. nicht aus, viele (im obigen Sinne unflexible) schematische Vorgehensmodelle zur Verfügung zu stellen, aus denen für jedes Projekt je ein passendes Modell ausgewählt werden kann; Veränderungen, die während des Projektverlaufs eine flexible Anpassung des Vorgehens selbst erfordern, würden beispielsweise nicht berücksichtigt werden. Außerdem bleibt das generelle Problem, daß zwar nicht mehr ein einziges, nun aber mehrere Vorgehensmodelle alle möglichen Projektformen berücksichtigen müßten, um das passende Modell für jedes spezifische Projekt zur Verfügung stellen zu können.

---

[7] Vgl. Floyd (1994a) sowie Floyd (1994b).

[8] Vgl. Beck (1999).

[9] Vgl. Kapitel 3 sowie Mack (2001).

Vielmehr bietet es sich an, Umgebungen zur Definition von Vorgehensweisen zu beschreiben, die Konstrukte anbieten, um eine eigene, spezifische Vorgehensweise zu erstellen. Zu den bekannten Lösungen zählen hier der *Unified Process* (UP), der in einer Weiterentwicklung der Firma Rational zum *Rational Unified Process* (RUP) ausgebaut wurde[10], sowie *Object-oriented Process, Environment and Notation* (OPEN)[11]. Hier werden ausführlich Aktivitäten beschrieben, die typischerweise in der Softwareentwicklung anfallen. Zusätzlich gibt es Hinweise, welche dieser Aktivitäten wie strukturiert werden sollten und welche Besonderheiten auftreten können. Zum Einsatz muß aber aus den gegebenen Beschreibungen sowie ggf. zusätzlichen eigenen Elementen eine eigene an das konkrete Projekt angepaßte Vorgehensweise zusammengestellt werden.

In der vorliegenden Arbeit werde ich mich vor allem mit dieser zweiten Lösungsmöglichkeit beschäftigen und dabei der Frage nachgehen, inwieweit die Anforderungen an die Flexibilität und den Nutzen in der Praxis durch solche Umgebungen zur Definition von Vorgehensweisen ausreichend berücksichtigt werden können. Konkret wird dabei OPEN im Zentrum meiner Untersuchungen stehen: Ich werde Erfahrungen aus dem Einsatz von OPEN mit Betonung auf Flexibilität und Praxisnutzen beschreiben und auch Gedanken aus dem Ansatz der Software-Expedition einfließen lassen. OPEN wurde hier aus zwei Gründen ausgewählt: Zum einen bietet OPEN besonders gute Anlagen, die geforderte Flexibilität zu unterstützen. Zum anderen habe ich die Gelegenheit, die in der vorliegenden Arbeit entwickelten Vorstellungen in einem Projekt in der Praxis zu erproben und so die theoretische Auseinandersetzung mit dem Thema um praktische Erfahrungen zu ergänzen.

1.3. Aufbau

Um eine Auseinandersetzung der verschiedenen Ansätze zu ermöglichen, entwickele ich im zweiten Kapitel Anforderungen an eine flexible Vorgehensweise mit praktischem Nutzen. Dazu werde ich zunächst abgrenzen, was ich im Rahmen der vorliegenden Arbeit unter den Begriffen »Vorgehensweise« und »Flexibilität« in der Softwareentwicklung verstehe. Diese Anforderungen an die Flexibilität sowie die dargelegten Kriterien bilden dann die Grundlage für die in den folgenden Kapiteln durchgeführten Untersuchungen.

---

[10] Vgl. Kruchten (1999).

[11] Vgl. Graham et al. (1997) sowie Kapitel 4.

Im dritten Kapitel werde ich mit OPEN eine Umgebung zur Definition von Vorgehensweisen einführen, bei der die Umsetzung der technischen Unterstützung bei der Entwicklung von Software im Vordergrund steht. Zunächst sollen hier Prinzipien, Ziele und Aufbau von OPEN vorgestellt werden. Im Anschluß daran werde ich OPEN vor allem im Hinblick auf die im zweiten Kapitel aufgestellten Kriterien an flexible Softwareentwicklung kritisch untersuchen. Dabei werde ich auch auf die Frage aus dem Titel dieser Arbeit eingehen, indem ich darlege, welche Aspekte in OPEN für eine flexible Vorgehensweise dienlich sind und welche Mängel bestehen, um dynamische Projekte tatsächlich angemessen zu unterstützen.

Mit der Software-Expedition gebe ich im vierten Kapitel einen Überblick über einen Ansatz, der sich von den Vorstellungen der klassischen Alternativen wie dem Wasserfallmodell löst und einen starken Fokus auf Erfahrung und Flexibilität legt. Neben der Beschreibung der generellen Prinzipien und Ziele, der Sichtweise von Softwareentwicklung als Expedition und dem Aufbau der Vorgehensweise selbst, werde ich die zentralen Argumente und Gedanken zur Flexibilität in der Softwareentwicklung herausstellen und zur Weiterverwendung im folgenden Kapitel aufbereiten. Die Untersuchung der Software-Expedition scheint mir für die vorliegende Arbeit vor allem deswegen so wertvoll, weil sie ständige Veränderungen als typisch für die Entwicklung moderner Softwaresysteme ansieht und als Konsequenz Flexibilität im Vorgehen explizit adressiert und Lösungen als Konsequenz anbietet.

Unter Einbeziehung der im vierten Kapitel gewonnen Erkenntnisse bezüglich der Flexibilität beschreibt das fünfte Kapitel Ansätze für eine OPEN-konforme Vorgehensweise, die für einen bestimmten Projekttyp flexible Softwareentwicklung ermöglichen soll. In diese OPEN-Instanz sollen auch die praktischen Erfahrungen einfließen, die bei der Entwicklung und Anwendung der Vorgehensweise in dem Projekt parallel zur Arbeit gemacht wurden (vgl. auch den folgenden Abschnitt 1.4).

Neben einer Zusammenfassung und Bewertung der Ergebnisse findet sich im sechsten Kapitel ein Ausblick dazu, wie die im vorigen Kapitel vorgestellte Vorgehensweise weiterentwickelt und durch andere Perspektiven ergänzt werden kann.

Bevor ich zum zweiten Kapitel übergehe, soll im folgenden Abschnitt das bereits angesprochene Praxisprojekt noch kurz beschrieben und charakterisiert werden.

1.4. Das begleitende Praxisprojekt

Im Rahmen der Arbeit hatte ich die Gelegenheit, meine Gedanken und die Ansätze für eine Vorgehensweise in einem Softwareentwicklungsprojekt der Firma ***CTH Consult Team Hamburg GmbH*** auszuprobieren und Erfahrungen aus der Praxis einfließen zu lassen[12]. Dieses Projekt läßt sich nach Goldberg[13] mit im weiteren Verlauf beschriebenen Einschränkungen als »Neuimplementierung eines Altsystems« (Goldberg: »Legacy-rewrite project«) bezeichnen. Der Kunde möchte dabei die Aufnahme und Abwicklung von Leasing-Verträgen durch eine neue Software unterstützen, da sich die zur Zeit genutzten Programme in einem nicht mehr wartbaren oder erweiterbaren Zustand befinden, auf veralteten Technologien aufbauen und den fachlichen Anforderungen nicht genügen. In einem ersten Schritt soll der Programmteil realisiert werden, der sich mit der softwareunterstützten Erfassung und Bearbeitung der Leasing-Anträge befaßt. Im Laufe der parallel zur vorliegenden Arbeit durchgeführten Vorstudie (s.u.) hat sich herausgestellt, daß zusätzlich auch mit der Realisierung eines Teilsystems zur Kalkulation noch während der Vorstudie sowie eines Internet-Angebots gleich im Anschluß begonnen werden sollte.

In weiteren Ausbaustufen, sollen sukzessive weitere Elemente der Softwareunterstützung erneuert werden. Somit war eine weitere projektspezifische Aufgabe, daß neu entwickelte Teilsysteme zunächst mit den verbleibenden Altsystemen kompatibel sein müssen. Gleichzeitig sollen sie aber auf aktueller Technologie basieren und die Anbindung an Neuentwicklungen der restlichen Software ohne gravierende Umstrukturierung ermöglichen. Aufgrund der Komplexität des Systems schied der Gedanke einer kompletten Neuentwicklung (alle Systemteile in einem Schub) von vorne herein aus; denn zum einen war der Aufwand zur Entwicklung nicht überschaubar[14] und zum anderen war auch auf Kundenseite nicht klar, ob die Mittel zur Sanierung des kompletten Systems verfügbar sein werden. Ergebnis der ersten Stufe mußte also ein voll funktionsfähiges Teilsystem mit Einbeziehung der alten Systeme sein.

Der hohe Realisierungsaufwand kommt unter anderem auch dadurch zustande, daß neben Nachbildung des Altsystems auf neuer technologischer Basis auch einige neue Funktiona-

---

[12] Vgl. CTH (2001).

[13] Vgl. Goldberg (1995) sowie Abschnitt 2.1.

[14] Schon die Schätzung zur Realisierung der zur Integration in den Produktivbetrieb benötigten Funktionen ergab einen Aufwand von mehr als 2000 Personentagen.

litäten aufgenommen und realisierte fachliche Abläufe optimiert werden sollen. Vor allem die Übersichtlichkeit der Benutzungsschnittstelle ist in der verfügbaren Software nicht akzeptierbar, aber wegen der Komplexität der Eingabemasken und benötigten Daten auch nur sehr schwer zu optimieren[15].

Bei dem größten Teil des Projektes, der im Rahmen dieser Arbeit begleitet und untersucht werden konnte, handelte es sich zunächst um eine Vorstudie. Obwohl schon während der Vorstudie mit der Konzeption und Realisierung eines eigenständigen und zentralen Teils des Systems begonnen wurde[16], machten es politische Gründe unvermeidbar, auf Wunsch des Kunden zunächst eine ca. zwei- bis drei-monatige Vorstudie inklusive grobem Zeit- und Kostenrahmen für das bevorstehende Projekt durchzuführen. Dabei wurden schon Anforderungen des Kunden in Gesprächen aufgenommen, um sich eine Vorstellung von dem zu realisierenden System zu verschaffen. Der Fokus lag aber deutlich auf der Analyse von Risiken und Chancen, der Argumentation für eine iterative Vorgehensweise bei der eigentlichen Implementierung, einer generellen Planung des Zusammenwirkens der Teillösungen sowie der Systemarchitektur des Kernsystems und nicht auf der Erstellung eines umfangreichen Pflichtenheftes. Zusätzlich hat sich das Projektteam dazu entschlossen, einfache Prototypen zu erstellen, um zwei großen Risiken bereits innerhalb der Vorstudie zu begegnen[17].

---

[15] Hier zur Orientierung ein paar grobe Zahlen: Das Altsystem beinhaltet ca. 40 komplexe Dialoge, die mit durchschnittlich 50 Oberflächenelementen (wie statischen Texten, Eingabefeldern, Karteireitern, Auswahlboxen, Listen, Tabellen, etc.) bestückt sind.

[16] Hierbei handelte es sich um die zugrundeliegende Kalkulation, die u.a. die Realisierung finanzmathematischer Berechnungen beinhaltete und so implementiert werden sollte, daß sie auch mit anderen Systemen nutzbar ist. Diese Entwicklung konnte auch über die Vorstudie hinaus betreut werden.

[17] Dabei handelte es sich zum einen um eine Visualisierung des Grundkonzeptes für die neue Benutzungsschnittstelle, die eine Vereinfachung der komplexen Dialoge bewirken sollte und zum

anderen um eine Performanzmessung, die zur Verifikation der neuen verteilten Systemarchitektur und der Anbindung an das Internet notwendig wurde.

2. Flexibles Vorgehen in der Softwareentwicklung

In diesem Kapitel definiere ich die zentralen Begriffe meiner Arbeit und erstelle eine Basis zur Untersuchung verschiedener Vorgehensweisen im weiteren Verlauf. Dazu sollen die Vor- und Nachteile von klar definierten und schematischen im Gegensatz zu flexiblen Vorgehensweisen aufgezeigt und die wesentlichen Aspekte der geforderten praxisrelevanten Flexibilität herausgestellt werden. Am Ende dieses Kapitels werde ich Kriterien aufstellen, anhand derer in den folgenden Kapiteln geprüft werden soll, inwieweit die ausgewählten Vorgehensweisen flexible Softwareentwicklung in der Praxis ermöglichen, und ob sich das Spannungsverhältnis zwischen der Forderung nach einer definierten Vorgehensweise auf der einen und der gewünschten Flexibilität auf der anderen Seite auflösen läßt.

1.5. Bestandteile eines Vorgehens in der Softwareentwicklung

In diesem Abschnitt werde ich grundlegende Begriffe des Vorgehens in der Softwareentwicklung für die vorliegende Arbeit klären. Beim Vorgehen in der Softwareentwicklung muß unterschieden werden zwischen dem theoretischen Vorhaben und der tatsächlichen Praxis. Typischerweise wird vom Projektmanagement das theoretische Vorhaben in einer »Vorgehensweise« beschrieben (oder es wird eine bereits existierende Vorgehensweise angepaßt bzw. einfach übernommen), nach der sich das tatsächliche Vorgehen in der Praxis richten soll. Dementsprechend definiere ich die Begriffe Vorgehen und Vorgehensweise wie folgt:

> ***Vorgehen*** **-** Art, in der Anwendungssoftware in der Praxis tatsächlich konzipiert, entwickelt und betreut wird.
>
> ***Vorgehensweise*** **-** Schriftlich fixierte oder mündlich tradierte Beschreibung, die als Vorlage für das Vorgehen dient und als solche Teil des Vorgehens selbst wird.

Leider hat sich hierzu in der Literatur der Softwaretechnik bislang kein einheitlicher Begriffsraum herausgebildet. Begriffe wie z.B. »method«, »process« und »process model« werden von den unterschiedlicher Autoren mit verschiedener Bedeutung verwendet und finden in den Ausdrücken der deutschen Literatur wie »Vorgehen«, »Vorgehensweise«,

»Vorgehensmodell«, »Prozeß« und »Ablauf« keine eindeutige Entsprechung[18]. Daher scheint mir eine entsprechende Definition notwendig[19].

Auf das Vorgehen in der Praxis wirken neben der Vorgehensweise noch andere Faktoren ein. Die wesentlichen Einflußfaktoren sind in Abbildung 2-1 dargestellt.

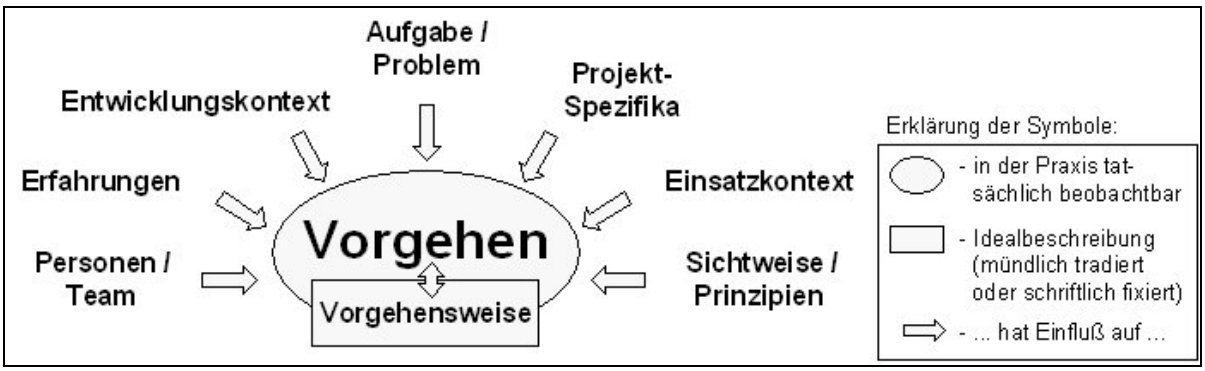

Abbildung 2-1 – Einflußfaktoren für das Vorgehen in der Softwareentwicklung

Während sich die vorliegende Arbeit vor allem mit der Auswahl und Anpassung von Vorgehensweisen beschäftigt und die Bedeutung von zugrundeliegenden Sichtweisen und Prinzipien beleuchtet, so dürfen die anderen dargestellten Einflußfaktoren doch in keinem Fall unberücksichtigt bleiben. Sie sollten vielmehr wesentliche Grundlage für die Auswahl der Vorgehensweise sein, da sie in den meisten Projekten vorgegeben sind und sich i.A. nur mit sehr großem Aufwand verändern lassen. Die in Abbildung 2-1 dargestellten Begriffe zeigen die wesentlichen Faktoren auf; diese sind aber nicht klar voneinander abzugrenzen, sondern gehen ineinander über und beeinflussen sich gegenseitig: So sind Erfahrungen und Sichtweisen an Personen gebunden, sie werden aber auch in bestimmten Projekten mit eigenem Entwicklungs- und Einsatzkontext entwickelt. Andererseits sind die

---

[18] Vgl. z.B. Jacobson (1992, S. 32f), Graham et al. (1997, S. 17ff), Humphrey (1987, S. 39), Eintrag „Methode“ und „Prozeß“ im Duden Informatik (1993, S. 405 bzw. S. 559), Floyd/Züllighoven (1999, S. 771ff), Sommerville 1995 oder Derniame (1999, S. 11).

[19] Auch im Hinblick auf die folgenden Kapitel (speziell Kapitel 4) soll hier noch angemerkt werden, daß sich dieses Verständnis vom Begriff „Vorgehensweise“ von der Definition in Mack (2001) unterscheidet: Während in der vorliegenden Arbeit Beschreibungen jeglicher Form als Vorgehensweisen aufgefaßt werden, trennt Mack Vorgehens*weisen*, in denen „die in der Praxis anzutreffende Rolle und Bedeutung von Erfahrungen und Lernprozesse in den Vordergrund“ (Mack 2001, S. 165f) gestellt werden, von Vorgehens*modellen*, die auf „theoretischer Modellbildung und normativen Vorschriften“ (ebenda) beruhen. Diese Trennung läßt sich für die in der vorliegenden Arbeit betrachteten Aspekte allerdings nicht eindeutig ziehen; zur Unterscheidung von Vorgehensweisen, die als feste Vorschriften von solchen, die als flexible Ressourcen aufzufassen sind, werde ich daher weiter unten qualifizierende Adjektive einführen (vgl. Abschnitt 2.2).

Personen wiederum ein wesentlicher Bestandteil des Entwicklungs- und des Einsatzkontextes. Diese komplexen Einflüsse der gezeigten Faktoren untereinander sind nicht dargestellt.

Im Folgenden werde ich kurz auf die verschiedenen Einflußfaktoren eingehen:

Personen

Daß die an einem Projekt unmittelbar beteiligten *Personen* (das *Team*) das Vorgehen in der Softwareentwicklung beeinflussen und daher berücksichtigt werden sollten, ist zwar offensichtlich, wird aber häufig bei der Beschreibung von Vorgehensweisen außer Acht gelassen. Personen haben individuelle Auffassungen, Perspektiven[20], Arbeitsweisen, Fähigkeiten, Eigenschaften und Vorstellungen, die an die Vorgehensweise durchaus verschiedene, individuelle Anforderungen stellen[21]. Neben dem Team der Entwickler sind hier auch Personen außerhalb der Entwicklungsorganisation wie z.B. Anwender, Auftraggeber und ggf. Mitarbeiter der Softwareabteilung in der Anwendungsorganisation gemeint.

Erfahrungen

Ein Aspekt, der mit den beteiligten Personen eng verzahnt ist, sind die *Erfahrungen*, die in früheren Projekten gewonnen wurden. Diese Erfahrungen fließen oft auch unbewußt in das Vorgehen mit ein: Es wird immer versucht werden, bewährte Arbeitsweisen auch in neuen Projekten zu etablieren; ähnliche Fehler sollen nach Möglichkeit nicht wiederholt werden. Daher sollte es in der Vorgehensweise auch Mechanismen geben, Erfahrungen einzubeziehen. Auch hier geht es um die Erfahrungen aller am Projekt beteiligten Personen – auf Entwicklerseite genauso wie auf Anwenderseite.

Entwicklungskontext

Zusätzlich zu den individuellen Eigenschaften und Erfahrungen, gilt es den *Entwicklungskontext* zu berücksichtigen. Darunter verstehe ich die technologische, soziale und organisationale Umgebung, in der die Systementwicklung stattfindet. Dazu gehören also auch die *Tools und Technologien* (z.B. Textverarbeitungsprogramme, Programmiersprachen, komplette Entwicklungsumgebungen oder ein Intranet, aber auch einfachere Dinge wie Karteikarten), die bei der Softwareentwicklung zur Verfügung stehen.

---

[20] Vgl. Floyd (1994A, S. 33f) zur Perspektivität in der Softwareentwicklung.

Aufgabe / Problem

Entwicklungs- und Einsatzkontext werden durch die gestellte *Aufgabe* bzw. das zu lösende *Problem* verbunden. Diese grundlegende Aufgabenstellung kann sehr verschieden sein. In der vorliegenden Arbeit werde ich mich aber auf die Entwicklung von großer, dedizierter und objektorientierter Anwendungssoftware und auf Vorgehensweisen, die nicht für eine spezielle Situation entworfen wurden, sondern möglichst in mehreren Projekten und Organisationen verwendbar sind, beschränken.

Große Software bezeichnet hier in Anlehnung an Floyd und Züllighoven Systeme, die aufgrund ihrer Komplexität für sich genommen nicht verständlich sind. „Sie [große Software; d. Verf.] besteht aus Programmen und umfangreichen definierenden Texten, deren Konsistenz schwer zu gewährleisten ist, weil sie Sichtweisen verschiedener Beteiligter widerspiegeln, die sich im Laufe der Zeit ändern." (Floyd/Züllighoven 1999, S. 764) Auch die Definition von *Anwendungssoftware* werde ich für diese Arbeit aus derselben Quelle übernehmen:

> „Anwendungssoftware ist ein Mittel zum Zweck, um fachliche Aufgaben in einem oder mehreren Anwendungsbereichen zu erledigen. [...] Anwendungssoftware modelliert einen *Gegenstandsbereich* der realen Welt und orientiert sich an einem *Einsatzkontext*, der fest sein kann (dedizierte Systeme) oder allgemein gehalten wird (Standardsoftware)." (Floyd / Züllighoven 1999, S. 764)

Objektorientierung setzt sich in der Praxis immer mehr als gängige Lösung zur Entwicklung in Neuprojekten durch. Ich werde in der vorliegenden Arbeit nicht weiter auf Objektorientierung eingehen, setze aber voraus, daß es sich bei den Projekten, die durch die beschriebenen Vorgehensweisen unterstützt werden sollen, um rein objektorientierte Projekte oder zumindest gemischte Projekte mit hohem objektorientierten Anteil handelt. Dies liegt neben den klassischen Vorteilen des objektorientierten Ansatzes auch darin begründet, daß es sich bei dem die Arbeit begleitenden Projekt um eine rein objektorientierte Entwicklung handelt und auch meine sonstigen praktischen Erfahrungen bisher aus der Entwicklung objektorientierter Systeme stammen. Zur Erklärung und Diskussion objektorientierter Entwicklung sei hier auf Standardwerke wie Gamma et al. (1995) oder Jacobson (1992) verwiesen.

---

[21] Vgl. hierzu insbesondere Kapitel 4 sowie Mack (2001).

Projektspezifika

Auch in zwei Projekten des gleichen Projekttyps gibt es so viele *Projektspezifika*, die zu beachten sind, daß letztendlich jedes Projekt zur Entwicklung von Anwendungssoftware als Einzelfall gesehen werden muß[22]. Solche Spezifika sind beispielsweise die Art der Abrechnung (vgl. Kapitel 5), die politische Situation zwischen den Organisationen von Kunden und Entwicklern oder die technologischen Restriktionen. Außerdem sind generell verschiedene Projekttypen vorstellbar. Goldberg (1995, S. 91) unterscheidet beispielsweise zwischen:

- *First-of-its-kind project* (Neuentwicklung);
- *Variation-on-a-theme project* (Modifikation eines bestehenden Systems);
- *Legacy-rewrite project* (Neuimplementierung eines Altsystems);
- *Creating reusable assets project* (Erstellen wiederverwendbarer Teillösungen) und
- *System enhancement or maintenance project* (Weiterentwicklung oder Wartung eines bestehenden Systems).

Einsatzkontext

Der *Einsatzkontext* (vgl. Abbildung 2-1) beschreibt das fachliche, soziale, organisationale und technologische Umfeld, in dem das zu entwickelnde System eingesetzt werden soll.

> „Bei ihrer Entwicklung [Anm.: der Entwicklung von Anwendungssoftware] ist auch die Gestaltung der Interaktion zwischen den Anwendungsfachleuten und dem eingesetzten Anwendungssystem zu beachten." (Floyd/Züllighoven 1999, S. 764)

Generell sollte danach gestrebt werden, daß sich Entwicklungs- und Einsatzkontext möglichst weitgehend entsprechen, damit der Transfer von dem einen in den anderen Kontext die Entwicklung nicht zusätzlich erschwert. Die Verschmelzung von Entwicklungs- und Einsatzkontext läßt sich durch verschiedene Maßnahmen unterstützen[23]: Das Team kann sich beispielsweise zusätzlich zu Personen der Entwicklungsorganisation auch aus Anwendern oder Entwicklern der Anwendungsorganisation zusammensetzen und kurze Iterati-

---

[22] Vgl. Mack (2001, S. 118).

[23] Vgl. hierzu auch Kapitel 5 sowie Beck (1999, S. 161f).

onszyklen sowie Prototypen unterstützen das gemeinsame Verständnis vom zu entwickelnden Zielsystem.

Sichtweisen / Prinzipien

Für das Vorgehen in der Softwareentwicklung selbst sind generell verschiedene *Sichtweisen* vorstellbar, die vor dem Hintergrund von Erfahrungen und Wertvorstellungen entstehen[24]. Hier gibt es beispielsweise die vielen Vorgehensweisen zugrundeliegende Produktionssicht, die Floyd (1994a) wie folgt charakterisiert:

> „Ich verwende im folgenden den Begriff *Produktionssicht*, um die idealisierenden Grundannahmen des Software-Engineering über die Konstruktion von Software zusammenzufassen: Softwareentwicklung beruht auf vorgegebenen Problemen mit fest definierbaren Anforderungen: der Herstellungsprozeß ist anhand von Prozeßmodellen formalisierbar; der Einsatzkontext von Software kann ausgeklammert werden." (Floyd 1994a, S. 29)

Da die Existenz solcher vorgegebener Probleme und fest definierbarer Anforderungen immer mehr angezweifelt wird, schlägt Floyd vor, anstelle der Produktionssicht die Design-Sicht zu verwenden: Die Design-Sicht betrachtet „die Softwareentwicklung als situativen Prozeß wechselseitigen Lernens zwischen den Beteiligten und behandelt Softwareentwicklung und -einsatz in veränderlichen offenen Kontexten"[25]. Eine Erweiterung der Design-Sicht stellt die Expeditionssicht von Mack (2000a) dar, die Softwareentwicklung am Leitbild der Expedition orientiert (vgl. Kapitel 4).

Zusammen mit den spezifischen Projektgegebenheiten und eingebettet in den organisationellen Rahmen ergeben sich aus der Sichtweise eine Reihe von *Prinzipien*[26], die im gesamten Vorgehen und auf verschiedenen Ebenen grundlegend für Entscheidungen sind. Ein solches Prinzip könnte dabei beispielsweise konsequent objektorientierte Entwicklung zusammen mit einem bestimmten Verständnis davon, was Objektorientierung ausmacht,

---

[24] Vgl. Züllighoven (1998, S. 10).

[25] Vgl. Floyd (1994a, S. 29).

[26] Vergleichbar sind die Prinzipien auch mit den von Goldberg (1995) erwähnten Maxime (»maxims«), die generelle Überzeugungen beschreiben, welche Aspekte eines Projektes kritisch für den Erfolg sind.

sein. Ein weiteres mögliches Prinzip ist, die Bedeutung von Erfahrungen und Individuen über die strikte Befolgung einer fest vorgegebenen Vorgehensweise zu stellen (vgl. hierzu auch Kapitel 4 und 5).

Vorgehensweise (typische Bestandteile)

Die Bewältigung der hier beschriebenen Einflußfaktoren ist eine wesentliche und sehr komplexe Aufgabe der Vorgehensweise. Daher wird häufig versucht, innerhalb der Vorgehensweise Strukturen zu etablieren, um übersichtlich sowie auf verschiedenen Ebenen und mit unterschiedlichem Fokus eine Berücksichtigung der Einflußfaktoren zu bewirken. Abbildung 2-2 stellt einige typische strukturelle Elemente dar. Im verbleibenden Teil dieses Abschnitts werde ich diese strukturellen Elemente kurz beschreiben.

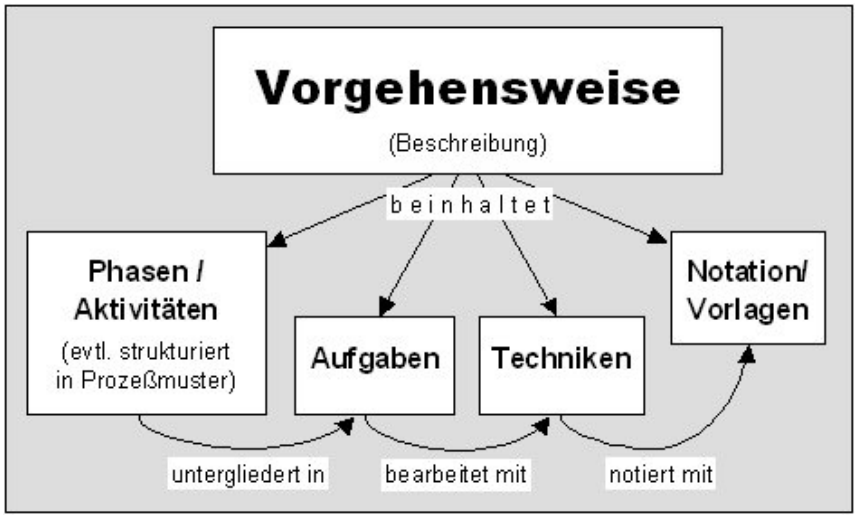

Abbildung 2-2 – Typische Bestandteile einer Vorgehensweise

Phasen

Eine mögliche Struktur bietet die Unterteilung in *Phasen* wie sie von Phasenmodellen propagiert wird (vgl. Abbildung 2-3). Ich werde in der vorliegenden Arbeit den Begriff Phase im Sinne von Oestereich (1999) verstehen als „eine zeitliche Gliederung des Entwicklungsprozesses [...] Jede Phase erfüllt einen definierten Zweck und führt zu definierten Ergebnissen.“[27] Wesentlich ist hier also vor allem die zeitliche Trennung der verschiedenen Phasen; Phasen werden in der Regel sequentiell bearbeitet, und Rückgriffe auf abgeschlossene Phasen sind nur in Ausnahmefällen möglich.

[27] Vgl. Oestereich (1999, S. 96).

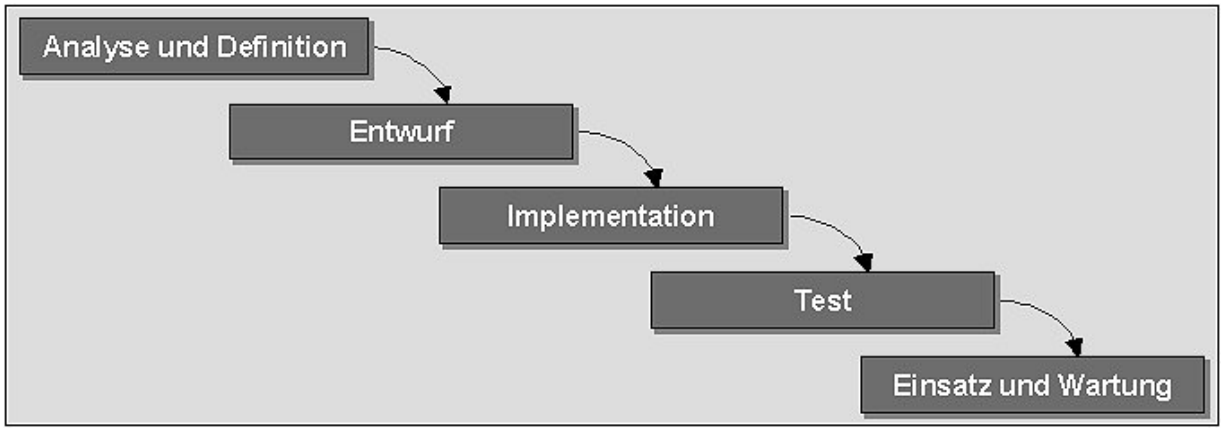

Abbildung 2-3 – Ein Wasserfallmodell (nach Floyd, Züllighoven 1999)

Aktivitäten

Genau wie Phasen dienen auch die in der vorliegenden Arbeit präferierten *Aktivitäten*[28] dazu, das Vorgehen zu strukturieren. Im Gegensatz zu Phasen sind Aktivitäten aber in erster Linie inhaltlich definiert und nicht immer zeitlich begrenzt. Eine Aktivität beschreibt also einen gewissen Bereich, der typischerweise in Projekten zu bearbeiten ist. Diese Aktivitäten können wie Phasen sequentiell geordnet sein, sie können aber auch parallel ablaufen. Eine Aktivität kann aber auch über die gesamte Entwicklungsdauer wichtig sein und somit keinen von vorne herein festgelegten Endzeitpunkt haben (vgl. Kapitel 3 und 5).

Lebenszyklusmodell

In vielen Vorgehensweisen werden Phasen bzw. Aktivitäten in einem *Lebenszyklusmodell* zueinander in Beziehung gesetzt. Das Lebenszyklusmodell bietet einen groben Überblick über den standardmäßigen Ablauf von Projekten und dient gleichzeitig zur Orientierung für die Mitarbeiter. In einer Darstellung eines Lebenszyklusmodells haben die Verbindungen zwischen den dargestellten Phasen bzw. Aktivitäten sowie die räumliche Anordnung allerdings durchaus unterschiedliche Bedeutungen, die in der Vorgehensweise erklärt sein sollten[29].

---

[28] Der Begriff »Aktivität« wurde hier in Anlehnung an »activities« aus Graham et al. (1997, S. 45f) gewählt.

[29] Vgl. z.B. das »contract-driven lifecycle model« in Kapitel 3 sowie das skizzierte Lebenszyklusmodell in Kapitel 5.

Aufgaben, Techniken, Notationen und Vorlagen

Da die in den Aktivitäten beschriebenen Inhalte meist noch immer unhandlich groß sind, werden – je nach Vorgehensweise – *Aufgaben*[30] zusammengestellt oder im Projektverlauf situativ erschlossen, die dann direkt von den Teammitgliedern bearbeitet werden können. Zur Durchführung dieser Aufgaben können wiederum sogenannte *Techniken* verwendet werden, die normalerweise die »best practice« beschreiben. Gemeint sind damit also Techniken, die sich in verschiedenen (eigenen und fremden) Projekten bewährt haben und daher auch zur Verwendung in zukünftigen Projekten empfohlen werden; für solche Techniken werden häufig auch Standard-Schulungen von Fremdanbietern angeboten. Schließlich gibt es verschiedene *Notationen und Vorlagen*, die dazu dienen, die Ergebnisse zu dokumentieren.

Auch wenn sich in nahezu allen Vorgehensweisen die meisten der hier beschriebenen Elemente wiederfinden, so gibt es doch deutliche Unterschiede in der Beschreibung selbst und vor allem in der erwarteten Verwendungsweise dieser Beschreibungen. Besonders interessant für die vorliegende Fragestellung ist der Grad der Flexibilität bzw. starren Vorgabe der einzelnen Elemente und ihrer Verbindungen. Diese Unterschiede werde ich im folgenden Abschnitt betrachten und dabei speziell auf die Merkmale sowie Vor- und Nachteile von schematischen und flexiblen Vorgehensweisen eingehen.

2.2. Schematische und flexible Vorgehensweisen

Wie im vorigen Abschnitt beschrieben, wird ein Vorgehen in der Softwareentwicklung – ob nun in der Beschreibung einer Vorgehensweise explizit gemacht oder nicht – immer durch die in Abbildung 2-1 gezeigten Einflußfaktoren wie Personen, Erfahrungen, Projektspezifika etc. beeinflußt. Daher erscheint es nützlich, sich diese Aspekte bewußt zu machen und strukturiert zu adressieren, um den komplexen Vorgang der Softwareentwicklung besser handhaben zu können. Dies mündet dann in der Forderung nach (definierten) Vorgehensweisen die beispielsweise McDermid wie folgt beschreibt:

> „The difficulties of large-scale software development are now widely appreciated, if not well understood. One of the key factors in countering

[30] Der Begriff »Aufgabe« wurde hier in Anlehnung an »tasks« aus Graham et al. (1997, S. 75f) gewählt.

> these difficulties is to establish an appropriate software development process.“ (McDermid 1991, Part II S. 15/35)

Zunächst soll diese Vorgehensweise das Erreichen der Hauptziele von Softwareentwicklung (rechtzeitige Auslieferung ohne Budgetüberschreitung und mit der erwarteten Qualität etc.) unterstützen. Hierfür lassen sich die von definierten und handlungsleitenden Vorgehensweisen erhofften Vorteile zu vier zentralen Punkten zusammenfassen, die ich im Folgenden kritisch betrachten werde:

- Übersicht und Struktur des Projektverlaufs;
- Unterstützung bei der Arbeit;
- Zeitersparnis auf längere Sicht;
- Befähigung zur Analyse und Verbesserung des Vorgehens.

Übersicht und Struktur des Projektverlaufs

McDermid (1991) beschreibt besonders drastisch, inwiefern eine *Übersicht und Struktur des Projektverlaufs* in Form einer klar beschriebenen Vorgehensweise und eines in Phasen bzw. Aktivitäten gegliederten Lebenszyklusmodells aus seiner Sicht für das Management unverzichtbar ist:

> „Most obviously, the ability to structure the process into phases and to structure the work to be undertaken, i.e. through a work breakdown structure, gives the basis necessary for good project control. Without clearly delineated phases, activities and deliverables it is not possible for managers to assess progress and to exercise the necessary control over the process. Without such structuring one only has chaos.” (McDermid 1991, Part II S. 5)

Ich werde noch ausführen, daß die generellen Konzepte von Phasen und einem zu Projektbeginn vordefinierten Arbeitsplan oder »work breakdown structure« heutigen Projekten nicht angemessen sind. McDermid betont hier aber trotzdem einen wichtigen Punkt, indem er sagt, daß ohne skizzierte – also nicht von vornherein festgeschriebene – Aktivitäten und grobe Struktur die notwendige Kontrolle und Übersicht über das Vorgehen fehlt.

Eine solche Übersicht ist nicht nur aus der Sicht des Managements, sondern häufig auch für die Entwickler selbst sehr hilfreich. In einer Vorgehensweise (und ggf. einem darin integrierten Lebenszyklusmodell) kann jeder leicht erkennen, in welchem Abschnitt der

Entwicklung sich das Projekt zur Zeit befindet, wie die nächsten Teilziele wahrscheinlich aussehen werden und was im Groben als nächstes zu tun ist. Eine ausgearbeitete Struktur kann auch zu Marketingzwecken verwendet werden, um zu dokumentieren, daß das Vorgehen einer gut durchdachten Vorgehensweise folgt und nicht dem von McDermid beschriebenen Chaos.

Unterstützung bei der Arbeit

Neben der Strukturierung des Vorgehens haben die meisten Vorgehensweisen den Anspruch, *Unterstützung bei der Arbeit* zu leisen. Bei einigen Vorgehensweisen scheint sich dieser Punkt aber eher ins Gegenteil zu verkehren; dort werden den Entwicklern so viele Vorschriften auferlegt und von ihnen zusätzliche Dokumente und Tätigkeiten erwartet, die ihrer intuitiven Arbeitsweise entgegen laufen, so daß die Anwendung einer Vorgehensweise oft als Erschwernis anstatt als Erleichterung oder Unterstützung gesehen wird. Sind die Mitarbeiter aber erst einmal in die Vorgehensweise eingearbeitet und sehen selbst die Vorteile, kann sie in vielerlei Hinsicht eine große Unterstützung sein: Tips zu spezifischen Aufgaben können bei der Bewältigung helfen, Checklisten können auf Gefahren und Fehlerquellen hinweisen, Vorlagen können immer wiederkehrende Arbeiten erleichtern, das Nachvollziehen bestimmter Arbeitsschritte vereinfacht sich durch die Einigung auf eine gemeinsame Vorgehensweise drastisch und es wird ein Grundkonsens über das gemeinsame Vorgehen geschaffen, was die Kommunikation erleichtert. Hier sind je nach Ähnlichkeit der Projekte leicht auch noch viel detailliertere Unterstützungen denkbar. So können Vorlagen für Protokolle und Richtlinien zur Erstellung von Anwendungsfalldiagrammen die konkrete Arbeit auch im Detail erleichtern.

Zeitersparnis auf längere Sicht

Ein weiterer erhoffter Vorteil von Vorgehensweisen ist eine *Zeitersparnis auf längere Sicht*. Schon bei der Projektinitialisierung kann eine Vorgehensweise zur zügigen Abwicklung beitragen, da z.B. das spezifische Vorgehen für das Projekt schneller aus der Vorgehensweise entwickelt werden kann. Außerdem kann die Einarbeitungszeit neuer Mitarbeiter verkürzt werden, wenn es eine Vorgehensweise gibt, an der das Vorgehen erlernt wird. Im Idealfall läßt sich eine interne Standard-Vorgehensweise (die nur noch in Details angepaßt werden muß) definieren, die das Zurechtfinden der Mitarbeiter in neuen Projekten erheblich vereinfacht. Ohne beschriebene Vorgehensweise liegen darüber hinaus alle Erfahrungen lediglich bei einzelnen Personen. Verlassen diese das Unternehmen, so gehen die Erfahrungen verloren. Auch wenn sich sicherlich nicht alle Erfahrungen prob-

lemlos niederschreiben lassen, so lassen sich doch gewisse Erkenntnisse leicht durch Hinweise, Vorlagen und Beschreibungen sinnvoller Arbeitsweisen festhalten. Trotzdem sollte nicht unerwähnt bleiben, daß das Beschreiben einer Vorgehensweise selbst und auch das Anpassen an den jeweiligen Projektkontext eine zusätzliche Arbeit bedeutet. Zeitsparend ist eine Vorgehensweise also nur, wenn dieses Beschreiben und Anpassen weniger Zeit in Anspruch nimmt als durch den Einsatz der Vorgehensweise eingespart wird.

Befähigung zur Analyse und Verbesserung des Vorgehens

Schließlich ergibt sich durch eine Vorgehensweise erst die *Befähigung zur Analyse und Verbesserung des Vorgehens.* Auch intuitives, nicht beschriebenes Vorgehen kann über die Zeit verbessert werden. Allerdings ist eine strukturierte Auseinandersetzung mit einem unbeschriebenen Vorgehen deutlich schwieriger:

> „Notice that the availability of a precise model is paramount in raising the effectiveness of the latter, since it provides a non-ambiguous basis for communication about the process." (Derniame 1991, S. 5)

Auch die Weitergabe von bewährtem Vorgehen in bestimmten Situationen ist mündlich zwar möglich, in mittleren oder größeren Organisationen aber kaum konsequent durchführbar. Eine Verbesserung und Erweiterung einer textuellen Beschreibung als Kommunikationsbasis läßt sich dagegen leicht durchführen. Kritisch bleibt allerdings, daß sich Anpassungen der Vorgehensweise nur auswirken, wenn diese auch tatsächlich das Vorgehen in der Praxis bestimmt; dazu wiederum ist ein Grad an Flexibilität notwendig, der die Anpassung an die tatsächliche Situation erlaubt.

Ob diese positiven Effekte von Vorgehensweisen tatsächlich spürbar sind, hängt daher nicht zuletzt von der Frage ab, inwieweit die Beschreibung der Vorgehensweise als schematische, präskriptive Vorschrift oder als flexible, aktiv zu gestaltende Empfehlung aufgefaßt wird. Abbildung 2-4 zeigt dieses Spannungsverhältnis zwischen größtmöglicher Flexibilität und der von definitorischen Vorgehensweisen erhofften detaillierten Handlungsanleitung. Keines der beiden Extreme führt allerdings zu einem sinnvollen Vorgehen: Im schlimmsten Fall mündet eine völlig freie und damit maximal flexible Vorgehensweise im Chaos, während sich in einer streng einzuhaltenden, schematischen Vorgehensweise die Kreativität der beteiligten Personen nicht entfesseln kann und so die Entwicklung angemessener Software behindert wird.

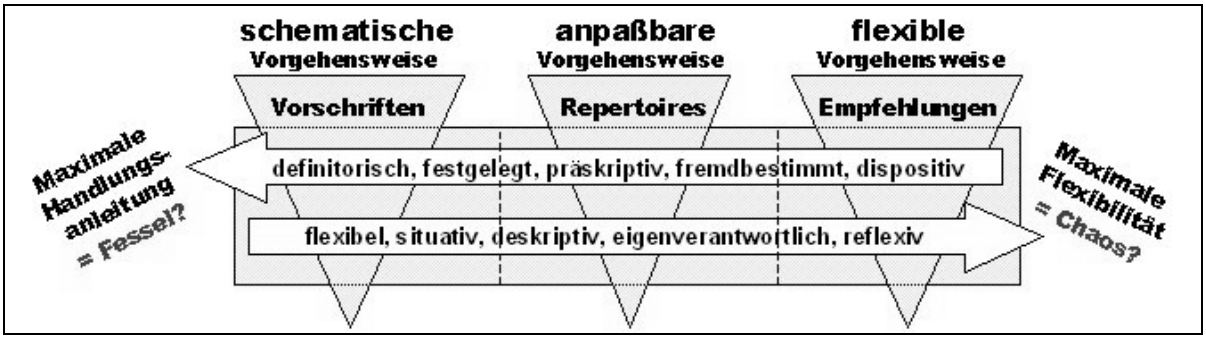

Abbildung 2-4 – Vorgehensweisen im Spannungsverhältnis

Wie ich schon in der Einleitung deutlich gemacht habe, kann eine Vorgehensweise in Form einer starren Vorschrift (z.B. schematische Vorgehensmodelle wie das Phasenmodell) nur in Ausnahmefällen positive Auswirkungen auf die Softwareentwicklung haben; als Hauptprobleme schematischer Vorgehensweisen werden immer wieder genannt:

(A) *Keine zwei Projekte sind gleich* – Daher ist es unmöglich eine starre Vorgehensweise zu definieren, die auf alle Projekte paßt.

(B) *Die Einarbeitung in umfangreiche Vorgehensweisen ist aufwendig* – Das Durcharbeiten von hunderten Seiten Beschreibung bevor das Projekt überhaupt beginnen kann, ist gerade in heutigen Projekten, die auf eine schnelle Durchführung angewiesen sind, unpassend.

(C) *Veränderungen im Projektverlauf können neue Anforderungen an die Vorgehensweise stellen* – Eine starre Vorgehensweise kann auf diese Veränderungen nicht eingehen. Das Vorgehen wird als konstant und einheitlich angenommen.

(D) *Vorgehensweisen spiegeln immer auch persönliche Präferenzen wider* – Starre Vorgehensweisen werden üblicherweise aus der Literatur übernommen und können nicht ausreichend an die eigene Situation und das eigene Team angepaßt werden.

(E) *Es bleiben immer wichtige Aspekte unberücksichtigt* – Eben weil sich die Projekte so stark voneinander unterscheiden, sind auch die Anforderungen an die Vorgehensweisen so verschieden, daß für das eigene Projekt zwangsläufig wesentliche Details fehlen.

(F) *Eine zu starre Vorgehensweise behindert die Kreativität* – Wird durch die Vorgehensweise zu viel vorgeschrieben werden damit auch neue Wege verboten. Damit können dann auch durch diese neuen Wege keine neuen Ideen entstehen.

Diese Probleme zeigen, daß zu starre und schematische Vorgehensweisen die Softwareentwicklung eher behindern als zu einem erfolgreichen Vorgehen zu führen. Daß eine Vorgehensweise durchaus auch positive Effekte haben kann, habe ich aber bereits weiter oben beschrieben. Daher wird eine flexible Vorgehensweise benötigt, deren Flexibilität sich vor allem in drei Punkten zeigen muß:

- Anpaßbarkeit (oder »Instanziierung«, vgl. Kapitel 3) an Personen und Kontext zu Projektbeginn (Probleme A, D, E und F)
- Reaktion auf Veränderungen im laufenden Projekt (Probleme C, E und F)
- Anwendbarkeit in der aktuellen Projektpraxis (Probleme B und E)

Im Folgenden werde ich diese drei Punkte näher erläutern:

Anpaßbarkeit an Personen und Kontext

Daß der *Kontext* des Projektes (und hier speziell der Entwicklungs- und Einsatzkontext) sowie die am Projekt beteiligten *Personen* in einer Vorgehensweise berücksichtigt werden müssen, habe ich bereits in Abschnitt 2.1 erläutert. Innerhalb einer über viele Projekte und Organisationen verwendbaren Beschreibung einer Vorgehensweise muß es möglich sein, sie an die spezifischen Erfordernisse der beteiligten Personen und des Kontextes anzupassen. Dieser Vorgang wird auch als »Instanziierung« bezeichnet (vgl. Abbildung 2-5 und Kapitel 3):

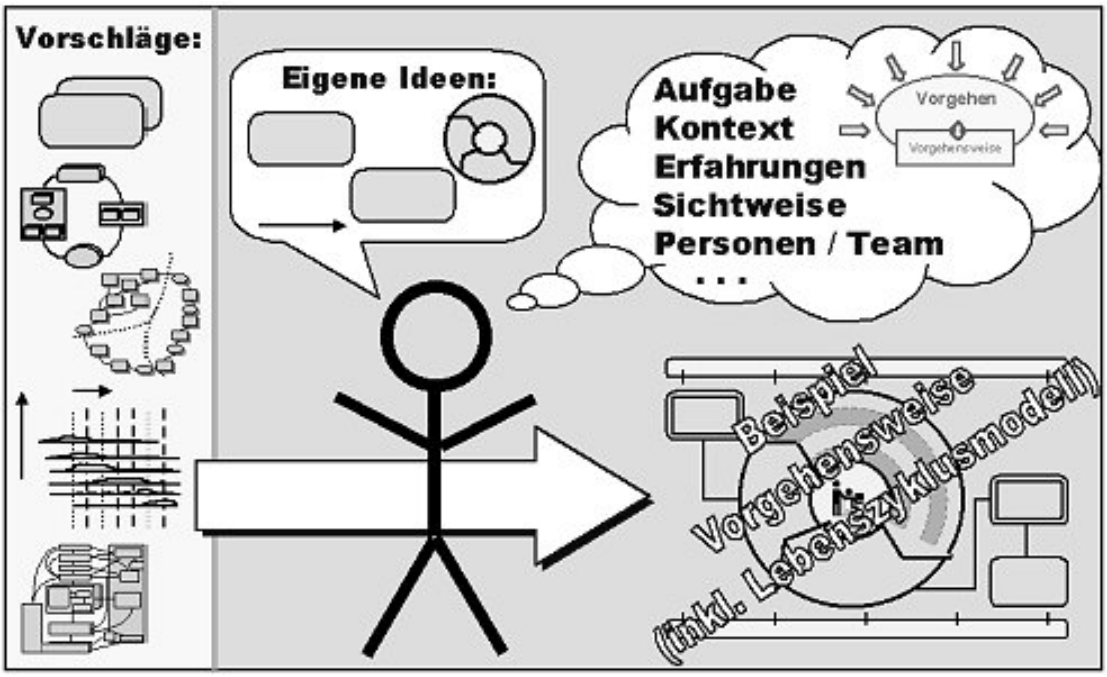

Abbildung 2-5 – Instanziierung einer eigenen Vorgehensweise

Die in der Einleitung bereits erwähnten Umgebungen zur Definition eigener Vorgehensweisen liefern nützliche Konzepte für eine solche Instanziierung[31]. So können aus den Vorschlägen der vorgestellten Aktivitäten, Aufgaben, Techniken oder auch Lebenszyklusmodellen die passenden herausgesucht, an die eigenen Bedürfnisse angepaßt und zu einer neuen Vorgehensweise zusammengestellt werden. Dabei sollten Ideen möglichst vieler beteiligter Personen in die Gestaltung mit einfließen, denn *sie* sollen ja später mit der Vorgehensweise arbeiten. Zusätzlich sollten auch die in Abbildung 2-1 aufgezeigten Einflußfaktoren eines Vorgehens berücksichtigt werden.

Reaktion auf Veränderungen

Neben der Instanziierung, die typischerweise zu Projektbeginn stattfindet, werden aber meist auch Anpassungen während der Projektdurchführung als *Reaktion auf Veränderungen* notwendig. Dave Thomas schreibt im Vorwort zu Jacobson (1992):

> „One of the more profound insights offered by modern software engineering is that change is inevitable, and that software must be designed to be flexible and adaptable in the face of changeing requirements." (Jacobson 1992, S. vi)

[31] Vgl. auch Finkelstein (1994), Henderson-Sellers / Bulthuis (1998) oder die Vorstellung von OPEN in Kapitel 3.

Diese unausweichliche Konfrontation mit Veränderung kann häufig so große Auswirkungen haben, daß nicht nur die Software selbst flexibel und adaptierbar gebaut werden muß, sondern auch die Vorgehensweise, die zur Entwicklung dieser Software dient. Gerade wenn Anforderungen an die Software ständiger Veränderung ausgesetzt sind, muß die Vorgehensweise mit Mechanismen aufwarten, Veränderungen in den Anforderungen aufnehmen und verarbeiten zu können:

> [...] „any real-world software process is a creative process involving numerous human actors; it is dynamic and evolves continuously, and cannot be reduced to the programming of automata. [...] The process model must coordinate activities and manage the information flow, and it must adapt to any evolution of the real world process. This, as will be seen, is a major issue." (Derniame 1999, S. 11)

Neben sich ändernden oder nach und nach verfeinerten Anforderungen, die dann zu neuen Konsequenzen führen, sind folgende Gründe für eine Änderung der Vorgehensweise denkbar, die bei Verwendung einer ungeeigneten Vorgehensweise zu extrem hohen Kosten oder Scheitern des Projektes führen können[32]:

- Veränderung in Personen oder Kultur des Entwicklungskontextes;
- Veränderung in Personen oder Kultur des Anwendungskontextes;
- Neue Benutzer der Software;
- Weiterentwicklung von in der Entwicklung verwendeten Technologien;
- Neue Ideen und Vorschläge von Benutzern oder Entwicklern;
- Veränderte Prioritäten und Termine, die sich beispielsweise aus Konkurrenzsituationen zu Mitbewerbern ergeben können;
- Änderung fachlicher und technischer Ausgangsannahmen;
- evtl. Entschluß, die Entwicklung abzubrechen oder komplett neu zu strukturieren.

Hier entsteht also ein Spannungsverhältnis zwischen dem Ziel, das Vorgehen durch eine klare Vorgehensweise zu unterstützen und der Forderung, diese Vorgehensweise so flexibel wie möglich zu halten. Graham beschreibt eine mögliche Lösung anhand eines anschaulichen Beispiels:

---

[32] Vgl. auch Beck (1999, S. 21ff), Verstegen (2000, S. 88f) und Derniame (1999, S. 11f).

> „Consequently, as many have pointed out, an OO process is not only simply a recipe book by which a series of steps is followed slavishly to produce the perfect ´meal´. Access to business logic is mandatory as is creativity and skill in design. Rather than a ´cookbook,´ a process or method may be better regarded as a good guidebook or road-map. [...] Such a book provides the traveler with the basic layout of the streets, complete with hints, procedures and rules that apply, thus providing for a successful navigation. However, no one expects it to predict occasional disturbances such as burst water mains or pavements closed for renovation and the expert user will need to consult the map less than the novice, often bending the rules to create greater flexibility and permit more originality and creativity." (Graham et al. 1997, S. 17)

> „A well defined process or method thus provides a standard, yet flexible, pattern-based framework for developing systems that blend engineering rigor with engineering creativity thus permitting success to be repeated and repetition of failure to be avoided. " (ebenda)

Anwendbarkeit in der Projektpraxis

In diesen Textstellen wird auch der dritte wesentliche Punkt deutlich: Eine Vorgehensweise muß *in der aktuellen Projektpraxis anwendbar* sein. Auch dies wirkt offensichtlich, steht aber in der theoretischen Erarbeitung einiger Vorgehensweisen zurück; denn einige Beschreibungen sind so umfangreich und komplex, daß ihre praktische Relevanz zumindest für eine größere Zahl von Projekten anzuzweifeln ist.

Ich habe bereits darauf hingewiesen, daß bei der Auswahl der Vorgehensweise natürlich auch die unterschiedlichen Aufgaben- bzw. Problemstellungen sowie die Spezifika des Projektes selbst betrachtet werden müssen. In diesem Rahmen sind sogar durchaus Projekte denkbar, bei denen eine fest vorgegebene, starre oder extrem komplexe Vorgehensweise, wie ich sie weiter oben kritisiert habe, ihren Nutzen haben kann. Für die Entwicklung dedizierter und objektorientierter Anwendungssoftware, auf die sich die vorliegende Arbeit bezieht (vgl. Abschnitt 2.1), scheinen starre, fest vorgegebene oder extrem komplexe Vorgehensweisen in der Praxis allerdings nicht sinnvoll.

2.3. Kriterien für eine flexible Vorgehensweise

In diesem Abschnitt werde ich Kriterien für flexible Vorgehensweisen zur Softwareentwicklung aufstellen, welche die bisher beschriebenen Konzepte zusammenfassen. Mit Hilfe dieser Kriterien sollen dann im weiteren Verlauf der Arbeit in Kapitel 3 OPEN, die Software-Expedition in Kapitel 4 und schließlich in Kapitel 5 Ansätze für eine eigene, OPEN-konforme Vorgehensweise untersucht werden. Neben der Flexibilität, die ich in den Fragestellungen (1a) zur Flexibilität im laufenden Projekt und (1b) zur Anpaßbarkeit vor Projektbeginn adressiere, werde ich auch die praktische Unterstützung im Punkt (2) berücksichtigen.

(1a) Wie flexibel kann innerhalb der Vorgehensweise mit Veränderungen umgegangen werden?

Hierbei gilt es zu untersuchen, inwieweit eine Restrukturierung des Vorgehens auch *im Projektverlauf* möglich ist, und welche Mechanismen die Vorgehensweise bietet, um diese zu unterstützen. Interessant ist auch die Art der Dokumentation von Vorgehen und Vorgehensweise und die Rolle des Teams (»kreative Individuen« oder »ausführende Ressourcen«). Zusätzlich sollte die Bedeutung und der Umfang von Arbeitsdokumenten sowie die Strategie bei der Sicherung ihrer Konsistenz beachtet werden.

Eine zweite Fragestellung, die eng mit der ersten zusammenhängt, betrifft die im vorigen Abschnitt erläuterte Möglichkeit, die Vorgehensweise zu Projekt*beginn* an die spezifischen Bedingungen (beteiligte Personen, Erfahrungen, Einsatz- und Entwicklungskontext, Art und Umfang des Projektes) anzupassen:

(1b) Inwieweit ermöglicht die Vorgehensweise Anpassungen an die spezifischen Bedingungen und die beteiligten Personen zu Projektbeginn?

Entscheidend ist hier die Unterstützung, die von der Vorgehensweise selbst zu dieser Anpassung geliefert wird sowie der Aufwand, der für eine solche Anpassung *zu Projektbeginn* betrieben werden muß. Für kleinere Projekte wäre weiterhin interessant, ob sich auch eine weniger aufwendige Instanz finden läßt, welche ein für Kleinprojekte sinnvolles Vorgehen beschreibt.

Die letzte Fragestellung bezieht sich nicht direkt auf die Flexibilität, sondern zielt vielmehr auf die Verwendbarkeit der Vorgehensweise in der Praxis:

(2) Welche praktische Unterstützung liefert die Vorgehensweise für die täglichen Aufgaben im Projekt?

Mit dieser Frage sollen Vorgehensweisen entlarvt werden, die zwar gute theoretische Ansätze bieten, sich aber in der Projektpraxis nicht oder nur schwerlich umsetzen lassen. Ebenso soll untersucht werden, wie viel praktische Unterstützung, die in der täglichen Arbeit hilfreich ist, die Vorgehensweise den beteiligten Personen an die Hand gibt oder ob die Erstellung von zahlreichen vorgeschriebenen Dokumenten die Arbeit eher behindert. Denn die Vorgehensweise soll im Endeffekt ja dazu dienen, das Vorgehen in der Praxis zu leiten:

> „The ultimate goal of software process technology is to reach the point where the process representation can be used to drive the actual process of software development." (Derniame 1999, S. 4)

Als erste Vorgehensweise werde ich im folgenden Kapitel OPEN vorstellen und anhand der hier beschriebenen Fragestellungen untersuchen, inwieweit OPEN flexible Softwareentwicklung ermöglicht oder sich zumindest als Rahmen zur Erstellung einer flexiblen Vorgehensweise eignet.

3. Object-oriented Process Environment and Notation – OPEN

In diesem Kapitel soll nun der *Object-oriented Process Environment and Notation* (OPEN) als eine Umgebung zur Definition von Vorgehensweisen vorgestellt werden. Nach der Vorstellung werde ich am Ende des Kapitels beurteilen, in welchen Aspekten OPEN die in der vorliegenden Arbeit geforderte Flexibilität zu unterstützen vermag. Zur generellen Einordnung von OPEN unterscheidet Graham drei Generationen von objektorientierten Vorgehensweisen[33]:

Zur ersten Generation Anfang der 1990er Jahre werden alle Vorgehensweisen (Graham schreibt von »methods«, vgl. Abschnitt 2.1 zur Begriffsklärung) gezählt, die von Einzelpersonen (zu den bekannteren zählen hier u.a. Booch, Rumbaugh, Coad und DeMarco) für ihre eigenen Projekte entwickelt und später veröffentlicht wurden. Probleme bestanden hier vor allem darin, daß die Autoren nicht über genügend Ressourcen verfügten, um eine vollständige Vorgehensweise zu beschreiben. Meist lag der Fokus auf der Darstellung und Notation der Problembereiche, also eher im Bereich von Techniken aus OPEN (vgl. Abschnitt 4.2.3), so daß lediglich ein Ausschnitt einer Vorgehensweise realisiert wurde.

Mitte der 1990er Jahre kamen dann die objektorientierten Vorgehensweisen der zweiten Generation auf. Obwohl der Autorenkreis der einzelnen Vorgehensweisen noch immer auf kleine Gruppen oder Einzelpersonen beschränkt war, ist deutlich zu erkennen, daß Einflüsse der anderen Autoren auf die neu entstehenden einwirkten. Ein gutes Beispiel bieten hier die in *Objectory* eingeführten und inzwischen sehr populären Anwendungsfälle[34], die als Technik zur Geschäftsprozeßanalyse in vielen Vorgehensweisen wiederzufinden sind[35].

In den letzten Jahren sind dann die Vorgehensweisen der dritten Generation aufgekommen, zu denen auch OPEN gehört. Diese Vorgehensweisen werden einerseits von einer größeren Gemeinde von Autoren entwickelt (im Fall von OPEN handelt es sich um das OPEN Consortium mit über 30 Mitgliedern[36]) und haben darüber hinaus das Ziel, für das gesamte Vorgehen der Softwareentwicklung Unterstützung zu liefern und nicht nur für einzelne

---

[33] Vgl. Graham et al. (1997, S. 4).

[34] Engl.: »Use Cases«, vgl. Jacobson et al. (1992).

[35] Zentral sind sie z.B. auch im „use-case-driven approach“ (vgl. Abschnitt 4.2).

[36] Vgl. Henderson-Sellers et al. (1998).

Teile. OPEN versucht, dieses Ziel auch dadurch zu erreichen, daß aus vielen Vorgehensweisen[37] gute und bewährte Ideen übernommen wurden.

3.1. Überblick

Zum generellen Verständnis, über den Aufbau von OPEN und zur Identifikation der Stärken und Schwächen werde ich hier zunächst einen Überblick über die generellen Grundsätze und Gedanken von OPEN geben. Wesentliche Grundgedanken lassen sich bereits aus der Bedeutung des Akronyms »O-P-E-N« selbst herauslesen:

O – für »object-oriented«

Wie viele Autoren festgestellt haben, liefert die Objektorientierung eine gute Unterstützung, Konzepte der realen Welt nahtlos (englisch: »seamless«) in ein Programm zu übersetzen[38]. Für die hier betrachtete Entwicklung dedizierter und großer Anwendungssoftware ist vor allem die Durchgängigkeit der Objektorientierung von großer Bedeutung, da dies eine weitgehende Vereinheitlichung der sprachlichen Konzepte in allen Ebenen der Entwicklung (von Gesprächen mit Anwendern über die fachlichen Objekte bis hin zur Implementierung programmiersprachlicher Objekte) stark vereinfacht. Meyer (1995) fordert dem entsprechend die konsequente Nutzung der objektorientierten Ideen in seinem »Seamlessness Principle«:

> "Object-oriented ideas are meant to be applied to all steps of software development, including analysis, design, implementation and maintenance, and to decrease the gaps between these steps." (Meyer 1995, S. 24)

Auf der anderen Seite kann schlüssig argumentiert werden, daß sich gewisse Sprünge in der Umsetzung nicht vermeiden lassen[39]. Das liegt u.a. daran, daß auf der hohen Abstraktionsebene der Anforderungserfassung eben keine Rücksicht auf technische Details von Programmiersprachen oder Hardware genommen wird und auch nicht genommen werden soll. Heute und in absehbarer Zukunft wird es aber weiterhin so sein, daß auf niedrigerer

---

[37] Vgl. Henderson-Sellers et al. (1998, S. 5).

[38] Vgl. allgemeine Literatur zur Objektorientierung wie z.B. Gamma et al. (1995) oder Jacobson et al. (1992).

[39] Vgl. Graham et al. (1997, S. 35ff).

Abstraktionsebene – also beispielsweise bei der Implementierung – unvorhergesehene Entscheidungen getroffen werden, die dann zu einem Ebenenbruch führen.

Trotz dieser Einschränkungen bietet die heutige Objektorientierung eine Durchgängigkeit, die sich in Projekten durchaus dazu nutzen läßt, von einer Repräsentation des abzubildenden Geschäftsprozesses einen weichen und in weiten Teilen reversiblen Übergang zur Implementierung zu schaffen[40]. OPEN unterstützt dies mit eigenen Vorschlägen sowie mit Verweisen auf detailliertere Literatur in den Beschreibungen der Aufgaben und Techniken (vgl. Kapitel 3.2.2 und 3.2.3).

P – für Process

Zur Strukturierung der Vorgehensweise und als Orientierungshilfe im tatsächlichen Vorgehen selbst, bietet OPEN als größte Einheit ein Lebenszyklusmodell bestehend aus Aktivitäten, auf das ich in Abschnitt 3.2.1 noch eingehen werde. Sowohl die Aktivitäten als auch die darunterliegenden Konzepte der Aufgaben und Techniken (s.u.) sind katalogartig beschrieben, so daß für jede spezifische Organisations- oder Projektsituation ein passendes Lebenszyklusmodell zusammengestellt und angepaßt werden kann.

E – für Environment

Diese Anpassung[41] geschieht innerhalb einer Umgebung, die in OPEN ebenfalls adressiert wird. Hierzu gehören viele Elemente, die in traditionellen Vorgehensweisen nicht berücksichtigt wurden, wie z.B. der Einfluß von Softwarewerkzeugen und Technologien, Auswirkungen der Organisationskulturen (sowohl die der Entwicklungs- als auch die der Anwendungsorganisation) sowie Individuelle Eigenschaften, Vorlieben und Fähigkeiten der beteiligten Personen.

N – für Notation

OPEN beinhaltet eine eigene Modellierungssprache – die *OPEN Modeling Language* (OML). OML wird zwar von den Autoren empfohlen, und sie weisen auf die Gefahren und Unzulänglichkeiten von Alternativen hin, in dieser Arbeit soll OML aber trotzdem nicht behandelt werden; denn ein großer Nachteil von OML besteht in der mangelnden Verbrei-

---

[40] Hintergründe zur Problematik in Bezug auf reversibles, nahtloses OO-Design zusammen mit einem Lösungsansatz – der *Business Object Notation* (BON) – finden sich z.B. in Waldén (1995).

[41] Vgl. dazu auch die Beschreibung der Instanziierung in Abschnitt 2.2.

tung. Mit der *Unified Modeling Language* (UML)[42] steht indes eine alternative Sprache zur Verfügung, die so verbreitet ist, daß von einem »de facto Standard« gesprochen werden kann. Daher räumen auch die Autoren von OPEN ein, daß UML in der Praxis in Projekten mit OPEN die am häufigsten verwendete Sprache sein wird. Firesmith et al. (1998) gibt daher Hinweise und Beispiele, wie die OML-Diagramme sinnvoll in UML-Diagramme übertragen werden können und mit Henderson-Sellers und Unhelkar (2000) steht ein Buch zur Verfügung, das sich dem Einsatz von OPEN mit der UML widmet. Darüber hinaus liegt der Mehrwert von OML auch nicht so sehr darin, daß viele neue Funktionen zur Verfügung gestellt werden, die mit der UML nicht realisierbar sind. Vielmehr scheint den Autoren die UML an einigen Stellen nicht intuitiv, und sie ermöglicht vereinzelt Darstellungen, die objektorientierte Entwicklung eher behindern, als sie zu unterstützen[43].

Weitere Konzepte von OPEN

Wie bereits angesprochen erhebt OPEN den Anspruch, eine *Unterstützung für den gesamten Prozeß* der Softwareentwicklung[44] zu liefern und nicht nur bei Teilen wie der Anforderungsermittlung oder objektorientierten Modellierung durch eine Auswahl an Techniken zu helfen. Dazu stehen verschiedene grundlegende Konzepte bereit, die ich im Folgenden kurz beleuchten werde.

Einen groben Überblick über das Vorgehen der Softwareentwicklung liefert das *Lebenszyklusmodell*, das in OPEN aus verschiedenen sich gegenseitig beeinflussenden *Aktivitäten* besteht und in einer *Instanziierung* an die spezifische Projektsituation angepaßt wird[45]. Zur Ausführung dieser Aktivitäten stehen *Aufgaben* zur Verfügung, die einen Umfang haben, der von einzelnen oder mehreren Teammitgliedern mit der Hilfe der *Techniken* bewältigt werden kann (vgl. Abbildung 3-1). Teil der Instanziierung ist es dann auch, den ausgewählten Aktivitäten für das spezifische Projekt passende Aufgaben und diesen wiederum Techniken zuzuordnen[46].

---

[42] Eine gute Einführung in die UML gibt z.B. das Buch von Fowler (2000).

[43] Vgl. Henderson-Sellers (1998, S. 10).

[44] Vgl. Henderson-Sellers und Unhelkar (2000, S.3).

[45] Vgl. Abschnitt 2.2.

[46] Zur näheren Beschreibung von Lebenszyklusmodell (»lifecycle model«), Aktivitäten (»activities«), Aufgaben (»tasks«) und Techniken (»techniques«) vgl. die Abschnitte 3.2.1 bis 3.2.3.

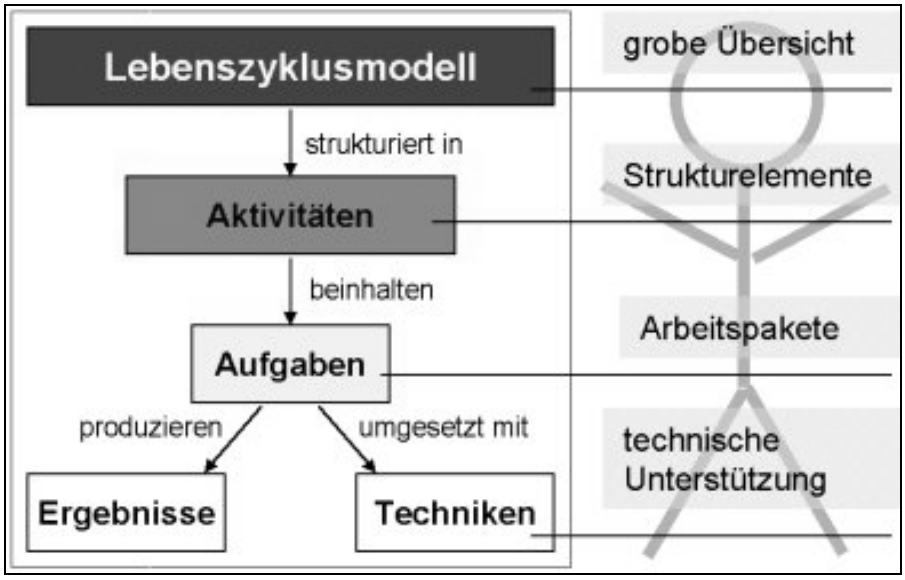

Abbildung 3-1 – Gliederung des Lebenszyklusmodells in OPEN

Entscheidend ist hier, daß Aktivitäten, Aufgaben und Techniken als erweiterbare Ressourcen zur Verfügung gestellt werden und nicht als allumfassende, ultimative Vorschrift anzusehen sind. Auch das Lebenszyklusmodell ist nicht, wie von anderen Vorgehensweisen bekannt, fest vorgegeben, sondern für die eigenen Bedürfnisse selbst zusammenzustellen. Bei den *auslieferbaren Ergebnissen* der Aktivitäten handelt es sich um verschiedene Arten von Dokumenten, Quellcode, Plänen usw.

In Vorgehensweisen zur Objektorientierten Softwareentwicklung finden sich sehr häufig die Konzepte der *iterativen und inkrementellen Systementwicklung*. Auch in OPEN werden verschiedene Aktivitäten nicht in einem Stück abgearbeitet, sondern über den Projektzeitraum verteilt in mehreren Iterationen durchlaufen. Zwischen diesen Iterationen stehen dann meist Tests und umfangreiche Reviews der Anwender oder Kunden, auf deren Grundlage der Umfang und Inhalt der nachfolgenden Iteration abgestimmt wird. Prototypen und Teilergebnisse bilden dabei die Grundlage für diese Reviews und werden in den folgenden Iterationen inkrementell zu einem voll funktionsfähigen System weiterentwickelt.

Testen erhält in OPEN ein starkes Gewicht: Zum einen stehen Beschreibungen von Aufgaben und Techniken zur Verfügung, um die Teammitglieder bei der Konzeption und Ausführung von Tests zu unterstützen und ihnen Richtlinien an die Hand zu geben[47]. Zusätzlich steht in OPEN am Ende jeder Aufgabe ein Test, der die Qualität und Relevanz der erzeugten Ergebnisse gewährleisten soll.

In den letzten Jahren wurde viel über den Nutzen und zusätzlichen Aufwand von *Wiederverwendung und objektorientierten Klassenbibliotheken* geschrieben[48]. Auch OPEN empfiehlt die Verwendung von Klassenbibliotheken zur beschleunigten und qualitativen Entwicklung (vgl. hierzu auch Kapitel 5).

3.2. Aufbau

Zur Zeit liegen in der Literatur einige Beschreibungen von OPEN Instanzen vor. In Kapitel 5 werde ich zusätzlich noch Ansätze zu einer weiteren Instanz sowie meine damit verbundene Praxiserfahrungen vorstellen. Die Studie solcher Beschreibungen und deren Argumentationen ist eine wichtige Informationsquelle für die Adaption von OPEN für eigene Entwicklungsvorhaben. Daher sollen in diesem Abschnitt die wichtigsten OPEN- Lebenszyklusmodell (zum Aufbau der Lebenszyklusmodell vgl. auch Abbildung 3-1) und die ihnen zugrundeliegenden Konzepte kurz angedeutet werden.

Der Rahmen dieser Arbeit läßt es nicht zu, eine OPEN-Instanz vollständig vorzustellen. Auch werde ich auf die Beschreibungen der Aktivitäten, Aufgaben und Techniken aufgrund ihrer Vielzahl[49] nicht eingehen können. Es geht mir in diesem Abschnitt vielmehr darum, anhand einiger bestehender Beispiele, das generelle Konzept von OPEN zur Instanziierung sowie Erstellung und Darstellung von Lebenszyklusmodellen zu zeigen, um im darauffolgenden Abschnitt 3.3, die für die Flexibilität günstigen und kritischen Punkte hervorheben zu können. Diese Kritikpunkte werde ich dann daraufhin untersuchen, inwieweit es sich um Probleme der einzelnen Ausprägung der OPEN-Instanz oder um generelle Defizite des OPEN-Ansatzes handelt.

Als umfangreichste Quellen für die Beschreibungen von Aktivitäten und Aufgaben sei hier zentral auf Graham et al. (1997) sowie für die Techniken auf Henderson-Sellers et al. (1998) verwiesen, die Beschreibungen und vor allem auch Verweise auf hilfreiche weiterführende Literatur geben. Auf andere Quellen werde ich an geeigneter Stelle gesondert hinweisen.

---

[47] Vgl. z.B. Henderson-Sellers und Unhelkar (2000, 223ff).

[48] Vgl. zum Beispiel Meyer (1994), Graham et al. (1997, S. 179ff), Sommerville (1995, Kapitel 20) und Piemont (1999).

[49] Graham et al. (1997) beschreiben beispielsweise 30 Aufgaben mit zusammen fast 50 Unteraufgaben und in Henderson-Sellers et al. (1998) finden sich über 200 Techniken.

3.2.1. Aktivitäten strukturiert in einem Lebenszyklusmodell

Allen in der Literatur auffindbaren Lebenszyklusmodellen zu OPEN liegt ein eigenes, leitendes Konzept zu Grunde. Die ersten zusammen mit der OPEN-Spezifikation in Graham et al. (1997) veröffentlichten Lebenszyklusmodell[50] gehen beispielsweise auf das Vertragsmodell von Meyer (1995)[51] zurück. Weitere leitende Konzepte für andere Lebenszyklusmodelle, auf die ich in diesem Abschnitt kurz eingehen werde, umfassen die Hervorhebung von Verantwortlichkeiten als treibendem Element[52] sowie den verbreiteten Ansatz der Orientierung an Anwendungsfälle[53].

»Contract-driven lifecycle model«

Meyer (1995) zufolge liegt die häufigste Ursache für Fehler in Softwaresystemen in der unklaren Definition von Modul-, Objekt- oder Komponentengrenzen und in den daraus resultierenden falschen Erwartungen beider Seiten[54]. Das *Vertragsmodell* begegnet diesem Problem, indem es Entwickler verpflichtet, Zusicherungen für ihre Objekte bzw. Teilsysteme zu machen. Es erweitert das grundsätzliche objektorientierte Konzept der Kapselung um den Vertrag zwischen Objekten. Der Vertrag wird dabei definiert als:

> ”a detailed statement of what the feature offers to the clients, and what it requires from them in order to work properly” (Meyer 1995, S. 15)

Dieser Vertrag spezifiziert also, was ein Objekt zu liefern hat und was im Gegenzug von anderen zugesichert wird. Es wird festgelegt, welche Leistungen zu erbringen sind (sog. Vorbedingungen oder »pre-conditions«), damit der Informationsaustausch stattfinden kann und von welchen Ergebnissen (sog. Nachbedingungen oder »post-conditions«) man nach Ablauf des Informationsaustauschs ausgehen kann.

---

[50] Vgl. das »contract-driven lifecycle model« und die »firesmith instantiation« (s.u.) sowie Firesmith et al. (1998, S. 35ff).

[51] Im Original: »design by contract«.

[52] Vgl. »responsibility-driven design« und »use-case-driven approach« (s.u.)

[53] Beide Ansätze werden in Henderson-Sellers und Unhelkar (2000) vorgestellt. Dabei basiert das »responsibility-driven design« auf Wirfs-Brock et al. (1990) und der »use-case-driven approach« auf Jacobson et al. (1992).

[54] Vgl. Meyer (1995, S. 16f).

OPEN macht sich das Vertragsmodell für die Interaktion von Aktivitäten unter anderem im Standard-Lebenszyklusmodell von OPEN, dem »contract-driven lifecycle model« (im Folgenden mit CDLM abgekürzt) zu Nutze[55]. Jede Aktivität besitzt darin eigene Vor- und Nachbedingungen, die sich meist auf die schon erwähnten auslieferbaren Ergebnisse beziehen: Bestimmte Aktivitäten (wie z.B. ein Einsatz des Systems in der Zielumgebung) sind erst sinnvoll, wenn ein bestimmtes Ergebnis erstellt wurde (z.B. ein Prototyp oder eine erste Version des Systems in einer Testumgebung). Eine weitere Nachbedingung jeder Aktivität ist ein Test. Dieser wird je nach Aktivität verschieden aussehen. Dabei kann es sich um Testen des Quellcodes im herkömmlichen Sinne handeln, es kann aber auch ein Review bestimmter Dokumente gemeint sein. Jede Aktivität sollte einen Hinweis dazu enthalten, wie der jeweilige Test auszusehen hat, und was die gewünschten Ergebnisse sind, die die Nachbedingung erfüllen.

Abbildung 3-2 zeigt das CDLM, das die Aktivitäten organisiert. Dabei stehen die Pfeile in der Abbildung nicht unbedingt für eine zeitliche Sequenz, in der die Aktivitäten sequentiell abgearbeitet werden sollen; sondern sie zeigen wesentliche Abhängigkeiten, die durch Vor- und Nachbedingungen entstehen. Dabei kann sich die Ausführung verschiedener Aktivitäten zeitlich überlagern, und es gibt zyklische Abhängigkeiten, die auf verschieden Versionen von Ergebnissen zurückgehen. Besonders an den Stellen an denen die Pfeile in beide Richtungen zeigen, ist also im CDLM eine beidseitige Abstimmung und Koordination der Aktivitäten vorgesehen.

[55] Vgl. Graham et al. (1997, S. 26).

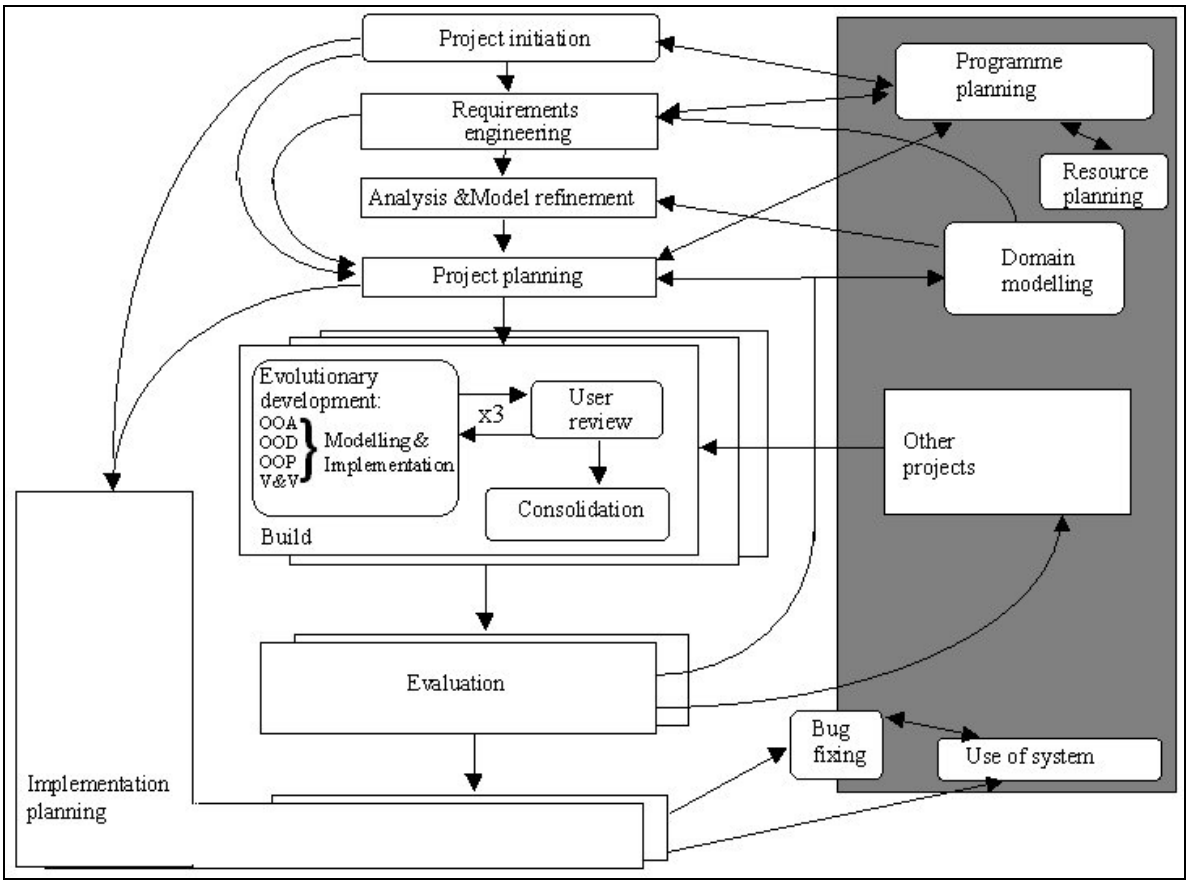

Abbildung 3-2 – OPEN »contract-driven lifecycle model«, CDLM
(aus Graham 1997, S. 46)

Im CDLM wird ein Entwicklungsvorhaben mit der Aktivität »Project initiation« begonnen. Hier werden erste Gespräche geführt, Verträge ausgehandelt und allgemeine Grundsteine für das Projekt gelegt. Die Aktivitäten »Requirements Engineering«, »Analysis & Model Refinement«, »Build« und »Evaluation« beschäftigen sich mit der Aufnahme, Analyse, Umsetzung und Beurteilung von Anwenderanforderungen an das zu erstellende System. Zu sehen ist hier die Aufteilung in mehrere *Iterationen* z.B. in der Aktivität »Build« (dargestellt durch sich überlappende Kästen). »Project Planning«, »Programme Planning«, »Ressource Planning« und »Implementation Planning« beinhalten das von OPEN ebenfalls unterstützte Projektmanagement. Der in Abbildung 3-2 rechts dargestellte dunkel unterlegte Bereich zeigt projektübergreifende Aktivitäten, die direkten Einfluß auf eventuell existierende andere Projekte im gleichen Bereich haben und entsprechend koordiniert werden müssen. Hier finden sich Aktivitäten zur Domänenmodellierung und dem Aufbau von Rahmenwerken (»Domain Modelling«) und zum direkten Testen von Schnittstellen und dem Einsatz des Systems (»Other Projects«, »Bug Fixing«, »Use of System«).

Zur weiteren Strukturierung sind die Aktivitäten auf zwei verschiedene Arten dargestellt: Im Gegensatz zu einer Aktivität in einem Rahmen mit abgerundeten Ecken, ist eine mit einem eckigen Rahmen dargestellte Aktivität durch das sogenannte *»Timeboxing«* zeitlich begrenzt. Es gibt hier also einen Endzeitpunkt, der in jedem Fall gehalten werden sollte und nach Ansicht von Graham et al. (1997, S. 47f.) folgende Vorteile bewirkt:

- *Unterscheidung von Wünschen und Notwendigkeiten* – durch die Verteilung von Prioritäten von Beginn an kann klar entschieden werden, was bis zum Auslieferungstermin in jedem Fall erledigt sein muß[56].
- *Bewußte Aufnahme neuer Funktionalität* – klare und kurzfristige Deadlines machen offensichtlich, daß neue Funktionalitäten oft nur in der nächsten Iteration aufgenommen werden können. So wird vermieden, daß nach und nach immer mehr neue Wünsche hinzukommen, ohne wirklich zusätzliche Zeit für die neue Funktionalität einzuplanen.
- *Motivation für das Team* – bei Erreichen jeder Deadline können sowohl die Entwickler als auch die Anwender ein direktes Resultat ihrer Arbeit sehen.
- *Einbeziehung der Anwender* – bei relativ kurzen Iterationszyklen werden die Anwender durch ihr Feedback im Rahmen eines »User Review« in die Entwicklung mit einbezogen.

»responsibility-driven design«

Da das CDLM recht umfangreich ist und aus einer großen Zahl von Aktivitäten besteht, die nicht in allen Projekten benötigt werden[57], stellen Henderson-Sellers und Unhelkar (2000) ein anderes OPEN-Lebenszyklusmodell vor (vgl. Abbildung 3-3) und zeigen, wie die konkrete Nutzung im Projekt aussehen könnte. Das Lebenszyklusmodell wird sich in der gezeigten Form für die Praxis allerdings kaum eignen; um den Umfang zu reduzieren und den Fokus auf die an dieser Stelle entscheidenden Details[58] zu legen, wird zu Schu-

[56] Das Verteilen von Prioritäten wird z.B. auch im „Planning Game“ des Extreme Programming als wesentlich erachtet (vgl. Beck 1999, Kapitel 15).

[57] Das CDLM ist in der Darstellung und Vielzahl der Aktivitäten darüber hinaus auch relativ unübersichtlich. Auf diese Punkte werde ich aber in Abschnitt 3.3.1 noch näher eingehen.

[58] Es ging dabei um grundlegende Gedanken zum Vorgehen und die Modellierung der Klassen – vgl. Henderson-Sellers und Unhelkar (2000, S. 115).

lungszwecken nämlich von der überaus unrealistischen Situation ausgegangen, daß die Anforderungen bereits unveränderbar *fest*stehen:

> „For this example only, we initiate the project with the assumption that we have been given all the requirements – so we do not have an activity for Requirements Engineering." (Henderson-Sellers und Unhelkar 2000, S. 115)

Das Beispiel beansprucht aber auch nicht, so wie es ist als Vorgehensweise in der Praxis verwendet zu werden, sondern zeigt neben der Anwendung der ausgewählten Aktivitäten, daß auch Anpassungen des CDLM verwendet werden können, um Unterstützung in Form eines Lebenszyklusmodells zu erhalten.

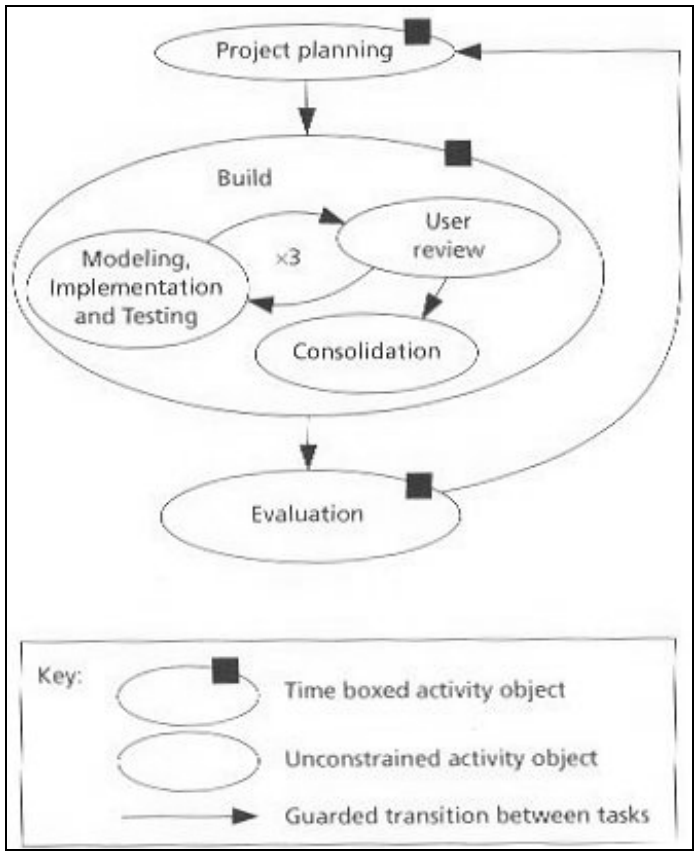

Abbildung 3-3 – Ein einfaches Lebenszyklusmodell für »responsibility-driven design« (aus Henderson-Sellers und Unhelkar 2000, S. 117)

Als treibendes Element werden hier Verantwortlichkeiten zur Modellierung benutzt:

> „A responsibility is any purpose, obligation or required capability of the instances of a class or package. A responsibility is typically implemented by a cohesive collection of one or more features. A responsibility can be

> a responsibility for doing, a responsibility for knowing or a responsibility for enforcing.“ (Henderson-Sellers und Unhelkar 2000, S. 65)

Henderson-Sellers und Unhelkar unterscheiden hier mit Bezug auf Wirfs-Brock et al. (1990) zwischen dem Fokus auf Daten, Funktionen oder Verantwortlichkeiten. Die hier gewählte Analyse und Modellierung der Verantwortlichkeiten bringt vor allem bei der Entwicklung von spezieller Anwendungssoftware große Vorteile, da Verantwortlichkeiten prinzipiell auf ein spezielles Problem bezogen sind. Daten und Funktionen lassen sich eher generell formulieren und können daher, nach Ansicht der Autoren, eher bei der Entwicklung von Massensoftware dienlich sein, die ohne Einbeziehung der Anwender geschieht.

»Firesmith instantiation«

Eine der Vorgehensweisen, aus denen OPEN entstanden ist, ist die »Firesmith Development Method«, die in diesem Lebenszyklus als OPEN-konformen Lebenszyklusmodell weiterbesteht (vgl. Abbildung 3-4). Hier finden sich in einem neuen Lebenszyklusmodell zusätzlich zu vom CDLM bekannten auch einige neue Aktivitäten. Schon an der verringerten Gesamtzahl verschiedener Aktivitäten läßt sich erkennen, daß dieses Lebenszyklusmodell eine weniger komplexe Vorgehensweise für kleinere Projekte darstellen soll, in der beispielsweise auch sämtliche Aspekte wegfallen, die sich auf die Entwicklung von Systemen mit mehreren Teilprojekten oder das Projektmanagement insgesamt beziehen.

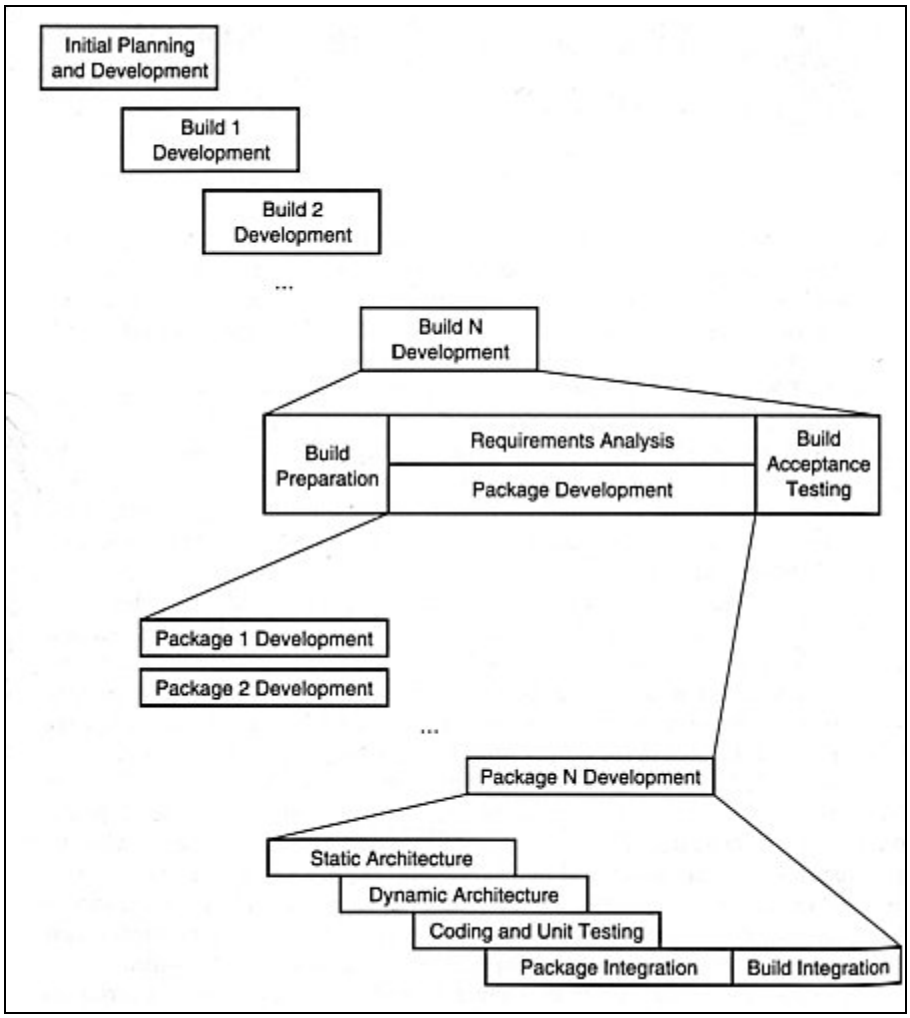

Abbildung 3-4 – OPEN »firesmith instantiation« (aus Firesmith et al. 1998, S. 36)

Die iterative Entwicklung bekommt in dieser Vorgehensweise ein noch stärkeres Gewicht, indem z.B. fast der gesamte Prozeß in der iterativen Aktivität »Build Development« zusammengefaßt ist. Innerhalb der Iterationen finden nach einer vorbereitenden Aktivität »Build Preperation« die »Requirements Analysis« und das wiederum iterativ organisierte »Package Development« parallel statt. Danach wird die Hauptiteration mit der Aktivität »Build Acceptance Test«, die auch das Testen der entwickelten Systemteile beinhaltet, abgeschlossen. Firesmith schreibt hierzu auch:

> „[...] the Firesmith Development Method uses an incremental, iterative and parallel development cycle. It is incremental in terms of builds and packages of classes, whereby packages are typically developed in a bottom-up dependency-based manner. It is iterative in that all of the models evolve through several iterations as knowledge increases and improvements are discovered. It is parallel in that all layers, subdomains

> and packages are typically developed by different teams working in parallel.
>
> The reader should recognize that there should be a great deal of overlap in the above activities and tasks due to the incremental, iterative and parallel nature of the object-oriented development cycle." (Firesmith 1998, S. 35)

Neben der iterativen Entwicklung werden hier also auch die wichtigen Konzepte der inkrementellen Entwicklung und parallelen Bearbeitung von Aktivitäten besonders hervorgehoben.

»use-case-driven approach«

Auch das in Abbildung 3-5 gezeigte Lebenszyklusmodell zeigt Aktivitäten, die aus den zuvor besprochenen Lebenszyklusmodellen bekannt sind. Zusätzlich dazu, sind hier zu jeder Aktivität die Bezeichnungen der wichtigsten zugeordneten Aufgaben genannt, um hervorzuheben, daß die Aufgaben letztendlich beschreiben, was getan werden soll; denn die Aktivitäten bilden lediglich einen strukturellen Rahmen, der zusammengenommen in Form des Lebenszyklusmodells einen Überblick über das Vorgehen zur Verfügung stellt.

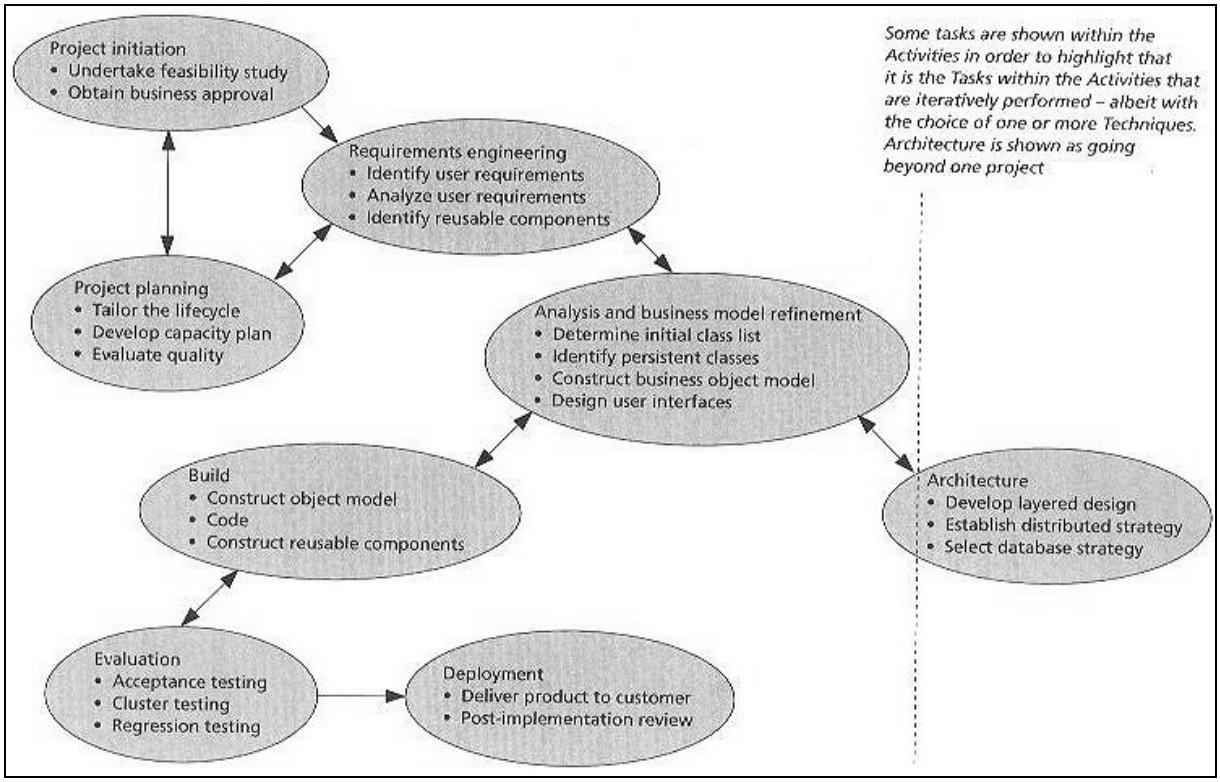

Abbildung 3-5 – Lebenszyklusmodell des »use-case-driven approach«
(aus Henderson-Sellers und Unhelkar 2000, S. 155)

Henderson-Sellers und Unhelkar schlagen mit dieser Instanz von OPEN ein Vorgehen mit UML Anwendungsfällen als treibendem Element vor. Die Autoren weisen aber auch darauf hin, daß die Verwendung von UML Anwendungsfälle seit einiger Zeit durchaus kontroverse Diskussionen hervorgerufen hat[59]. Weitestgehend unumstritten sind die Vorzüge im Bereich der Verbreitung und Einfachheit des UML-Standards: Während auf der einen Seite die Technik der Modellierung von Anwendungsfällen bei Entwicklern im objektorientierten Umfeld vorausgesetzt werden kann, so fällt es auf der anderen Seite sogar softwaretechnisch unerfahrenen Anwendern und Entscheidungsträgern relativ leicht, die in Anwendungsfällen dargestellten Sachverhalte nachzuvollziehen und als gemeinsame Diskussionsgrundlage zu nutzen. Außerdem können auf Basis der Anwendungsfälle relativ einfach Testfälle für Anwendertests formuliert werden.

Auf der anderen Seite werden hier aber auch Probleme bei der Modellierung mit Anwendungsfällen genannt. So liefern diese beispielsweise keine besonders gute Unterstützung, möglichst nahtlos Kandidaten für Objekte bzw. Klassen und ihre Beziehungen untereinander zu identifizieren. Dafür werden häufig Techniken wie textuelle Analyse, CRC-Cards oder in Einzelfällen ein Zwischenschritt über die Erstellung von Sequenzdiagrammen hinzugezogen[60]. Darüber hinaus wird darauf hingewiesen, daß aus UML Anwendungsfällen leicht ein Fluß von Informationen ablesbar ist. Dieser Fluß erschwert in Objektorientierung unerfahrenen Modellierern aber die Identifikation der Objekte auf ein Weiteres und verleitet zur traditionellen, imperativen Programmierung.

Neben den erwähnten bewährten Techniken, schlägt Graham et al. (1997) im Rahmen von OPEN auch ein durchgängiges Prinzip vor, wie die möglichst nahtlose Transformation von Anforderungen in Code unterstützt werden kann: An die Stelle der UML Anwendungsfälle treten dort, wie in SOMA[61] vorgeschlagen, sogenannte TOMs[62]. Diese sind von vorne herein so angelegt, daß die Transformation in weitergehende Modelle leicht fällt. So sollen Beschreibungen zu einem TOM beispielsweise in möglichst primitiven Sätzen (Subjekt – Prädikat – ein Objekt) abgefaßt werden, um die Textanalyse zur Identifikation der Klas-

---

[59] Vgl. Henderson-Sellers und Unhelkar (2000, S. 174).

[60] Vgl. Henderson-Sellers et al. (1998, S. 158f und 330f) sowie Henderson-Sellers und Unhelkar (2000, S. 174f).

[61] »Semantic Object Modeling Approach«, vgl. Graham (1998).

[62] »Task Object Models«, vgl. Graham (1997).

senkandidaten zu erleichtern. Über einige Zwischenschritte gelangt man schließlich zu den Klassen und ihrer Implementierung[63].

3.2.2. Konkrete handlungsleitende Aufgaben in Aktivitäten

In den Beschreibungen der OPEN-Aufgaben werden Hinweise, Tips und Anleitungen gegeben, was zu bestimmten Bereichen typischerweise zu tun ist und worauf geachtet werden sollte. Diese konkreten Handlungsanleitungen stehen bei Bedarf zu verschiedenen Themenbereichen zur Verfügung und werden gerade von und für unerfahrenere Entwicklern immer wieder gefordert (vgl. Kapitel 5).

Zur besseren Übersicht gruppieren Graham et al. (1997) die Aufgaben inhaltlich in sieben größere Bereiche: Benutzerinteraktion und fachliche Aufgaben, Gesamtarchitektur, Projektmanagement, Qualitätsmanagement, Datenbanken, Verteilung, Modellierung / Entwicklung und Wiederverwendbarkeit[64]. Zu jeder dieser Gruppen gibt es mehrere Aufgaben, die auf wenigen Seiten die wichtigsten Punkte erwähnen, und häufig auf weiterführende Literatur hinweisen.

Für die Zuordnung der Aufgaben zu den Aktivitäten schlägt Graham et al. (1997) die Anordnung in Tabellen vor:

		OPEN-Aktivitäten				
OPEN-Aufgaben		**Aktivität 1**	**Aktivität 2**	**Aktivität 3**	**...**	**Aktivität n**
	Aufgabe 1	F	R	F	...	M
	Aufgabe 2	D	D	F	...	F
	Aufgabe 3	O	F	O	...	F
	...	...	...	...	...	...
	Aufgabe n	D	F	O	...	O

Abbildung 3-6 – Gegenüberstellung von OPEN-Activities zu OPEN-Tasks

[63] Für eine detailliertere Beschreibung vgl. Graham (1997, S. 35ff).

[64] Die Originalbezeichnungen lauten: »user interaction and business issues«, »large-scale architectural issues«, »project management issues«, »quality issues«, »database issues« »distribution issues«, »modeling / building the system« und »reuse issues« (vgl. Graham 1997, S. 75ff).

In einer solchen Tabelle (vgl. Abbildung 3-6) finden sich alle verwendeten Aktivitäten und Aufgaben wieder. In der entsprechenden Tabellenzelle soll im Rahmen der Instanziierung für jedes Paar von Aktivität und Aufgabe angegeben werden, wie sinnvoll eine Nutzung der Aufgabe für das Bearbeiten einer Aktivität scheint. Dabei werden die Empfehlungen in 5 Stufen angegeben:

M: Mandatory (obligatorisch)
R: Recommended (empfohlen)
O: Optional
D: Discouraged (abgeraten)
F: Forbidden (verboten)

»Roadmaps« (nach Unhelkar, 2000) bieten eine alternative Darstellung für den Zusammenhang von Aktivitäten, Aufgaben und Ergebnissen, die für geforderte Ergebnisse die Einordnung in Aktivitäten und Aufgaben aufzeigt (vgl. Abbildung 3-7). Zu sehen ist die zuständige Person bzw. ihre Rolle sowie die Untergliederung der Aktivität in ihre Aufgaben und das erzeugte auslieferbare Ergebnis.

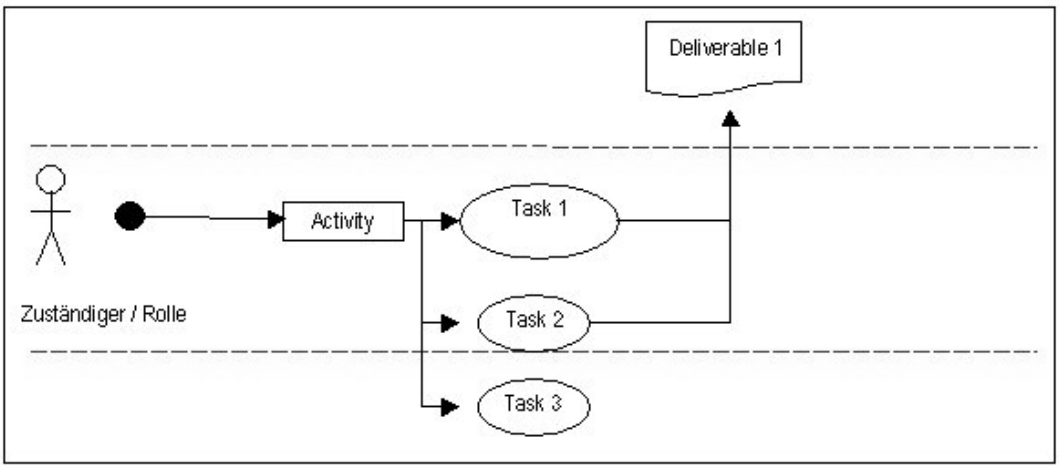

Abbildung 3-7 – Eine »Roadmap« zur Erstellung eines auslieferbaren Ergebnisses (aus Unhelkar 2000)

3.2.3. Techniken zur Ausführung von Aufgaben

Die Techniken beschreiben in OPEN *wie* die Aufgaben ausgeführt werden. Dabei können zur Ausführung einer Aufgabe eine oder mehrere Techniken, die selbst gewählt werden sollen, nötig sein. In Henderson-Sellers et al. (1998) sind dazu über 200 bewährte Techniken aus verschiedensten Quellen zusammengestellt, beschrieben und kritisch beleuchtet. Zusätzlich finden sich auf der dem Buch beiliegenden CD noch weitere Techniken, die in

die »experimentellere Techniken« und »nicht-objektorientierte / traditionelle Techniken mit eingeschränkter Nutzbarkeit in OPEN« eingeteilt sind, sowie ein Bereich mit »unterstützendem Referenzmaterial«[65]. Viele der empfohlenen Techniken werden von erfahrenen Entwicklern objektorientierter Systeme ganz automatisch verwendet. Trotzdem kann die OPEN-Zusammenstellung einerseits auf Gefahren einiger Techniken aufmerksam machen und andererseits neue Ideen geben. Für unerfahrenere Entwickler eignen sie sich außerdem als Grundlage für Schulungen, zum Selbststudium und als hilfreiches Nachschlagewerk.

Bei der großen Zahl der Techniken würde es den Rahmen dieser Arbeit deutlich übersteigen, sie alle vorzustellen. In den Anhängen B und C finden sich Listen der Aufgaben bzw. Techniken, um einen Einblick zu geben, worum es sich bei den Beschreibungen handelt. Entscheidend ist hier, daß eine Vielzahl an Techniken als hilfreiche Ressource zur Verfügung steht und von OPEN beschrieben und kritisch beleuchtet wird.

Da die Techniken nur bestimmen *wie* Aufgaben erledigt werden, ist es nicht unbedingt erforderlich, hier zu Projektstart eine konkrete Vorauswahl zu treffen. Der Projektleiter sowie erfahrene Entwickler können aber durchaus Empfehlungen und Hinweise geben, welche Techniken sich besonders für welche Aufgaben eignen; dies kann beispielsweise in Tabellenform (ähnlich wie in Abbildung 3-6) geschehen. Häufig werden sich in einer Organisation allerdings über die Zeit bewährte Techniken eingespielt haben, mit denen die Mitarbeiter schon gute Erfahrungen gesammelt haben. Dann gilt es diese ggf. durch neuere Techniken sinnvoll zu ergänzen.

Bei der Auswahl von Techniken muß auch beachtet werden, daß moderne Programme (z.B. sog. Integrierte Entwicklungsumgebungen), die zur Entwicklung von Software eingesetzt werden, einige Techniken vorgeben oder nahelegen[66]. Noch stärker als bei der Wahl der software-unabhängigen Techniken spielen bei der Auswahl der Softwarewerkzeuge die Vorlieben und Erfahrungen der Entwickler und Anwender eine entscheidende Rolle. In vielen (gerade kleineren) Projekten bestimmt die Auswahl der Teammitglieder gleichzeitig

---

[65] Die Original-Bezeichnungen lauten: »more experimental techniques«, »non-OO / traditional techniques with limited utility in OPEN« und »supporting reference material«.

[66] Einige Programme liefern sogar ihre eigene Vorgehensweise mit. So beinhaltet z.B. das Programm »Together Control Center« der Firma Togethersoft (vgl. Kruchten 1999 sowie http://www.togethersoft.com, Stand Februar 2001) in der Version 4.1 eine Beschreibung des »Feature Driven Development«.

einen Großteil der eingesetzten Softwareunterstützung. In anderen Projekten kann es wiederum sinnvoll sein, auch die benutzten Entwicklungswerkzeuge zu vereinheitlichen und damit beispielsweise den Rahmenbedingungen des Projektes, der Entwicklung des Marktes oder speziellen Kundenwünschen gerecht zu werden.

3.3. Flexibilität in Projekten mit OPEN

Um die Flexibilität von OPEN angemessen beurteilen zu können, muß grundsätzlich unterschieden werden, zwischen den von OPEN gelieferten Konzepten und Hilfestellungen auf der einen und ihren Umsetzungen in verschiedenen OPEN-Instanzen auf der anderen Seite. Ich werde hier nicht auf alle Lebenszyklusmodelle eingehen, sondern stellvertretend das Standardbeispiel des »contract-driven lifecycle model« (CDLM) herausgreifen, das zweifelsohne durch die ausführliche Beschreibung in der OPEN-Spezifikation selbst[67] eine zentrale, vorbildartige Stellung einnimmt. Anschließend daran sollen dann allgemeine Konzepte von OPEN betrachtet werden.

3.3.1. OPEN-Standardbeispiel

Eine Übersicht zum CDLM findet sich in Abbildung 3-2 (S. 53). Um hier strukturiert auf die wesentlichen Punkte einzugehen, werde ich wiederum die in Kapitel 2 entwickelten Fragen heranziehen:

(1a) Wie flexibel kann im CDLM mit Veränderungen umgegangen werden?

Gleich in diesem ersten Punkt tritt ein wesentlicher Schwachpunkt des CDLM zutage. Während eine Anpassung an die Projektgegebenheiten im Rahmen der Instanziierung zu Beginn des Projektes möglich ist (vgl. Frage 1b), gibt es keine ausreichende Flexibilität, um auf Veränderungen im Projektverlauf mit einer notwendigen Anpassung des Lebenszyklusmodells zu reagieren. Es wird davon ausgegangen, daß das Lebenszyklusmodell mit seinen Aktivitäten vollständig konstant bleibt; Änderungen sind nicht vorgesehen. Die Zuordnung der Aufgaben geschieht etwas dynamischer, indem noch während der Projektdurchführung entschieden werden kann, welche der empfohlenen Aufgaben gewählt werden. Erst auf der Ebene der Techniken findet sich schließlich vollständig die gewünschte

[67] Graham et al. (1997).

Flexibilität, indem hier die Teammitglieder selbständig (ggf. mit einer unterstützenden Empfehlung aus der OPEN-Instanz) entscheiden, welche Techniken sie benutzen.

Bei der konkreten Beschreibung der Aktivitäten fällt besonders der Bereich der Anforderungsermittlung auf:

In der Praxis werden während der Durchführung eines Projektes die Anforderungen sehr häufig umgestaltet und neu definiert. Erst im Verlaufe des Projektes, durch das Experimentieren mit Prototypen, durch Hinzuziehen der Endanwender und durch Ergebnisse des gegenseitigen Lernens (vgl. Floyd, 1994b) tauchen neue wichtige Anforderungen auf. Des weiteren klären sich Mißverständnisse zwischen Kunden und Entwicklern, die dazu führen können, daß der gesamte Projektplan neu strukturiert werden muß, häufig erst im Laufe des Projektes[68].

Die Anforderungserfassung selbst ist auch im CDLM relativ flexibel gestaltet. Es gibt mehrere Aktivitäten, die zur Erfassung, Validierung und Erweiterung der Anforderungen dienen (»Requirements Engineering«, »Analysis and Model Refinement«, »User Review«, »Consolidation« und »Evaluation«). Dabei gibt es jedoch einige Probleme: Erstens sind die meisten dieser Aktivitäten von vorne herein zeitlich begrenzt und können so nicht zu jeder Zeit der Entwicklung flexibel reagieren. Und zweitens werden wesentliche Veränderungen, die u.U. eine Umstrukturierung des Vorgehens notwendig machen, nicht vorgesehen und können daher auch im Lebenszyklusmodell nicht angemessen behandelt werden.

Auch die Behandlung der Dokumente bzw. auslieferbaren Ergebnisse muß im CDLM als eher unflexibel angesehen werden. Da die Zahl dieser definierten Ergebnisse[69] sehr groß ist, erscheint es fraglich, wie die Konsistenz gesichert werden soll. Graham et al. (1997) erwähnen zwar die Notwendigkeit eines Konfigurationsmanagements und die damit zu-

---

[68] Goldberg (1995) stellt eine sehr anschauliche Beschreibung dar, die deutlich macht, wie viele Abbildungen und Transformationen des Systems bei der Kommunikation zur Anforderungsermittlung benötigt werden: Der Kunde hat ein Konzept vom gewünschten System im Kopf (»Concept Space«), welches er in eine sprachliche Version transformieren muß, um es dem Entwickler mitzuteilen (»Articulation Space«). Dieser hat seinerseits verschiedene Techniken zur Hand (»Analysis Methods«), um im Gespräch mit dem Anwender (»Analysis Model Space«) ein Modell zur Analyse zu erstellen. Erst auf Basis dieses Modells wird schließlich das System entwickelt (»Design and Implementation Model Space«).

[69] Graham et al. (1997, S. 147ff) erwähnt weit über 40 verschiedene Typen, die natürlich nicht alle für jedes Projekt erstellt werde müssen, sehr viele davon werden aber empfohlen.

sammenhängenden Probleme, wenn man aber von einer Veränderlichkeit der zu den Dokumenten führenden Gegebenheiten ausgeht, scheint bei einer solchen Zahl an Dokumenten der entstehende Aufwand nicht zu bewältigen. Außerdem stehen für die angegebenen Dokumente bisher keine Vorlagen zur Verfügung anhand derer man eine klare Abgrenzung erkennen könnte.

Die Anpaßbarkeit an spezifische Projektsituationen und das Projektteam vor Projektstart findet dagegen in OPEN eine bessere Berücksichtigung:

(1b) Inwieweit ermöglicht das CDLM Anpassungen an die spezifischen Bedingungen und die beteiligten Personen zu Projektbeginn?

Betrachtet man das CDLM für sich allein, dann ist eine Anpassung an die Projektgegebenheiten nur bedingt möglich. Wiederholt wird aber darauf hingewiesen, daß OPEN- Lebenszyklusmodell wie beispielsweise das CDLM i.A. nicht einfach übernommen werden können. Stattdessen müssen sich die beteiligten Personen mit der Instanziierung von OPEN und der Anpassung an die gegebene Situation beschäftigen.

Als erste Übersicht über die Organisation des Projektes eignet sich die Grafik zum Lebenszyklusmodell des CDLM (Abbildung 3-2, S. 53) zudem nicht besonders gut. Auf den ersten Blick erschließt sich nicht, in welchem Verhältnis die einzelnen Aktivitäten zueinander stehen, wo zeitliche und wo inhaltliche Abhängigkeiten bestehen und ob es sich überhaupt um ein iterative Vorgehensweise handelt oder nicht. Als erste Übersicht ist hier eine anschaulichere Grafik erforderlich.

(2) Welche praktische Unterstützung liefert das CDLM für die täglichen Aufgaben im Projekt?

Zur praktischen Arbeit der Teammitglieder liefert das CDLM mit der großen Anzahl der beschriebenen Techniken eine gute Unterstützung. Zu allen Bereichen der Softwareentwicklung werden Hilfen und Quellen weiterführender Literatur bereitgestellt, die die tägliche Arbeit erheblich erleichtern können. Es fehlt in der Grundbeschreibung von Graham et al. (1997) allerdings an Vorlagen und Details für die zu erstellenden Ergebnisse.

3.3.2. OPEN allgemein

Diese Nachteile sind allerdings nicht alles Nachteile von OPEN insgesamt. Vielmehr birgt die Umsetzung des CDLM einige Schwächen, die in anderen Ausprägungen von OPEN leicht umgangen werden können.

(1a) Wie flexibel kann innerhalb der Vorgehensweise mit Veränderungen umgegangen werden?

Ein wesentlicher Punkt, der in keinem der beschriebenen Lebenszyklusmodelle zu OPEN Beachtung findet, ist das Feedback zum Vorgehen und der Vorgehensweise selbst. In heutigen dynamischen Projekten sollten alle beteiligten Personen die explizite Möglichkeit haben, ihre Kritik am Vorgehen zu äußern und so Einfluß auf eine Umstrukturierung oder Neugestaltung der Vorgehensweise haben. Derniame (1999) beschreibt diese Notwendigkeit im Vergleich des gelebten Vorgehens (»real-world process«) mit der definierten Vorgehensweise (»process model«) wie folgt:

> „However, any real-world software process is a creative process involving numerous human actors; it is dynamic and evolves continuously, and cannot be reduced to the programming of automata. [...] The process model must coordinate activities and manage the information flow, and it must adapt to any evolution of the real world process. This, as will be seen, is a major issue." (Derniame 1999, S. 11)

Die von Derniame daraufhin geforderte Definition eines »Meta-Process« (also einer Vorgehensweise zur Beschreibung und Anpassung der verwendeten Vorgehensweise) hat meiner Ansicht nach vor allem theoretische Relevanz. In der Praxis wird sich zwar auf die Ebene des Meta-Prozesses begeben, wenn eine Anpassung der Vorgehensweise vorgenommen wird; dies geschieht aber weniger formal und explizit als vielmehr durch informelle Gespräche und Umstrukturierungen. Auch wenn eine Formalisierung für Praxisprojekte im Hinblick auf die enge Zeitplanung nicht angemessen erscheint, so ist eine Verankerung der Selbstreflexion in der Vorgehensweise dringend erforderlich.

Die Veränderlichkeit der Rahmenbedingungen hat auch einen großen Einfluß auf die Erstellung der Ergebnisse. Späte Änderungen an einem der zahlreichen Dokumente in OPEN müßten nachgebessert werden, um die Konsistenz der Dokumente zu gewährleiste. Dies kann zu einem hohen Arbeitsaufwand führen, der die eigentliche Entwicklung hemmt[70].

---

[70] Eine systematische Auseinandersetzung mit diesem Problem liefert Madhavji (1992) mit dem »Prism Model of Changes«. In diesem Modell werden zunächst einmal alle »items of change« – also alles Veränderliche, wie Personen, Gesetze, Strategien, Prozesse, Dokumente etc. – festgestellt und aufgenommen. Dann sind für jede Veränderung dieser »items« zahlreiche Eigenschaften zu beschreiben, die innerhalb der »Change Structure« gespeichert werden. Daneben steht mit der »Dependency Structure« eine weitere Infrastruktur zur Verfügung, in der alle Abhängigkeiten der

Eine radikale Alternative zum Umgang mit Ergebnissen stellt Beck (1999) für XP vor: Dort werden außer dem Quellcode *keine* Dokumente aufbewahrt, so daß Veränderungen zwar eine Überarbeitung des Codes erfordern, dies aber keine Änderungen an weiteren Dokumenten oder anderwertigen Ergebnissen nach sich zieht. Dies ist natürlich mit dem generellen Aufbau von OPEN (Erzeugen von auslieferbaren Ergebnissen in Aktivitäten) nicht vereinbar. Darüber hinaus scheint dieses Vorgehen für viele Projekte in der Praxis nicht ratsam. Beck scheint hier den Nutzen von Grafiken und Übersichtsbildern deutlich zu unterschätzen[71]. Trotz des Extraaufwandes der notwendig ist, solche Übersichts- oder Designdokumente zu erstellen und parallel zum Code zu pflegen gibt es doch viele Gelegenheiten, in denen Anwendungsfälle, Klassendiagramme, Architekturmodelle, Netzwerktopologien und andere Grafiken sehr hilfreich – wenn nicht sogar unentbehrlich – sind.

Der Aspekt der Kommunikation und Kooperation sowie die persönlichen Eigenarten der am Projekt beteiligten Personen, kommen in den Darstellungen der OPEN-Literatur zu kurz. Zyklische Entwicklung zusammen mit zahlreichen Reviews (die ja Kommunikation implizieren) werden in den vorgestellten Lebenszyklusmodellen zwar in gewissem Maße unterstützt, aber dies scheint nicht ausreichend, um der zentralen Rolle der beteiligten Personen gerecht zu werden.

(1b) Inwieweit ermöglicht die Vorgehensweise Anpassungen an die spezifischen Bedingungen und die beteiligten Personen zu Projektbeginn?

Für eine solche Anpassung steht in OPEN die Instanziierung mit mehreren Beispielen (vgl. Abschnitt 3.2) zur Verfügung. Nützlich ist OPEN als gemeinsame Basis auch daher, weil so der Austausch über die Organisation und das Vorgehen in Projekten erleichtert wird. Es ist generell einfacher, eine Vorgehensweise, die auf einer bekannten Basis erstellt ist, an die eigene Situation anzupassen (und dabei vielleicht noch auf andere Lösungen, die auf derselben Basis beruhen zurückgreifen zu können), als eine vollständig eigene Vorgehensweise zu entwickeln. In dieser Instanziierung in OPEN ist es auch möglich, Spezial- und Kleinprojekte zu unterstützen. Dafür können eigene Aktivitäten oder Aufgaben zum

---

»items of change« beschrieben sind. Ständiges Feedback hält dabei die Infrastrukturen aktuell, und Metafeedback sorgt für die Nutzbarkeit des »Prism Model of Changes« insgesamt.

[71] Beck (1999, S. 111f) erwähnt sie als »pictures«, samt ihrer Vorzüge als *temporäre* Hilfe. Die »pictures« werden aber nach der Unterstützung bei einem Gespräch oder einem Gedanken sofort wieder verworfen.

OPEN-Vorrat hinzugenommen werden oder aus dem umfangreichen Vorrat eine Auswahl getroffen werden.

Die Darstellung der Tabellen als Zuordnung von Aufgaben zu Aktivitäten und ggf. von Techniken zu Aufgaben (vgl. Abbildung 3-6) ist relativ einfach zu erstellen und gibt eine erste Übersicht, welche Aufgaben wo verwendet werden. Sie bietet allerdings bei der Projektdurchführung nur begrenzte Hilfe. Insgesamt wird die Anpassung der Vorgehensweise an die spezifischen Bedingungen zu Projektbeginn in OPEN aber gut unterstützt.

(2) Welche praktische Unterstützung liefert die Vorgehensweise für die täglichen Aufgaben im Projekt?

Wie bereits erwähnt, liefern die zahlreichen Beschreibungen von Aufgaben und Techniken eine breite Unterstützung für die Projektpraxis. Abbildung 3-8 zeigt, daß je nach Gestaltung des Lebenszyklusmodells und Auswahl von Aktivitäten, Aufgaben und Techniken, OPEN-konforme Vorgehensweisen sowohl recht schematisch und handlungsanleitend als auch hoch flexibel sein können. So wird mit OPEN eine Möglichkeit geschaffen, innerhalb eines einheitlichen Konzepts, ein großes Spektrum von schematischen, anpaßbaren und flexiblen Vorgehensweisen zu nutzen und über diese zu kommunizieren.

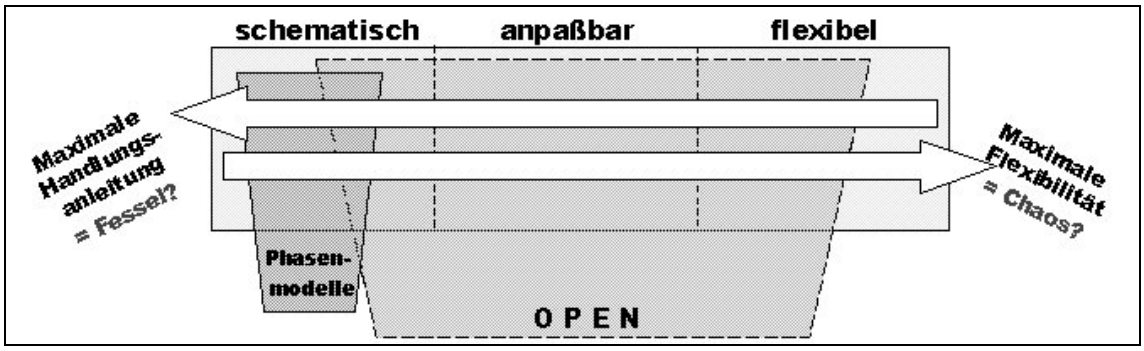

Abbildung 3-8 – Die Software-Expedition zwischen Schema und Flexibilität

Nachdem ich mit dem CDLM in diesem Kapitel schon auf eine recht schematische Vorgehensweise von OPEN eingegangen bin, soll das folgende Kapitel nun mit der Software-Expedition Gedanken einer besonders flexiblen Vorgehensweise aufzeigen.

4. Die Software-Expedition – Flexibilität im Fokus

Im zweiten Kapitel habe ich gezeigt, daß ein flexibles Vorgehen ohne starre Handlungsvorschriften zwar notwendig ist, Beschreibungen von typischen Aufgaben u.ä. aber andererseits trotzdem eine Hilfe zur Entwicklung großer, dedizierter und objektorientierter Anwendungssoftware darstellen können. Das vorliegende Kapitel stellt nun mit der Software-Expedition[72] einen aktuellen Management-Ansatz vor, der versucht, über die Orientierung der Softwareentwicklung am Leitbild moderner Expeditionen, größtmögliche Flexibilität mit handlungsleitenden Elementen zu vereinbaren.

Dabei soll hier darauf hingewiesen werden, daß sich die vorliegende Arbeit nur mit einigen Ausschnitten der Software-Expedition beschäftigt. In der Literatur über die Software-Expedition stehen im Gegensatz zu dieser Arbeit die generelle Sichtweise von Softwareentwicklung und die Organisation des Entwicklungsteams im Zentrum, während hier der Realisierung technischer Unterstützung bei der Softwareentwicklung etwas mehr Raum gegeben wird. Trotzdem liefert die Software-Expedition wertvolle Ideen für die Zusammenstellung einer flexiblen Vorgehensweise in der vorliegenden Arbeit. Ich werde daher zunächst die generelle Idee, die Expeditionssicht als grundlegende Sichtweise für die Softwareentwicklung zu benutzen, vorstellen. Aus dieser Sichtweise und dem darauffolgend beschriebenen Ablauf und Aufbau von Software-Expeditionen, werde ich schließlich die wesentlichen Aspekte der Flexibilität anhand der im zweiten Kapitel aufgestellten Fragen und Kriterien herausstellen und zur Nutzung im nachfolgenden Kapitel aufbereiten.

4.1. Die Expeditionssicht

Bei der Software-Expedition geht es darum, zur Orientierung in der Softwareentwicklung das Vorgehen in einer Expedition anstelle der ingenieursmäßigen Durchführung eines Projektes als Leitbild zu nutzen. Der Softwareentwicklung liegt dann die Expeditionssicht zugrunde und nicht die Projektsicht, in der Softwareentwicklung als industrieller Produktionsprozeß aufgefaßt wird und „an dessen Ende Software als industriell gefertigtes Produkt steht“ (Mack 2001, S. 52).

---

[72] Die im vorliegenden Kapitel beschriebenen Ideen und Konzepte beziehen sich, wenn nicht explizit anders angegeben, vollständig auf Mack (2000a), Mack (2000b) und Mack (2001).

Mack bezieht sich vor allem auf Expeditionen „in ein (teilweise) unbekanntes Gebiet“ (Mack 2001, S. 230) bei dem sich das Expeditionsteam „verschiedenen sich ändernden Umwelteinflüssen und schwer kalkulierbaren Risiken“ (ebenda) aussetzt.

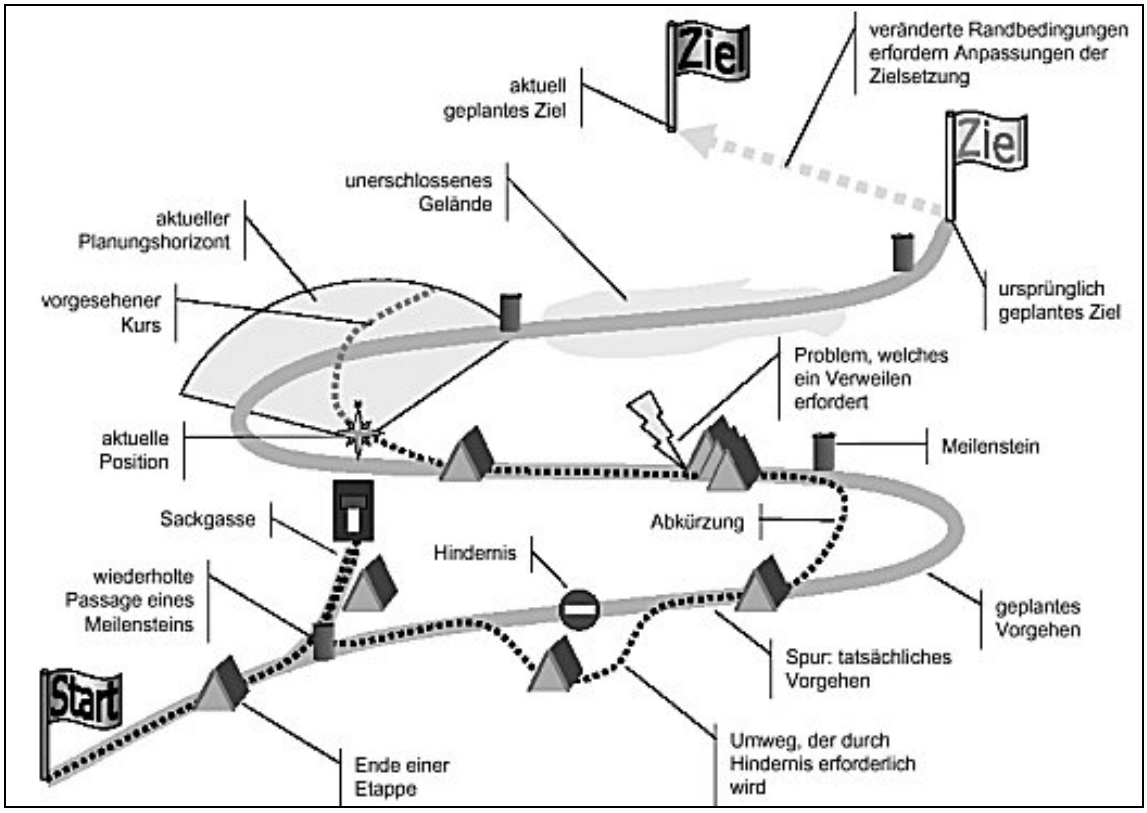

Abbildung 4-1 – Exemplarisches Vorgehen einer Expedition (aus Mack 2001, S. 148)

Abbildung 4-1 zeigt charakteristische Elemente einer solchen Expedition, die auf ein Vorgehen in der Softwareentwicklung übertragen werden können: Zunächst gibt es einen Startpunkt (vergleichbar beispielsweise mit der Initiierung des Entwicklungsvorhabens) sowie ein Ziel (z.B. die Inbetriebnahme des Systems), das sich allerdings im Laufe der Expedition durch unvorhergesehene Ereignisse und Veränderungen der Annahmen verschieben kann. Start und Ziel verbindet eine Spur, die nur in seltenen Fällen der kürzesten Verbindung entspricht. Viele Veränderungen und Unwägbarkeiten (finanzieller Rahmen, politische Entscheidungen etc.) und unerschlossene Gebiete (z.B. unbekannte Technologien) zwingen das Team zum Erschließen von neuen alternativen Routen vor Ort (d.h. also *während* der Entwicklung und nicht in einer Planungsphase *vor* Projektstart). Nur kleine Teile dieser Spur liegen zu jedem Zeitpunkt im aktuellen Planungshorizont und können überblickt und somit sinnvoll geplant werden. Im Vorgehen einer Expedition finden sich weiterhin die aus der Softwareentwicklung bekannten statischen Meilensteine, die ein angestrebtes Zwischenergebnis im Vorgehen festlegen und somit inhaltlich orientiert sind.

Von den Meilensteinen unterscheidet Mack die Etappen[73], die ein Tages- (oder Wochen-) Pensum beschreiben und wegen unterschiedlicher Leistungen je nach Situation und Momentanproblemen durchaus unterschiedliche inhaltliche Fortschritte ergeben[74].

Neben dem generellen Charme dieses Ansatzes, der sich vor allem darin zeigt, daß sich in der Softwareentwicklung immer wieder neue Elemente entdecken lassen, die eine Parallele zur Expeditionen geradezu aufdrängen, ist für die vorliegende Arbeit zentral, daß auch in einer Expedition auf unvorhersehbare Veränderungen fokussiert wird – Wandel und Flexibilität stehen im Zentrum:

> „Charakteristisch ist auch eine hohe Flexibilität im Vorgehen, die wesentlich zum Expeditionserfolg beiträgt. Dies hängt insbesondere mit dem Umstand zusammen, daß die Expeditionsroute oft nicht detailliert im Voraus geplant werden kann: Die konkrete Route läßt sich meist erst vor Ort genau bestimmen.“ (Mack 2001, S. 133)

> „Die Beteiligten folgen weniger einem verbindlichen Schema der Softwareentwicklung als einem flexiblen, aus der spezifischen Situation entstehenden Prozeß, dessen Verlauf durch ständige Reflexion und Kalibrierung an ein verändertes Umfeld bestimmt wird.“ (Mack 2001, S. 166)

Durch eine zu diesem Anlaß angestellte Studie[75] belegt Mack darüber hinaus die These, daß in der aktuellen Projektpraxis ohnehin schon viele Begriffe aus dem Bereich der Landschaftsbeschreibung und des Reisens benutzt werden. Läßt man sich nun darauf ein, Softwareentwicklung als Expedition zu betrachten, dann fällt schnell auf, daß viele Begriffe der bisher verwendeten Sprache ohnehin besser zum Bild der Expedition als zu einem planmäßigen Produktionsprozeß passen; Begriffe wie »Vorgehen«, »Meilenstein«, »Etappen«, »Neuland betreten« oder »Systemlandschaft«[76] gehören schon lange zum Sprach-

---

[73] Vgl. Mack (2001, S. 133f).

[74] Diese Etappen sind am ehesten vergleichbar mit Iterationen in der Softwareentwicklung, wobei Iterationen in der Literatur typischerweise einen Zeitraum von einigen (oft 2-4) Wochen umfassen (vgl. McDermid, S. 27/22; Graham 1997, S. 190f.; Oestereich 1999, S. 96; Beck 2000, S. 133). Mack empfiehlt eine Etappendauer von einem bis maximal 14 Tagen, typischerweise aber eine oder zwei Wochen (vgl. Mack, S. 186).

[75] Mack führte von März bis Juni 2000 Interviews mit 17 Experten aus dem Bereich der Softwareentwicklung – vgl. Mack (2001, S.77).

[76] Vgl. Mack (2001, S. 17).

schatz der Softwareentwicklungspraxis. Eine konsequente Fortführung dieser alltäglichen Sprache der Softwareentwicklung führt zur Expeditionssicht[77]:

> „Die *Expeditionssicht* ist eine Sichtweise im Sinne eines Leitbildes. Sie orientiert den Prozeß der Softwareentwicklung an den zentralen Merkmalen und Leitideen moderner Expeditionen, insbesondere an der zu bewältigenden Herausforderung, dem zentralen Stellenwert des Expeditionsteams, der Art der Vorbereitung, dem proaktiven und prospektiven Umgang mit Risiken sowie der Orientierung und einer flexiblen Vorgehensweise in einem unbekannten Gebiet." (Mack 2001, S. 139)

Die Verwendung dieser Sichtweise hat einige Auswirkungen auf das Vorgehen in der Praxis, die zu Unterschieden im Vergleich zur Softwareentwicklung mit zugrunde liegender Projektsicht führen. Mack (2001, S. 155f) identifiziert Verschiebungen

- vom treibenden Moment einer rational-logischen Lösung eines gegebenen Problems hin zu einer sozialen und organisationalen Bewältigung der mit der Softwareentwicklung verbundenen Herausforderungen;
- von der Annahme stabiler Randbedingungen zu einer bewußten Integration ihrer Wandelbarkeit als Aspekt von Software-Vorhaben in den Entwicklungsprozeß;
- von der Planung als zwingend einzuhaltende Vorschrift hin zu einer Auffassung von Planung als Ressource;
- von einem schematischen Problemlösungsprozeß hin zu einem Prozeß, der sich durch die Interaktion der Beteiligten situativ im Verlauf des Vorhabens entfaltet;
- von einer Steuerung zum Abgleich von Soll- und Ist-Werten hin zu einer vom Team organisierten und von »innen« kommenden situativen Reorientierung;
- „von einer Vorstellung der beteiligten Menschen als »Ressource« hin zu einer Wahrnehmung der Beteiligten als verantwortungsvolle Individuen" (Mack 2001, S. 156).

---

[77] Die im folgenden Zitat verwendeten Begriffe Leitbild und Sichtweise sollen in der vorliegenden Arbeit nicht unterschieden werden (vgl. Mack 2001, S. 232 bzw. 235).

Im folgenden Abschnitt werde ich am Ablauf einer Software-Expedition skizzieren, wie sich diese Verschiebungen und die Einbeziehung der Expeditionssicht allgemein auf das Vorgehen in der Praxis auswirken.

4.2. Ablauf einer Software-Expedition

Während hier vorgestellt werden soll, wie Mack (2001) einen Ablauf einer Software-Expedition beschreibt, so gilt es zu beachten, daß der Ablauf der eigentlichen Entwicklung nicht im Zentrum von Macks Untersuchungen steht:

> „Die Software-Expedition ist in erster Linie ein Ansatz zur Organisation der an einem Software-Entwicklungsprozeß beteiligten Menschen und weniger ein technisches Vorgehensmodell zur Produktion von Softwaresystemen. Dennoch ist aus Sicht der Beteiligten der Prozeß der softwaretechnischen Entwicklung ebenso wichtig wie die Koordination der Zusammenarbeit zwischen den Beteiligten." (Mack 2001, S. 200)

Es geht Mack also vor allem um das Management der beteiligten Personen. Die ausführlichere softwaretechnische Betrachtung sind also nicht direkt Gegenstand der bisher verfügbaren Literatur zur Software-Expedition. Vielmehr stellt der Ansatz der Software-Expedition frei, welches softwaretechnische »Handwerkszeug« (Programmierung, Teststrategie, Modellierungssprache etc.) zur Unterstützung hinzugezogen wird, gibt aber konkrete Hinweise zur Auswahl. Drei verschiedene Ansätze der Softwaretechnik stellt Mack für diese technische Unterstützung vor und beschreibt kurz ihren Bezug zur Software-Expedition[78].

Insgesamt definiert Mack die Software-Expediton wie folgt:

> „Eine *Software-Expedition* ist eine Vorgehensweise zu professionellen Entwicklung von Softwaresystemen, die sich am Leitbild der Expedition orientiert. Sie ist insbesondere durch eine starke Teamorientierung und einen hohen Anteil an Selbstorganisation des Expeditionsteams, ein explorativ Vorgehen sowie eine situative Reorientierung des Teams im Anwendungsgebiet charakterisiert." (Mack 2001, S. 166)

---

[78] Bei diesen Ansätzen handelt es sich um die *Unified Modeling Language* UML, den *Unified Process* UP (vgl. Mack 2001, S. 200ff) und *Extreme Programming* XP (ebenda sowie in Mack 2000b).

Die hervorgehobene „*Selbstorganisation des Expeditionsteams*“ sowie „*situative Reorientierung im Anwendungsgebiet*“ stellen die hier wesentlichen Unterschiede zur Beschreibung von anderen Vorgehensweisen (vgl. auch Abbildung 2-2 auf S. 15) dar. In Abbildung 4-2 sind die für die vorliegende Arbeit interessanten Aspekte der Software-Expedition noch einmal herausgestellt[79]. Anstelle der klar umrissenen Beschreibungen von Aktivitäten, Aufgaben und Techniken (vgl. wiederum Abbildung 2-2) sind hier in der Praxis beobachtbare, situative und erfahrungsbasierte Aktivitäten zu sehen, die einen hohen Grad an Flexibilität ermöglichen[80].

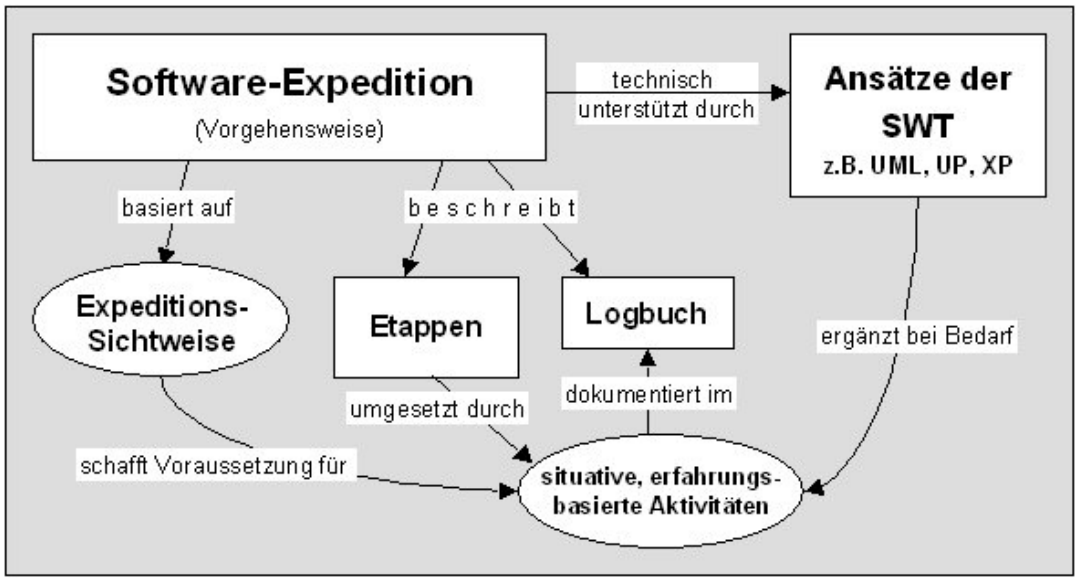

Abbildung 4-2 – Vorgehensweise einer Software-Expedition

Diese situativen Aktivitäten werden, wie bereits erwähnt, durch andere softwaretechnische Ansätze technisch unterstützt. So wird beispielsweise die verbreitete Technik der Anwendungsfallanalyse zur Anforderungserfassung in der Software-Expedition zwar nicht explizit eingeführt, eine Verwendung ist aber durchaus möglich und wird empfohlen[81]. Bei der Auswahl zusätzlicher Aktivitäten oder Techniken sowie bei der situativen Reorientierung vor Ort bietet die im vorigen Abschnitt beschriebene Expeditionssicht die wesentliche Ent-

[79] Für einen Gesamtüberblick der Software-Expedition s. Mack (2001).

[80] Zur Bedeutung der verwendeten Symbole, S. Erklärung in Abbildung 2-1.

[81] Für eine wirklich leichtgewichtige und flexible Softwareentwicklung mit möglichst wenigen Dokumenten empfiehlt Mack (2001, S. 205f) neben dem Code nur noch das Logbuch und Anwendungsfälle (»use cases«). Dies ist aber nicht als Vorschrift, sondern als Erfahrungswert zu sehen. So sind durchaus Software-Expeditionen mit größerem Dokumentenumfang vorstellbar (wie z.B. bei Unterstützung durch den Unified Process – vgl. Mack 2001, S. 205ff).

scheidungshilfe, indem sie das Augenmerk auf die in der modernen Softwareentwicklung kritischen Aspekte legt.

Neben den schon im vorigen Abschnitt beschriebenen Etappen, welche der Koordination einer konsequent iterativen und inkrementellen Entwicklung dienen, ist das *Logbuch* ein Konzept zur Durchführung einer Software-Expedition:

> „Ein wichtiges Instrument zur Dokumentation des Vorgehens auf Expeditionen ist das Logbuch, in dem die wichtigsten Ereignisse, besondere Situationen, Anmerkungen zur Ausrüstung etc. festgehalten werden." (Mack 2001, S. 150)

> „Der Expeditionsleiter führt das Logbuch mit dem Ziel, das Vorgehen der laufenden Software-Expedition (Nachvollziehbarkeit in Zweifelsfällen) zu dokumentieren und die damit verbundenen Lernergebnisse und Erfahrungen an zukünftige Software-Expeditionen weitergeben zu können." (Mack 2001, S. 188)

Dabei soll die Aktualisierung des Logbuchs nur „rund 10 bis 20 Minuten pro Tag" (ebenda) in Anspruch nehmen. Der in der Expeditionssicht herausgestellte hohe Stellenwert von Erfahrung wird also auch hier deutlich: Das Logbuch dient vor allem dazu, die Erfahrungen aus der Praxis zu reflektieren und Anhaltspunkte festzuhalten, um sie später für andere Software-Expeditionen zur Verfügung stellen zu können.

In Expeditionen hat die *Zusammenarbeit im Team* den zentralen Stellenwert und ist der in der Projektsicht verbreiteten maximalen Arbeitsteilung i.A. vorzuziehen. Das Expeditionsteam enthält beispielsweise neben Entwicklern und Organisationsberatern auch die Anwender. In den allermeisten Expeditionen ist es ebenso undenkbar, einzelne Teammitglieder nach der Hälfte der Zeit gegen andere auszutauschen, wie das Team gegen Ende der Entwicklung um neue Entwickler aufzustocken, um den gesetzten Zeitrahmen zu halten. Dies deckt sich wiederum mit vielfältigen Beobachtungen aus der Praxis, an denen immer wieder festgestellt wurde, daß der erhöhte Kommunikations- und Anlernaufwand, den die Integration neuer Teammitglieder unweigerlich mit sich bringt, sehr leicht den Nutzen der zusätzlichen Arbeitskräfte überschattet. Insgesamt geht es also in einer Software-Expedition darum, von Beginn an ein gutes Team zusammenzustellen und dieses Team dann durch partnerschaftliche Führung und individuelle Selbstorganisation ohne schematische Vorschriften bei guter Motivation und so maximal produktiv zu halten.

Den Entwicklungs- und Einsatzkontext (vgl. Abbildung 2-1) faßt Mack im Bild der Expedition zum *Territorium der Softwareentwicklung* zusammen. Abbildung 4-3 zeigt typische Elemente eines Territoriums mit dem Anwendungsgebiet. Das Anwendungsgebiet ist dabei die Umgebung der Organisation in der das zu entwickelnde System eingesetzt werden soll. Hier wird deutlich, wie viele konkrete Personen, Regelungen und Phänomene Einfluß auf Erfolg oder Mißerfolg der Softwareentwicklung haben können. Gerade weil viele dieser Phänomene und die Ansichten der Beteiligten veränderlich sind, ist es so wichtig im Vorgehen der Softwareentwicklung, flexibel auf diese Veränderungen reagieren zu können.

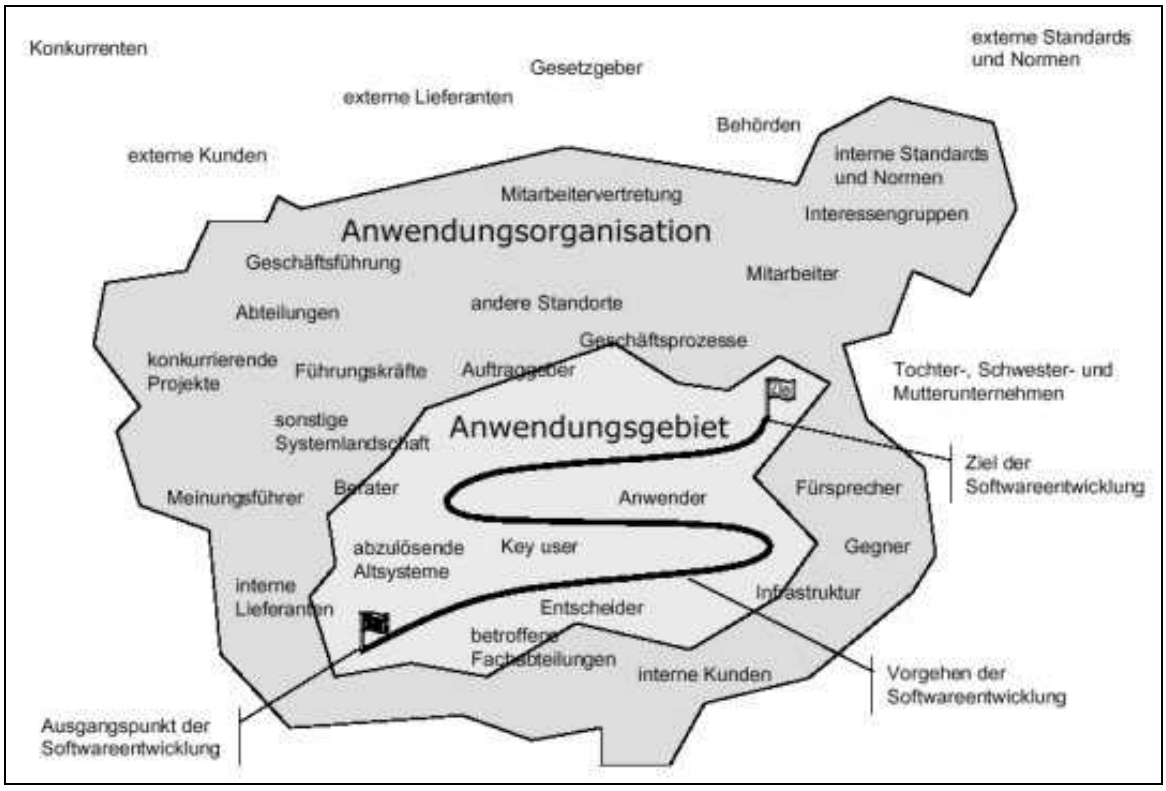

Abbildung 4-3 – Territorium der Anwendungsorganisation (aus Mack 2001, S. 143)

Die Softwareentwicklung selbst bezeichnet Mack dann als Exploration des Territoriums. Betont wird dadurch das schrittweise Erkunden des Geländes – also das Kennenlernen der entscheidenden Personen, die Klärung der Problemstellung, die Einarbeitung oder gar Umgestaltung von Arbeitsabläufen innerhalb der Anwendungsorganisation usw. Diese Exploration findet dann iterativ und inkrementell in Etappen statt und gewährleistet so die benötigte Flexibilität.

Risiken frühzeitig zu identifizieren und zu managen ist bei Expeditionen und auch Software-Expeditionen von besonderer Bedeutung. Dabei ist das Risikomanagement allerdings nicht durch Aufstellen und Abarbeiten einer Liste der Risiken zu Projektbeginn erledigt.

Vielmehr handelt es sich um eine kontinuierliche Aufgabe, in der auch gerade mit solchen Risiken umgegangen werden muß, die sich durch Veränderungen der Randbedingungen erst *während* des Entwicklungsprozesses manifestieren. Dabei wird empfohlen, alle Teammitglieder ständig über den aktuellen Stand der Risiken und auch der *Ressourcen* (verbrauchte und verbleibende Arbeitszeit, Budget und Aufgaben sowie eine Liste der noch nicht behobenen Fehler – Menschen werden nicht als Ressource gesehen) informiert zu halten[82].

Ich werde nun auf den typischen Ablauf einer Software-Expedition eingehen: Software-Expeditionen lassen sich typischerweise in drei Hauptabschnitte unterteilen: Die eigentliche Durchführung vor Ort – also im Territorium der Anwendungsorganisation – wird zeitlich eingerahmt von der Vorbereitung und der Nachbereitung „zu Hause“ (vgl. Abbildung 4-4).

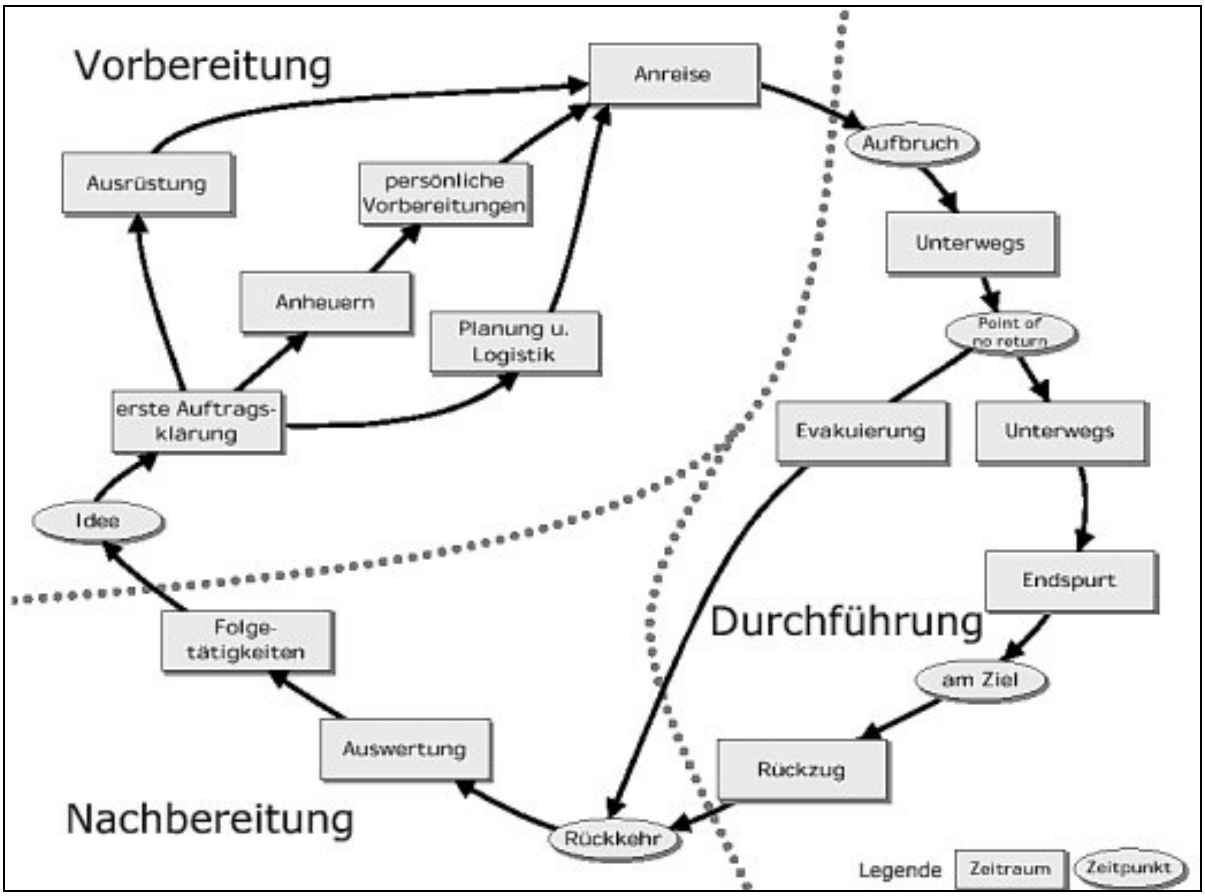

Abbildung 4-4 – Ablauf einer Software-Expedition (aus Mack 2001, S. 148)

[82] Vgl. Mack (2001, S. 201).

Nach der Idee zu einem bestimmten Softwareentwicklungsvorhaben und der Formulierung in einem Auftrag gibt es innerhalb der *Vorbereitung* noch einige Dinge zu erledigen: Bei der Zusammenstellung des Teams beispielsweise (idealerweise 6-8, maximal 10 Personen[83]) sollte neben technischen Fähigkeiten, die sich in der persönlichen Vorbereitung im Training noch ausbauen lassen, vor allem auch auf soziale Kompetenzen geachtet werden.

Auch im Bereich der Ausrüstung findet sich in der Expedition ein griffiges Leitbild. Mack identifiziert als Charakteristika für expeditionstaugliche Ausrüstungsgegenstände unter anderem (mit den korrespondierenden Übertragungen auf Software-Expeditionen in Klammern)[84]:

- leichtes Gewicht / Kompaktheit (einfache Konfiguration und Einarbeitung)
- Robustheit (Fehlertoleranz, Belastbarkeit, keine Abstürze)
- universelle Einsatzfähigkeit (geringe Belastung durch wenige universelle Werkzeuge)
- Einsatzfähigkeit unter den erwarteten Bedingungen (Angemessenheit und Performanz der Werkzeuge)

Im Bild der Expedition wird auch sofort deutlich, mit welchen Umständen und Schwierigkeiten das Hinzufügen zusätzlicher Ausrüstungsgegenstände (z.B. neuen Programmiertools in Software-Expeditionen) verbunden ist. Neben der Beschaffung und dem Transport muß das Team im Umgang mit den neuen Gegenständen geschult werden, um einen wirklichen Nutzen aus ihnen ziehen zu können. Daher muß eine endlose Diskussion um die zu verwendenden Ausrüstungsgegenstände (bzw. Software-Tools) verhindert werden, die durch das Aufkommen neuer Tools sowie durch verschiedene Präferenzen der einzelnen Teammitglieder aufkommen kann. Mack schlägt vor, die Recherche und Evaluation von Werkzeugen zu einem „festgesetzten Zeitpunkt“ (Mack 2001, S. 182) zu beenden, um eine solche belastende endlose Diskussion zu verhindern.

Vor der eigentlichen Anreise in das Anwendungsgebiet der Software-Expedition (dem Einsatz beim Kunden) gilt es noch, die initiale Planung zu betrachten. Allgemein wird Planung in der Software-Expedition im Gegensatz zu anderen Ansätzen nicht als alles be-

---

[83] Vgl. Mack (2001, S. 180).

[84] Vgl. Mack (2001, S. 181f).

stimmende, treibende Vorschrift, sondern eher als Ressource aufgefaßt, derer sich das Team bedienen kann und anhand derer der bisherige Fortschritt nachvollzogen werden kann. So findet eine Planung des gesamten Weges in der Vorbereitung nur sehr grob statt und fokussiert auf die erste Etappe sowie die ersten Schritte nach der Ankunft im Anwendungsgebiet. Zusätzlich sollten hier zu erwartende Risiken identifiziert und ein allgemeines Risikomanagement-Konzept beschlossen werden.

Die eigentliche *Durchführung* der Software-Expedition bewegt sich zwischen den Zeitpunkten des Aufbruchs (Beginn der Entwicklung vor Ort) und der Rückkehr. Zunächst wird anhand von Prototypen zusammen mit Anwendern und Entscheidungsträgern diskutiert, ob das Entwicklungsvorhaben wie begonnen zum Ziel geführt werden kann. Nach ca. 10-20% der geschätzten Expeditionsdauer (am »Point of no return«) sollte diese Diskussion abgeschlossen sein und das System tatsächlich weiterentwickelt oder ggf. eingestellt werden (»Evakuierung«) – das Team ist im Anwendungsgebiet „unterwegs“. Kurz vor Erreichen des Ziels steht typischerweise ein Endspurt, in dem letzte dringliche Fehler ausgemerzt werden und die endgültig freizugebende Version des Systems erstellt und in Betrieb genommen wird.

Der an die Durchführung anschließende Abschnitt der *Nachbereitung* dient vor allem dazu, die in der Expedition gemachten Erfahrungen zu reflektieren und teilweise für spätere Vorhaben zu dokumentieren. Aus der Zusammenfassung der Ergebnisse und Lernerfahrungen können sich auch Ideen für neue Expeditionen oder für die Weitergabe der Erfahrungen in Form von Fachvorträgen oder Veröffentlichungen ergeben.

Der in Abbildung 4-4 dargestellte Ablauf einer Software-Expedition ist nur als ein möglicher Ablauf zu sehen. Software-Expeditionen in der Praxis werden mehr oder weniger davon abweichen und verschiedene Akzente auf einzelne der aufgezeigten Bereiche legen. Es ist gerade die Intention von Mack, hier *kein* schematisches Modell vorzugeben, nach dem dann Software-Expeditionen abzulaufen haben. Vielmehr sollen die Teammitglieder und vor allem der Expeditionsleiter durch das Leitbild der Expedition in die Lage versetzt werden, vor Ort ihren eigenen Ablauf der Software-Expedition situativ selbst zu gestalten.

In dem in Kapitel 2 angesprochenen Spannungsverhältnis zwischen Handlungsanleitung und Flexibilität (vgl. Abbildung 2-5) tendiert die Software-Expedition also zu der rechts dargestellten maximalen Flexibilität. Das Abgleiten ins Chaos wird dabei vor allem durch den Leitbildcharakter der Expedition verhindert. Zusätzlich gibt Mack, seinem Schwer-

punkt zum Management der beteiligten Personen folgend, konkrete Ratschläge vor allem in organisatorischen Aspekten wie iterativer und inkrementeller Entwicklung, kurzer Iterationszyklen, objektorientierter Entwicklung, kleiner Teams, universeller und leichtgewichtiger Ausrüstung etc. Zur technischen Unterstützung bei den täglichen nicht-organisatorischen Aufgaben zur Softwareentwicklung die, wie weiter oben beschrieben, nicht im Fokus von Mack (2001) stehen, werden bewährte softwaretechnische Ansätze kurz beschrieben und für Detailfragen auf diese verwiesen. Nur durch Hinzuziehen und weiterer Betrachtung dieser Ansätze kann diese technische Unterstützung gewährleistet werden, zu der Dinge gehören wie z.B. auf welche Aspekte bei der Interviewführung zur Anforderungsermittlung mit dem Kunden geachtet werden sollte, wie die objektorientierte Analyse, Design und Implementierung angegangen werden kann oder was bei der Entwicklung von Entwicklungs- bzw. Testkonzepten beachtet werden muß.

Wie ich in Kapitel 2 schon angedeutet habe und im Rahmen der Erfahrungen im Praxisprojekt in Kapitel 5 noch ausführen werde, sind dies wesentliche Aspekte beim Erstellen einer flexiblen und praxisorientierten Vorgehensweise – dem Fokus dieser Arbeit.

4.3. Flexibilität in Software-Expeditionen

Im vorigen Abschnitt habe ich gezeigt, daß die Software-Expedition der Flexibilität eine hohe Relevanz zuspricht. Ich werde dies nun anhand der Fragestellungen aus Kapitel 2 genauer untersuchen. Bei allen Anmerkungen in diesem Abschnitt muß allerdings beachtet werden, daß Mack (2001) keineswegs intendiert, mit der Software-Expedition eine vollständige und umfassende Vorgehensweise vorzustellen, sondern lediglich darauf hinzielt, „konkrete Leitideen und Orientierungen für die Vorbereitung und Durchführung von Software-Expeditionen zu entwerfen, wobei sich das Vorgehen einer Software-Expedition hier nur in Grundzügen skizzieren läßt“ (Mack 2001, S. 165).

Die Fragen aus Abschnitt 2.3 lauteten:

(1a) Wie flexibel kann innerhalb der Vorgehensweise mit Veränderungen umgegangen werden?

Da die Software-Expedition von vorne herein von Instabilität der Randbedingungen und damit der Notwendigkeit, mit Veränderungen umzugehen, ausgeht, ist zu erwarten, daß dieser Punkt gut behandelt wird. Tatsächlich wird dies an vielen Stellen sichtbar; beispielsweise wird mit dem Gedanken der situativen Reorientierung während der Entwicklung anstelle von schematischer Planung als Vorbereitung ein Mechanismus zur wieder-

holten Restrukturierung aufgezeigt. Unterstützt wird dies durch die Hervorhebung der beteiligten Personen und ihrer Selbstorganisation, die auf der Basis ihrer individuellen Erfahrungen und nicht als unreflektierte Ausführung eines schematischen Modells entsteht. Auch die Beschränkung auf wenige Dokumente (wenig, universelle und leichtgewichtige Ausrüstung), unterstützt den flexiblen Umgang mit Veränderungen.

(1b) Inwieweit ermöglicht die Vorgehensweise Anpassungen an die spezifischen Bedingungen und die beteiligten Personen zu Projektbeginn?

Die Software-Expedition bietet automatisch eine ganz eigene Art der Anpassung an die beteiligten Personen: Das Team ist so zentral, daß der Ablauf der Entwicklung vollständig von ihm selbst gestaltet wird. Eine Anpassung findet also in hohem Maße durch die Interpretation und Umsetzung der Expeditionssicht im eigenen Vorgehen statt.

Dabei müssen allerdings die Anforderungen an jedes einzelne Teammitglied beachtet werden. Diese werden auf der einen Seite für die generelle Art der Organisation und Zusammenarbeit mit anderen Personen als primärem Gegenstand der Software-Expedition verringert. Die Unterstützung bei der technischen Arbeit muß auf der anderen Seite aber in zusätzlichen Ansätzen gesucht werden. Die Software-Expedition selbst bietet nur eine Übertragung der Expeditionssicht auf die aktuelle Aktivität und die spezifischen Bedingungen an, auf deren Basis zusammen mit den eigenen Erfahrungen die notwendigen Entscheidungen getroffen werden müssen. Es ist denkbar, daß sich gerade Teammitglieder mit weniger Erfahrung oder Erfahrung in anderen Bereichen[85] hiervon überfordert fühlen. Diese verlangen nach klaren Empfehlungen für ihre konkreten technischen Aufgaben.

(2) Welche praktische Unterstützung liefert die Vorgehensweise für die täglichen Aufgaben im Projekt?

Zunächst einmal läßt sich feststellen, daß die Software-Expedition aus Beobachtungen der Praxis hervorgegangen ist und allein dadurch schon einen Praxisbezug vorweisen kann. Auf der anderen Seite bietet aber das Heranziehen der Expeditionsmetapher in vielen Bereichen die einzige Unterstützung für die täglichen Aufgaben, die sich – wie Mack vorschlägt – durch das Hinzuziehen anderer softwaretechnischer Ansätze ergänzen läßt. Für die vorliegende Arbeit ist aber gerade diese Unterstützung alltäglicher Praxisaufgaben von

[85] Dies betrifft z.B. in der Praxis zur Zeit sehr häufig anzutreffende Entwickler, die auf objektorientierte Technologien umgeschult werden müssen, sowie die sogenannten „Quereinsteiger“, die nicht auf eine langjährige und fundierte Ausbildung für ihre Tätigkeit zurückgreifen können.

zentraler Bedeutung[86]. Die kurze Beschreibung von drei verbreiteten softwaretechnischen Ansätzen und ihres Bezugs zur Software-Expedition gibt dabei für eine konkrete Vorgehensweise für die Praxis im Sinne der vorliegenden Arbeit (vgl. Kapitel 2) zu wenig genaue Informationen. Auch fehlt – wie Mack selbst anmerkt[87] – bisher eine Erprobung der Software-Expedition sowie praktische Erfahrungen mit der Verbindung der beschriebenen softwaretechnischen Ansätze.

Die Beantwortung dieser Fragen hat gezeigt, daß auch die Software-Expedition dem Spannungsverhältnis zwischen Handlungsanleitung bzw. praktischer Unterstützung und Flexibilität[88] unterworfen ist. Wie in Abbildung 4-5 noch einmal gezeigt, tendiert die Software-Expedition hier deutlich zur Seite der Flexibilität, auch wenn durch die Ergänzung anderer Ansätze zusätzliche handlungsanleitende Elemente hinzukommen können.

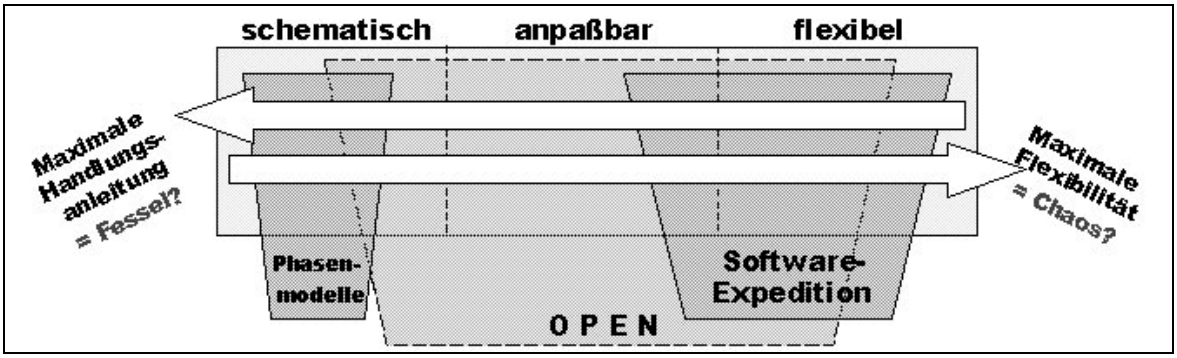

Abbildung 4-5 – Die Software-Expedition zwischen Schema und Flexibilität

Nachdem OPEN in Kapitel 3 allgemein vorgestellt wurde, werde ich im folgenden Kapitel Ideen für eine eigene OPEN-Instanz vorstellen. In diese OPEN-Instanz, sollen sowohl zentrale Gedanken und Eigenschaften als auch Überlegungen zu den Schwachstellen der Software-Expedition einfließen:

- ***Personen-orientiertes Management***
  Besonders wertvoll erscheint mir an der Software-Expedition der Teamaspekt. Personen (Entwickler, Entscheider und Anwender) werden nicht mehr einfach nur als zu

[86] Dies liegt wieder am generellen Fokus der beiden Arbeiten. Mack sieht zwar auch die große Bedeutung de softwaretechnischen Unterstützung (vgl. Mack 2001, S. 200ff), verweist dazu aber auf andere Autoren und beschäftigt sich in seinen Arbeiten vor allem mit der Organisation der beteiligten Menschen.

[87] Vgl. Mack (2001, S. 225).

[88] Vgl. Kapitel 2.

verplanende Ressource eingesetzt, sondern als aktive, das Vorgehen beeinflussende und gestaltende Individuen angesehen.

- ***Erfahrungsbasierte Flexibilität***
  Da die Software-Expedition auf die Erfahrungen der beteiligten Personen baut, ist es diesen möglich, sich immer wieder situativ neu zu orientieren. Dies ist in heutigen Entwicklungsvorhaben unabdingbar. Es sollte allerdings auch beachtet werden, daß die Kenntnisse unerfahrenerer Teammitglieder nicht überschätzt werden und daß nicht *zu* viel von diesen Erfahrungen abhängt.

- ***Planung als Ressource***
  Die Software-Expedition behandelt Planung nicht als zwingende Vorschrift, sondern als Ressource, die für das gesamte Team transparent gehalten wird. Dies ermöglicht weitere Flexibilität im Bezug auf Veränderungen der Rahmenbedingungen.

- ***Unterstützung für die Praxis***
  Wie weiter oben gezeigt, bietet die Software-Expedition in der vorliegenden Form noch zu wenig konkrete Unterstützung für technische Aufgaben der Praxis. Hier werden zusätzliche Konzepte (die ja in der Software-Expedition auch erwähnt sind) benötigt, um eine umsetzbare Vorgehensweise zu beschreiben.

5. Ein flexibler Einsatz von OPEN in der Praxis

In diesem Kapitel soll der Name »OPEN« wörtlich genommen werden: Die in OPEN beschriebenen Aspekte und Hilfestellungen werden also nicht als erschöpfend angesehen; vielmehr bildet OPEN eine *offene* Basis, die durch andere Ansätze und eigene Gedanken erweitert werden kann und soll. Generell geht es mir darum, Grundgedanken für eine flexible, aus der Praxis motivierte OPEN-Instanz darzustellen, die den beteiligten Personen konkrete Unterstützung im Projekt gibt. Diese Gedanken gehen dabei auf Diskussionen zu Ansätzen aus folgenden Quellen zurück:

- Grundsätzliche Überlegungen zur Flexibilität (vgl. Kapitel 2),
- OPEN (vgl. Kapitel 3),
- Softwareentwicklung als Expedition (vgl. Kapitel 4),
- Ideen aus anderen Ansätzen (vor allem aus XP, RUP, STEPS, WAM, FDD und dem V-Modell 97)[89]
- informelle Gespräche mit Managern, Beratern, Entwicklern und Studenten[90],
- allgemeine Literatur zur Softwaretechnik (auf die genauen Quellen werde ich an den entsprechenden Stellen hinweisen) und
- Erfahrungen, die sich mit dem Einsatz im begleitenden Projekt ergeben haben und Anpassungen noch im laufenden Projekt notwendig machten[91].

Für die vorliegende Arbeit stand dabei die Aufnahme, Verarbeitung und Verbindung von Aspekten und Informationen aus diesen Quellen im Vordergrund und nicht die Ausarbeitung einer vollständigen Vorgehensweise oder OPEN-Instanz. Ich werde nach einem Überblick über die Situation im begleitenden Projekt die resultierenden Ideen zu einer möglichen Unterstützung durch eine Vorgehensweise vorstellen und anschließend auf die Erfahrungen aus der Praxis noch einmal gesondert eingehen.

---

[89] Vgl. Beck (1999) zu Extreme Programming (XP), Jacobson (1999) oder Kruchten (1999) zum Raional Unified Process (RUP), Floyd et al. (1989) zur Software-Technik für Evolutionäre Partizipative Systementwicklung (STEPS), Züllighoven (1998) zum Werkzeug & Material-Ansatz (WAM), Coad et al. (1999) zum Feature Driven Development (FDD) und Dröschel (1998) zum V-Modell 97.

[90] Informelle Gespräche zum Thema Vorgehen in der Softwareentwicklung konnte ich vor allem im Rahmen meiner Programmier- und Beratungstätigkeit in zwei verschiedenen Firmen (Higher Order GmbH – jetzt Coremedia AG und CTH Consult Team Hamburg GmbH) sowie in Kundengesprächen innerhalb meiner Arbeit bei diesen Firmen in den letzten Jahren führen.

[91] Vgl. auch Abschnitte 1.4 und 5.3 sowie den (z.T. vertraulichen) Projektreport CTH (2001).

5.1. Besondere Einflußfaktoren in der Praxis

Bevor ich meinen Ansatz zur OPEN-Instanz vorstelle, möchte ich zunächst auf einige bekannte, aber oft unzureichend adressierte Besonderheiten der Praxis hinweisen. Diese besonderen Einflußfaktoren sind in der Erfahrung mit dem die Arbeit begleitenden Projekt und im Austausch mit den beteiligten Personen deutlich geworden und betreffen beispielsweise die Art der Abrechnung, politische Entscheidungen, die Forderung nach Vorstudien sowie die konkrete Unterstützung der Entwickler.

Art der Abrechnung

Ein Punkt, der eine große Auswirkung auf das Vorgehen hat, in vielen Vorgehensweisen aber übersehen wird, ist die Art der Abrechnung. In der Praxis werden infolge der weiten Verbreitung von Phasenmodellen noch immer Projekte nach der Berechnung eines Festpreises durchgeführt. So wird vom Auftraggeber erwartet, daß die Entwickler nach einer gewissen Zeit (oder sogar schon zu Projektstart) einen so guten Überblick über das zu erstellende System haben, daß sie akkurat abschätzen können, was die Entwicklung kosten wird und wie lange sie dauert. Zu diesem Zeitpunkt wird dann ein Vertrag aufgesetzt, an dem bis Projektende nach aller Möglichkeit festgehalten werden soll.

Obwohl dies zunächst ein legitimes Vorgehen zu sein scheint, zeigen sich gerade hier die Nachteile der Phasenmodelle besonders deutlich: Heutige Projekte (im Bereich großer, dedizierter und objektorientierter Anwendungssoftware) können nicht mehr bis zum Ende durchgeplant und abgeschätzt werden[92]. Zu groß ist der Einfluß der ständigen Veränderung. Im Endeffekt haben also bei dieser Art der Abrechnung beide Parteien das Nachsehen: Die Entwicklungsorganisation, weil der Aufwand in den allermeisten Fällen unterschätzt wird[93] und die versprochene Funktionalität im vertraglich festgeschriebenen Rahmen nicht oder nur mit zusätzlicher Eigeninvestition geliefert werden kann und die Anwendungsorganisation, weil die logische Konsequenz meist eine Verschlechterung der Qualität oder eine Einschränkung der im Vertrag nicht genau detaillierten Funktionalität ist. Dazu kommt häufig ein Streit über die Vertragslage, der weder dem Verhältnis der Organisationen untereinander noch der Qualität oder Wartung der Software dienlich ist.

---

[92] Vgl. Kapitel 2.

[93] Vgl. Weltz und Ortmann (1992) sowie Standish Group (1995).

Eine Alternative, die nur teilweise und sehr zögerlich von den Großkunden angenommen wird, ist die Abrechnung nach Aufwand. Hier rechnen die Entwickler die tatsächlich eingesetzte Zeit stundenweise ab. Problematisch kann sich hier auswirken, daß die Interessen von Entwicklungsorganisation (viele Entwickler lange beschäftigen bedeutet großen Umsatz) und Anwendungsorganisation (wenige Entwickler für kurze Zeit beauftragen bedeutet geringe Kosten) entgegengesetzt verlaufen. Als Kompromiß beschreibt Beck (2000, S. 159ff) einerseits einen Bonus zur Fertigstellung bestimmter Teile (»completion bonus«) und andererseits eine Abschwächung des Festpreismodells in eine Art Abonnement. Dabei werden zwar ebenfalls Festpreise vereinbart, aber nur für die jeweils nächste Iteration und im Zusammenhang mit der Planung der in dieser Iteration zu entwickelnden Funktionalität. So haben beide Seiten zu Beginn jeder Iteration die Möglichkeit, auch in den Bezahlungsmodus aktiv einzugreifen.

Politische Entscheidungen

Die Entscheidung über die Art der Abrechnung bekommt in der Praxis sehr schnell eine politische und projektübergreifende Bedeutung. Aus wirtschaftlichen Gesichtspunkten ist es in Projekten oft notwendig, auf unangenehme Forderungen nach einem Festpreis o.ä. einzugehen. Die Ungenauigkeit der Schätzung sowie erhöhte Risiken müssen dabei dann in Kauf genommen werden. Hier kann sich die Flexibilität dadurch auszahlen, daß es möglich ist, solche Kompromisse auszuführen, ohne die Unterstützung der Vorgehensweise zu verlieren.

Forderung nach Vorstudien

Eine weitere, häufig anzutreffende Forderung bei der Planung größerer Projekte, ist die Durchführung einer mehrwöchigen Vorstudie. Der Kunde erwartet, daß sich die Teammitglieder der Entwicklungsorganisation innerhalb dieser Vorstudie einen Eindruck über die Projektsituation verschaffen und so in der Lage sind einzuschätzen, inwieweit eine Realisierung im angedachten Zeit- und Kostenrahmen möglich ist. Am Ende der Vorstudie steht dann typischerweise kein entwickeltes (Teil-)System, sondern ein umfangreiches Dokument zur Beschreibung der Situation. Dabei kann es sich um eine allgemeine Einschätzung der Lage und eine erste Analyse der bereits aufgedeckten Risiken und Chancen handeln.

Eine ausführliche Spezifikation in einem klassischen Pflichtenheft[94], wie es typischerweise in Phasenmodellen gefordert wird, sollte aus den in den vorangegangenen Kapiteln dargelegten Gründen unter allen Umständen vermieden werden.

Konkrete Unterstützung der Entwickler

Im Gespräch mit Entwicklern über das Vorgehen kommt immer wieder die Frage nach den einzusetzenden Softwarewerkzeugen und weiterer konkreter Unterstützung bei der täglichen Arbeit auf. Hier muß darauf geachtet werden, daß die ausgewählten Programme die Vorgehensweise unterstützen und ihre Umsetzung nicht erschweren. Auch wenn die Werkzeuge selbst prinzipiell austauschbar sind, so sollte doch berücksichtigt werden, daß jedes Werkzeug eine Art der Benutzung nahe legt, und daß das Einarbeiten in neue Werkzeuge zusätzlichen Aufwand verursacht. So muß ein Kompromiß aus vielseitigen, gleichzeitig aber intuitiv benutzbaren Werkzeugen[95] gefunden werden.

5.2. Ein Ansatz für eine passende OPEN-Instanz

Ich werde nun einen Ansatz dazu vorstellen, wie die Bearbeitung dieser Aufgaben durch eine angemessene OPEN-Instanz unterstützt werden kann. Dabei geht es mir auch vor dem Hintergrund des begleitenden Projektes vor allem um die in Abschnitt 2.3 ausgearbeiteten Kriterien zur Flexibilität und zur Umsetzbarkeit der Vorschläge in der Praxis.

5.2.1. Anforderungen / Herleitung der Prinzipien

Die bisherige Diskussion in der vorliegenden Arbeit sowie Erfahrungen aus der Praxis haben gezeigt, daß die hier gesuchte flexible Vorgehensweise zur Entwicklung von großer, dedizierter und objektorientierter Anwendungssoftware an die spezifische Projektsituationen anpaßbar sein, Veränderlichkeit von Anforderungen und Rahmenbedingungen berücksichtigen, Unterstützung bei der täglichen Arbeit liefern sowie Erfahrungen der beteiligten

---

[94] Zu Bedeutung und Aufbau eines Pflichtenheftes vgl. z.B. Balzert (1996, S. 104 ff) sowie folgende Beschreibung: „Die Ergebnisse der gesamten Problemanalyse werden in einem Dokument, der sog. *Anforderungsdefinition* auch *Pflichtenheft* genannt, festgehalten. Sie bildet ein verbindliches Dokument zwischen Auftraggeber und Softwareproduzent und ist Grundlage für den abzuschließenden Vertrag nach dessen Unterzeichnung sie nur im gegenseitigen Einverständnis geändert werden darf.“ (Duden Informatik 1993, S. 659)

[95] Vgl. auch die Aspekte zur Ausrüstung von Software-Expeditionen in Kapitel 3.

Personen einbeziehen muß[96]. Diese generellen Erkenntnisse sollen in der zu beschreibenden OPEN-Instanz durch konkrete Prinzipien berücksichtigt werden. Dazu werde ich im Folgenden auf jede dieser Erkenntnisse kurz eingehen, dabei die Kernpunkte für die zu beschreibende Vorgehensweise erarbeiten und diese abschließend noch einmal zu sechs Prinzipien zusammenfassen.

Unterstützung spezifischer Projektsituationen

In Kapitel 3 habe ich gezeigt, daß OPEN gute Hilfestellungen bereitstellt, die spezifische Projektsituation zu unterstützen. Aus vielfältigen Beschreibungen zu Aktivitäten, Aufgaben und Techniken kann eine projektspezifische Auswahl getroffen werden, die zur Unterstützung der beteiligten Personen bereitsteht. Mehrere praxiserprobte Beispiele in der Literatur zu OPEN (vgl. Kapitel 3) geben zusätzliche Orientierung für diesen Anpassungsprozeß. Für die hier zu beschreibende OPEN-Instanz möchte ich zusätzlich den Begriff »Repertoire« als eine projekt-, team- und erfahrungsabhängige Menge von sich ergänzenden und die wesentlichen Bereiche abdeckenden Aufgaben oder Techniken im Sinne von OPEN verwenden. Der Prozeß der Zusammenstellung einer eigenen Vorgehensweise für ein konkretes Projekt umfaßt dann folgende Schritte:

1. Aufstellen grundlegender Prinzipien[97]
2. Auswählen und Anpassen der Aktivitäten[97]
3. Strukturieren der Aktivitäten in einem Lebenszyklusmodell[97]
4. Zusammenstellen eines Grundrepertoires (auf Organisations-, Abteilungs- oder Projektebene)

Die *Identifikation grundlegender Prinzipien* führe ich in diesem Abschnitt vor. In der Praxis entstehen diese meist aus den vielfältigen Erfahrungen vergangener Projekte. Wichtig ist aber, daß diese Prinzipien klar artikuliert und für alle Beteiligten transparent gemacht werden. Uneinigkeit über die grundlegenden Prinzipien kann sonst dazu führen, daß einzelne Teammitglieder gegeneinander arbeiten. Die Notwendigkeit für klare Prinzipien be-

---

[96] Vgl. die Ausführungen in Kapitel 2 bis 4.

[97] Besonders für kleinere Projekte empfiehlt es sich, auf die vorhandenen Beispiele aufzusetzen (oder diese sogar unverändert zu lassen), um in diesen Anpassungsprozeß nicht zu viel Zeit zu investieren. Bei größeren Projekten wird sich auch größerer Anpassungsaufwand aber schnell durch eine passende Vorgehensweise rentieren.

deutet aber nicht, daß diese für alle Zeiten feststehende Gesetze sind. Veränderte Rahmenbedingungen (s.u.) können u.U. dazu führen, daß auch Änderungen der Prinzipien notwendig werden. Hier hat der Projektleiter die schwierige Aufgabe abzuwägen, welche Diskussionen um neue Prinzipien das Projekt vorantreiben und welche es behindern können.

Auf die *Auswahl der Aktivitäten* und ihre *Zusammenstellung zu einem Lebenszyklusmodell* bin ich in Kapitel 3 bereits eingegangen und habe verschiedene Alternativen der OPEN-Literatur vorgestellt. Hier soll nur noch einmal darauf hingewiesen werden, daß auch die Beschreibungen der Aktivitäten aus Graham et al. (1997) an die Gegebenheiten innerhalb der Organisation angepaßt und ggf. durch eigene Gedanken ergänzt werden sollten. Im folgenden Abschnitt 5.2.3 werde ich das für das begleitende Praxisprojekt erarbeitete Lebenszyklusmodell und die darin verwendeten Aktivitäten vorstellen.

Um das Team im Projekt zu entlasten, sollten zentrale Teammitglieder (je nach Projekt- und Teamgröße der Projektleiter zusammen mit einigen oder allen Teammitgliedern) schon zu Projektbeginn ein *Grundrepertoire aus den in OPEN beschriebenen Aufgaben und Techniken*, das der Aufgabenstellung angemessen erscheint, zusammenstellen[98]. Dabei kann es sich z.B. um so grundlegende softwaretechnische Aufgaben wie die Identifikation von Klassen oder die Modellierung von Anwendungsfällen mit UML-Diagrammen handeln. Als Bestandteil des Grundrepertoires an Techniken sind z.B. das Führen von Interviews, Abstraktionstechniken oder das Verwenden zentraler Softwarewerkzeuge vorstellbar. Diese Grundrepertoires werden dann von jedem Teammitglied um weitere Aufgaben und Techniken ergänzt, die für spezielle Arbeiten hilfreich sind.

Wichtig ist, daß vor allem die Zusammenstellung der Repertoires kein vollständig formaler und genau spezifizierter Prozeß ist, weil sonst die Kreativität und Freiheit der beteiligten Personen zu stark eingeschränkt wird und auf neuartige (bisher nicht berücksichtigte) Situationen nicht angemessen reagiert werden kann. So ist das Hinzuziehen anderer (OPEN-fremder) Quellen für Beschreibungen durchaus erwünscht und die Anpassung sämtlicher Beschreibungen vor dem eigenen Erfahrungshintergrund erforderlich. Hier zeigt sich die Erweiterbarkeit von OPEN, auf die ich bereits in Kapitel 3 hingewiesen habe.

---

[98] Zum in Kapitel 4 angesprochenen Problem der Darstellung dieser Zusammenstellung vgl. Abschnitt 5.2.3.

Berücksichtigung veränderlicher Anforderungen und Rahmenbedingungen

Den flexiblen Umgang mit Veränderungen der Anforderungen und Rahmenbedingungen im Projektverlauf habe ich bereits im vorigen Kapitel als einen Schwachpunkt der OPEN Standardbeispiele identifiziert. OPEN liefert zwar einige klassische Ansätze, die sich zur Berücksichtigung von Veränderungen im laufenden Projekt nutzen lassen, das Problem wird aber nicht direkt adressiert. Vor allem drei bewährte softwaretechnische Grundideen werden auch in OPEN angesprochen, aber in den verfügbaren Instanzen nicht deutlich genug herausgestellt:

1. Iterative Entwicklung
2. Inkrementelle Auslieferung
3. Kontinuierliche Qualitätssicherung

Gerade auch das Anwenden der Vorgehensweise in der Praxis hat gezeigt, daß eine konsequent *iterative Entwicklung* häufiger Diskussion bedarf und in der Kooperation mit dem Kunden einige Probleme aufwirft. Auf diese Probleme und die gewählten Lösungsansätze werde ich in Abschnitt 5.3 noch näher eingehen. Grundsätzlich ist iterative Entwicklung mit dazugehörigen regelmäßigen Reviews des Kunden und der Anwender aber die einzige Möglichkeit, fachliche Anforderungen des Kunden, die sich im Laufe des Projektes verändern, angemessen umzusetzen. Die dazu notwendige umfangreiche Kommunikation kann z.B. durch die Entwicklung beim Kunden vor Ort extrem vereinfacht werden. Noch günstiger wirkt sich die Integration eines Mitarbeiters der Kundenorganisation in das Entwicklungsteam aus. So steht immer jemand als fachlicher Ansprechpartner zur Verfügung, der auch noch als Vermittler bei Unstimmigkeiten zwischen Kunden und Entwicklern dienen kann, da es ihm leichter fallen wird, die Problematik von beiden Seiten zu betrachten.

Gegenstand der Reviews sollten nach Möglichkeit lauffähige Teile des Systems oder grundlegende fachliche sowie ggf. auch technische Entscheidungen sein. So wird das System also *in Inkrementen* weiterentwickelt, die immer wieder als Diskussionsgrundlage für die weitere Entwicklung herangezogen werden. Nur durch inkrementelle Auslieferung zu Testzwecken und wenn möglich inkrementelle Integration des Systems in die Arbeitsabläufe können neue Vorstellungen des Kunden in die weitere Systementwicklung mit vertretbarem Aufwand einfließen.

> „Viele Organisationen machen die Erfahrung, daß sich erst bei der Einführung von Systemen klärt, wie qualifizierte Arbeitstätigkeit sinnvoll unterstützt werden kann.“ (Floyd 1994b, S. 37)

Schließlich weist auch OPEN darauf hin, daß *Qualitätssicherung* nicht lediglich aus einer abschließenden Testphase kurz vor Abnahme des Systems bestehen darf. Beim Testen sollte es sich vielmehr um einen kontinuierlichen und wenn möglich durch geeignete Software teilautomatisierten Prozeß handeln der auf sämtliche Entwicklungsaufgaben betrifft. Das Aufstellen und Einhalten von Entwicklungsrichtlinien, sinnvolle Dokumentation des Quellcodes und Dokumente zum Überblick und zur Handhabung des Systems sind hier weitere wichtige Aspekte.

Zusätzlich zu diesen drei softwaretechnischen Grundideen hat die Betrachtung der Software-Expedition in Kapitel 4 gezeigt, daß für eine hohe Flexibilität die *aktive und kreative Selbstorganisation der Teammitglieder mit Einbeziehung der Lernerfahrungen* anstelle der passiven Befolgung eines Schemas im Mittelpunkt stehen muß. Auch wenn OPEN mit dem Lebenszyklusmodell eine Art grobes Schema bereitstellt, so soll dieser Punkt trotzdem in der Vorgehensweise berücksichtigt werden. Ohne dem Abschnitt 5.2.3 zu sehr vorzugreifen, soll hier schon auf wesentliche Eigenschaften hingewiesen werden, die eine solche aktive und kreative Selbstorganisation auch bei Verwendung des Lebenszyklusmodells ermöglichen: Das Lebenszyklusmodell (das bei größeren Projekten sogar von den Teammitgliedern selbst erstellt wird und somit kein von außen vorgegebenes Schema ist) stellt lediglich eine grobe Übersicht und Struktur des Projektes dar (die Aktivitäten). Es liegt bei den Teammitgliedern, diese groben Aktivitäten durch die Zuordnung von Aufgaben unter Einbeziehung ihrer Erfahrungen zu füllen. Außerdem wurde in die OPEN-Instanz eine neue Aktivität »Anpassung des Vorgehens« zur Berücksichtigung von Veränderungen aufgenommen (vgl. Abschnitt 5.2.4).

Zwei weitere Aspekte, möchte ich in diesem Rahmen von der Software-Expedition aufgreifen. Zum einen soll auch in dieser Vorgehensweise *Planung als Ressource* aufgefaßt werden. Auch die Planung selbst ist ständiger Veränderung und unvorhersehbaren Ereignissen ausgesetzt und sollte somit nicht als festgeschriebene Regel verstanden werden. Um die Planung als Ressource und auch als Motivationsinstrument nutzen zu können, bietet es sich an, sie für alle Teammitglieder transparent zu machen. Zum anderen ist die Beschränkung auf *wenige, zentrale Dokumente* ein wichtiges Mittel, um flexible und situative Entscheidungen zu ermöglichen. Existieren zu viele Dokumente mit hohem Abhängigkeits-

grad, so zieht eine unvorhersehbare Entscheidung automatisch das Aktualisieren zahlreicher Dokumente nach sich.

Unterstützung für die tägliche Arbeit

Während es für die Flexibilität essentiell ist, sich auf wenige Dokumente zu beschränken, so fordert die Praxis aber Unterstützung bei der täglichen Arbeit auch in Form von Vorlagen für Dokumente, die auf Kundenwunsch, als Überblick, zur Strukturierung der Gedanken oder als Kommunikationsgrundlage eben doch immer wieder benötigt werden. Während beispielsweise Beck (2000, S.111 f) empfiehlt Grafiken und andere Überichtsdokumente bei Bedarf zu erstellen und nicht aufzubewahren, so halte ich dies in vielen Fällen für wenig sinnvoll. Zentrale Fragen werden immer wieder, auch in Diskussionen mit verschiedenen Personen, auftreten; hier können die einmal entworfenen Zeichnungen oder Kurzbeschreibungen einen guten Einstieg in die Thematik vermitteln. Darüber hinaus haben sich einige Dokumenttypen (Anwendungsfalldiagramme, Grafiken zum Schichtenmodell, fachliche und technische Glossare, Entwicklungsrichtlinien, Gesprächsprotokolle etc.) so sehr in der Projektpraxis etabliert und bewährt, daß es nicht sinnvoll scheint, diese zu verbannen. Ich werde daher empfehlen, in einer Vorgehensweise vor allem die wenigen und zentralen Dokumente durch Bereitstellen von Vorlagen zu unterstützen, die erfahrungsgemäß ohnehin erstellt werden.

Als weitere Unterstützung der Entwickler bei ihrer täglichen Arbeit sollen in der Vorgehensweise die OPEN-Beschreibungen von Aufgaben und Techniken benutzt werden. Vor allem unerfahrene Entwickler haben so die Möglichkeit sich schnell in die neuen Aufgaben einzuarbeiten und werden auf typische Fehlerquellen sowie bewährte Methoden aufmerksam gemacht.

Einbeziehen von Erfahrungen der beteiligten Personen

Ein weiterer Aspekt, der in OPEN zu kurz kommt, in der Software-Expedition aber zentral behandelt wird, ist das Einbeziehen von Erfahrungen der beteiligten Personen. Wie oben angedeutet, werden die Erfahrungen schon durch die aktive Rolle der Teammitglieder bei der Ausgestaltung des Prozeßmodells in das Vorgehen einfließen. Lernerfahrungen auf Kunden- und Entwicklerseite werden auch als wesentlich für die Entwicklung eines angemessenen Systems angesehen und stellen ein zusätzliches Projektergebnis dar.

> „Die in einem Projekt gemachten Erfahrungen sind ein wichtiges Projektergebnis, welches für die Gestaltung zukünftiger Projekte von hohem Wert ist.“(vgl. Mack 2001, S. 160)

Gleichzeitig wird den Entwicklern aber, wie oben beschrieben, Unterstützung in Form von Beschreibungen zu Aufgaben und Techniken sowie Vorlagen angeboten, so daß die individuellen Erfahrungen um bewährte Konzepte angereichert werden können.

5.2.2. Grundlegende Prinzipien

Zusammenfassend läßt sich auch in der Argumentation des vorigen Abschnitts das schon in Kapitel 2 angesprochene Spannungsverhältnis zwischen größtmöglicher Flexibilität und präskriptiver konkreter Handlungsanleitung erkennen. Auf der einen Seite steht die unabhängige, aktive Selbstorganisation der Teammitglieder mit Einbeziehung ihrer Lernerfahrungen, während auf der anderen Seite konkrete Unterstützung der Teammitglieder gerade auch von den Entwicklern selbst gefordert wird. Die OPEN-Instanz, für die ich in diesem Kapitel die Grundlagen schaffen will, soll das Spannungsverhältnis dahingehend lösen, daß ein Lebenszyklusmodell sowie Aufgabenbeschreibungen als konkrete Unterstützung zur Verfügung stehen, diese aber den Lernerfahrungen und situativen Entscheidungen der beteiligten Personen untergeordnet sind. Die folgenden Prinzipien haben sich als Konsequenz aus den im vorigen Abschnitt beschriebenen Anforderungen ergeben und werden die Grundlage für die OPEN-Instanz sein:

- **Beteiligte Personen (ihre Kommunikation und ihre Erfahrungen) stehen im Mittelpunkt;**
- **Qualitativ hochwertige Software durch iterative Entwicklung, inkrementelle Auslieferung und kontinuierliches Testen sowie Einbeziehen von Lernerfahrungen durch häufige Reviews;**
- **Ineinandergreifen von Analyse, Design, Implementierung und Testen statt Trennung in sequentielle Phasen;**
- **Erfahrungsbasierte Aktivitäten und Aufgaben statt feste Dokumente als treibendes Moment – Bereitstellen von Vorlagen für nur wenige zentrale Dokumente;**
- **Flexibilität in Planung, Projektstruktur und Vorgehen;**
- **Realisierung der Prinzipien in einem übersichtlichen Lebenszyklusmodell mit strukturierenden Aktivitäten.**

Auf die Abbildung der Prinzipien in einem Lebenszyklusmodell aus Aktivitäten werde ich im folgenden Abschnitt genauer eingehen.

5.2.3. Das Lebenszyklusmodell

Abbildung 5-1 zeigt die Umsetzung dieser Prinzipien in Aktivitäten, die in einem Lebenszyklusmodell organisiert sind. Wesentliche Teile dieses Lebenszyklusmodells waren bereits zu Beginn des begleitenden Praxisprojekts vorhanden und wurden eingesetzt. Einige Ergänzungen haben sich aber erst im Laufe des Projekts im Rahmen der Aktivität »Anpassung des Vorgehens« entwickelt (vgl. auch den weiter unten gezeigten frühen Entwurf).

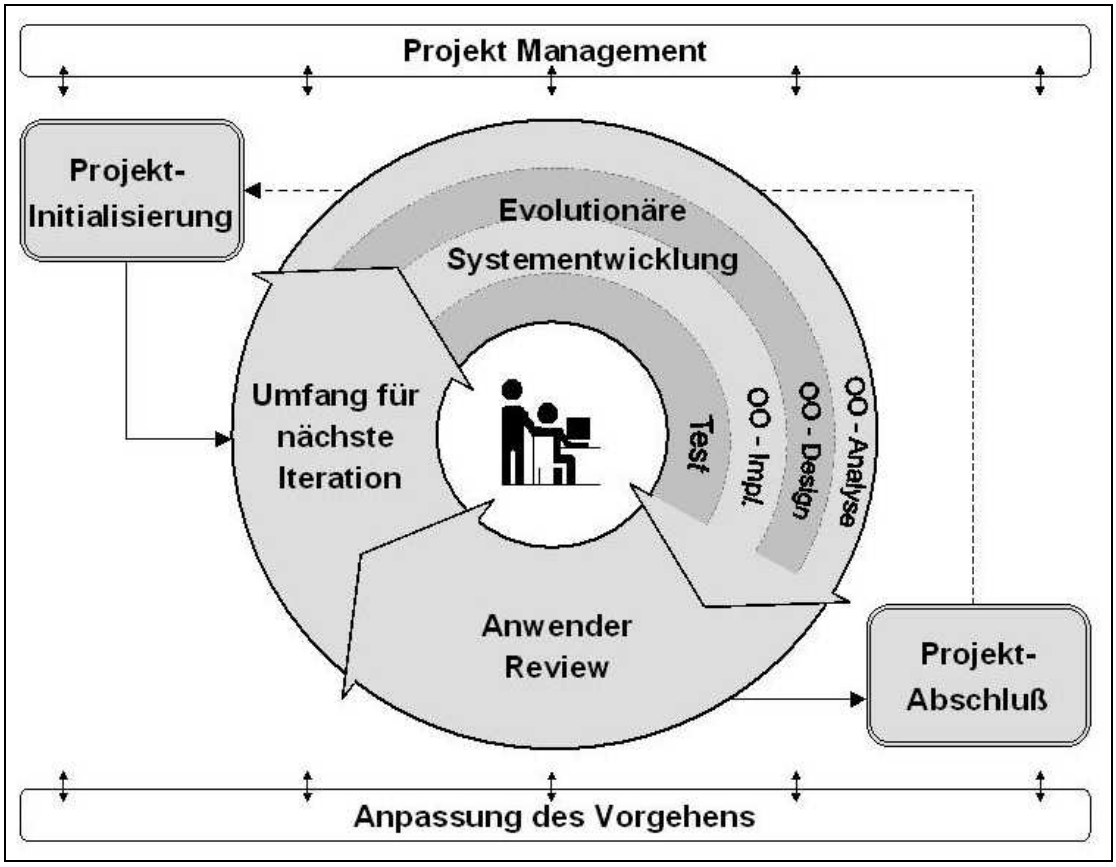

Abbildung 5-1 – Übersicht über das Lebenszyklusmodell

Bei der Zusammenstellung des Lebenszyklusmodells wurde darauf geachtet, daß einerseits die im vorigen Abschnitt identifizierten Prinzipien zusammen mit den für die Durchführung von Softwareentwicklungsprojekten wichtigen Aktivitäten abgebildet wurden; andererseits sollte aber die Übersichtlichkeit erhalten bleiben, damit das Lebenszyklusmodell zur Darstellung der Vorgehensweise nach außen sowie zur projektinternen Orientierung genutzt werden kann. Dabei haben Erfahrungen mit ersten Entwürfen des Lebenszyklusmodells gezeigt, daß viele der prinzipiellen Aspekte in Darstellungen wie dem »contract-driven lifecycle model« (CDLM)[99] aus OPEN nicht klar werden. Wie in Kapitel 3 angemerkt, wurde darüber hinaus die Bedeutung der die Aktivitäten verbindenden Pfeile nicht deutlich. Im hier vorliegenden Lebenszyklusmodell ist dies eindeutig: Die Pfeile im Zentrum der Grafik, die zur Verbindung der grau unterlegten Kernaktivitäten dienen, beziehen sich auf die *zeitliche* Abfolge der Aktivitäten. Diese Bedeutung erschließt sich dem Betrachter auf Anhieb auch schon durch die Anordnung der Aktivitäten, die zusammen mit den Pfeilen weitgehend der üblichen Lesereihenfolge von links oben nach rechts unten folgen. Leichte Überlappungen der Aktivitäten (wie beispielsweise weiterführendes Testen

[99] Vgl. Kapitel 4.

während des »Anwender Reviews«) sind möglich, aber nicht grundsätzlich intendiert. Zusätzlich zu den Kernaktivitäten befinden sich am oberen und unteren Rand projektbegleitende Aktivitäten, die von »Projektinitialisierung« bis »Projektabschluß« (und evtl. mit der »Wartung / Erweiterung« sogar darüber hinaus) ausgeführt werden müssen.

Das *erste Prinzip*, das die zentrale Stellung der beteiligten Personen, ihrer Kommunikation und Erfahrungen fordert, ist in der Grafik an mehreren Stellen visualisiert: Zum einen weist die Grafik im absoluten Zentrum eine schematische Darstellung von Menschen auf. Dies soll andeuten, daß es die beteiligten Personen sind, die das gesamte Prozeßmodell anpassen, umsetzen, mit konkreten Inhalten füllen und so der statischen Grafik zu einer flexiblen Dynamik im Projektalltag verhelfen.

Der auffälligste Teil der Grafik, der große zyklische Mehrfachpfeil in der Mitte, beinhaltet die Aktivitäten der eigentlichen Entwicklung des Systems. Hier bekommt die Kommunikation vor allem dadurch ein starkes Gewicht, daß zwei der drei Aktivitäten – Umfang für nächste Iteration und »Anwender Review« – durch Austausch und Diskussion von Entwicklern, Kunden und Anwendern geprägt sind.

Zusätzlich – dies konnte in der Grafik nicht dargestellt werden, ohne die Übersichtlichkeit zu stark einzuschränken – sind die Erfahrungen der Teammitglieder bei der Ausgestaltung der Aktivitäten gefordert, wenn es darum geht die relevanten Repertoires aus Aufgaben und Techniken zusammenzustellen.

Das *zweite Prinzip* der Qualitätssicherung zeigt sich ebenfalls in den zentralen Entwicklungsaktivitäten: Die zyklische Anordnung stellt dar, daß es sich bei der Abfolge der Aktivitäten um einen iterativen Prozeß handelt. Nach einem Durchlauf der drei Aktivitäten »Umfang für nächste Iteration«, »Evolutionäre Systementwicklung« und »Anwender Review« ist eine Iteration beendet und es folgt ein weiterer Durchlauf mit Beginn der Aktivität Umfang für nächste Iteration. Vor jedem »Anwender Review«, in dem Lernerfahrungen aus dem bisherigen Projektverlauf aufgearbeitet und besprochen werden, muß ein Inkrement des zukünftigen Systems als Ergebnis der Iteration an den Kunden ausgeliefert werden, damit für das »Anwender Review« eine Diskussionsgrundlage geschaffen ist.

In der »Evolutionären Systementwicklung« ist das Testen als eine Unteraktivität gezeigt, die parallel zu Analyse, Design und Implementation von Beginn an durchgeführt wird. So wird einerseits der kontinuierlichen Qualitätssicherung und gleichzeitig dem *dritten Prin-*

zip vom Ineinandergreifen von Analyse, Design und Implementierung Rechnung getragen. Gerade mit der Unterstützung von Objektorientierung und geeigneten Softwarewerkzeugen läßt sich leicht zwischen den verschiedenen Ebenen wechseln; in der objektorientierten (OO) Analyse werden Geschäftsobjekte identifiziert und analysiert, im OO-Design werden wenn möglich dieselben Objekte zu komplexen, implementierbaren Strukturen zusammengesetzt und schließlich in der OO-Implementierung in programmiersprachlichen Objekten kodiert. Diese Durchgängigkeit des objektorientierten Konzepts machen sich auch Programmierumgebungen zunutze, indem sie für die Analyse und das Design einheitliche Modelle zur Verfügung stellen, aus denen automatisch Codeteile generiert werden können. In einem anderen Schritt (dem »Reverse-Engineering«) können aus dem geschriebenen Code auch wieder Analyse- und Designmodelle generiert werden, so daß ein Aufgreifen der Analyse auch nach teilweise erfolgter Implementierung problemlos möglich ist[100].

Im *vierten Prinzip* wird beschrieben, daß die Vorgehensweise auf erfahrungsbasierte Aktivitäten und Aufgaben als treibendem Moment beruht und nicht auf Dokumenten. Auf die Nachteile, die bei der extensiven und zentralen Verwendung von Dokumenten entstehen, habe ich bereits an anderer Stelle hingewiesen. Die Sicherung der Konsistenz, der Abgleich nach Änderungen sowie die erforderliche schnelle Übersicht machen ein dokumentgetriebenes Vorgehen für eine flexible Vorgehensweise in einem Projekt mit veränderlichen Anforderungen und Randbedingungen hinderlich. Um den Entwicklern im Team eine Hilfe bei der täglichen Arbeit an die Hand zu geben, sollen stattdessen Beschreibungen von sogenannten »best practices« in Form von OPEN-Aktivitäten und -Aufgaben zur Verfügung stehen. Für wenige zentrale Dokumente, die immer wieder auftauchen, sollen aber durchaus Vorlagen bereitgestellt werden, die den Entwicklern als Ressource und Empfehlung dienen (vgl. auch Abschnitt 5.2.5).

Die Flexibilität in Planung, Projektstruktur und Vorgehen, die im *fünften Prinzip* gefordert ist, wird im Lebenszyklusmodell vor allem durch die projektbegleitenden Aktivitäten der »Projektplanung« und der »Anpassung des Vorgehens« realisiert. Anlehnend an die Software-Expedition von Mack (2001, vgl. Kapitel 4) wird auch in dieser OPEN-Instanz Planung als Ressource betrachtet, die situativer Beurteilung und Veränderung unterliegt. In

[100] Besonders gute Ergebnisse konnten im begleitenden Projekt in diesem Bereich mit dem Programm *Together Control Center* (aktuelle Version: 4.2) der Firma *Togethersoft* erzielt werden (vgl. http://www.togethersoft.com, Stand: Januar 2001).

stark standardisierten Projekten ist häufig festzustellen, daß die Arbeit doppelt durchgeführt wird. Einmal, zum produktiven Ausbau des Systems und zusätzlich, um die Dokumentationsstandards der Organisation zu befriedigen. Dies ist sicherlich nicht sinnvoll; daher ist es wichtig, daß die Teammitglieder durch ihr Feedback zur Verbesserung der Vorgehensweise beitragen. Die Einbeziehung dieses Feedbacks, das ich auch in Kapitel 3 als fehlenden Aspekt in den Beispielen zu OPEN herausgestellt habe, geschieht hier mit der Aktivität »Anpassung des Vorgehens« (vgl. Abschnitt 5.2.4).

Mit diesen Ausführungen habe ich dem *sechsten Prinzip* dahingehend entsprochen, daß alle Prinzipien in einem übersichtlichen Lebenszyklusmodell (vgl. Abbildung 5-1) abgebildet sind. Abbildung 5-2 zeigt dagegen einen frühen Entwurf des Lebenszyklusmodells, um die Entwicklung aufzuzeigen:

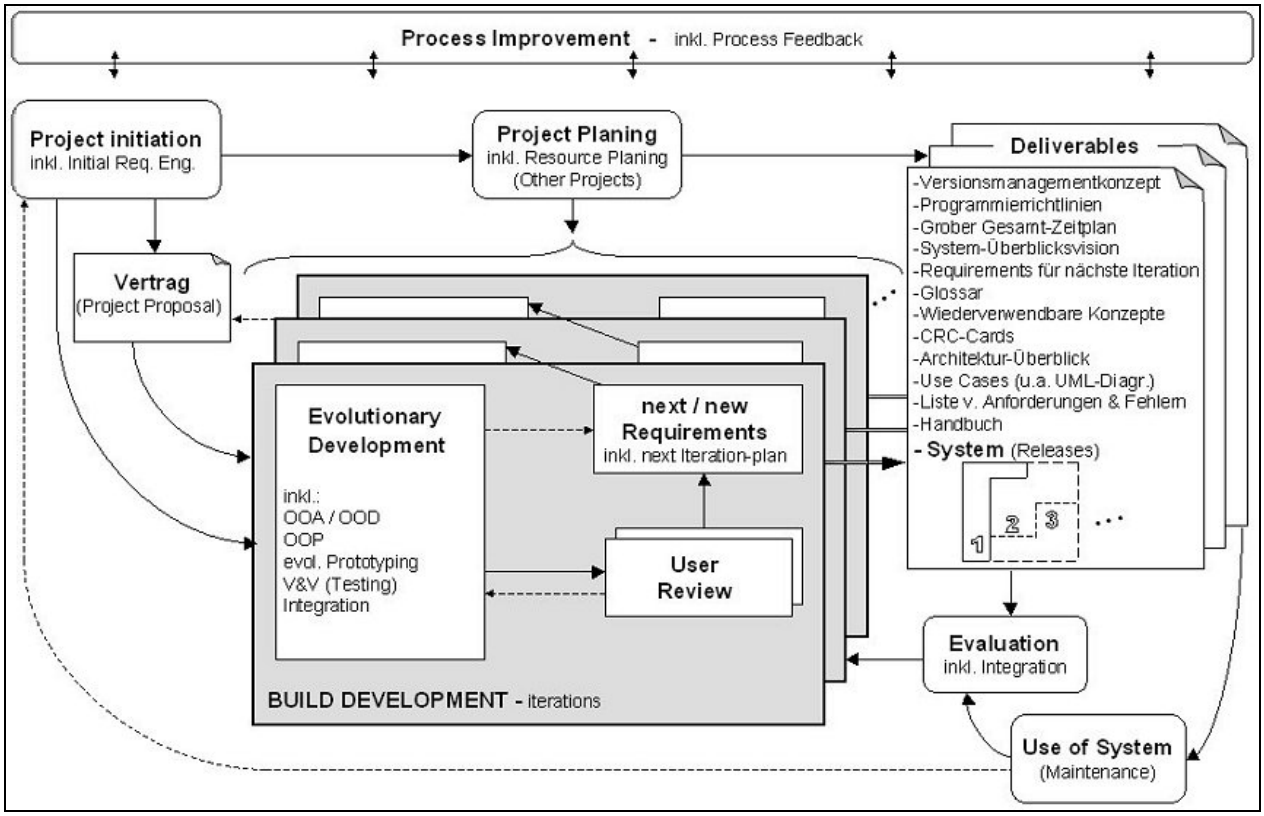

Abbildung 5-2 - Früher Entwurf des Lebenszyklusmodells

Auf Anhieb lassen sich hier einige Aspekte erkennen, die mit den erarbeiteten Prinzipien nicht vereinbar sind – z.B.:

- Die Übersichtlichkeit und Intuitivität sind durch die komplexe Darstellung der Aktivitäten, Vermischung von englischen und deutschen Begriffen sowie die

hohe Anzahl der dargestellten Elemente (z.B. Einbeziehung der auslieferbaren Ergebnisse) nicht gegeben;

- die Bedeutung der Pfeile ist uneinheitlich – es werden generelle Abhängigkeiten, zeitliche Abfolge und Erstellung Zuordnung von Aktivitäten und den erstellten Ergebnissen dargestellt;
- Personen werden nicht dargestellt;
- das Ineinandergreifen von Analyse, Design, Implementation und Testen wird nicht deutlich;
- eine Aktivität zur Anpassung des Vorgehens als Reaktion auf Veränderungen fehlt.

5.2.4. Die Aktivitäten des Lebenszyklusmodells

Im Folgenden werde ich kurz auf die Inhalte der einzelnen im Lebenszyklusmodell dargestellten Aktivitäten eingehen, um einen Eindruck vom tatsächlichen Projektverlauf nach den Ideen dieser OPEN-Instanz zu vermitteln:

Projektplanung und Anpassung des Vorgehens

Diese beiden Aktivitäten haben im Lebenszyklusmodell eine Sonderstellung, da sie während des gesamten Projektverlaufs berücksichtigt werden müssen.

Die *»Projektplanung«* umfaßt dabei typische Tätigkeiten des Managements. Es folgt eine erste Liste von typischen Aufgaben der »Projektplanung«, die in den allermeisten Projekten auftreten werden:

- Grobe Zeitplanung des Gesamtprojekts[101];
- Grobe Zeitplanung für Ablauf der Iterationen;
- Ressourcenüberwachung (Arbeitszeit, Hardware, Software, Tools, Planung, etc.);
- Beobachten von Risiken und Nutzen von Chancen;
- Budgetplanung;

[101] Beck (1999) weist darauf hin, daß sich die Entwicklungszeit (also die Zeit, die tatsächlich zur Entwicklung zur Verfügung steht) und die Kalenderzeit (also die tatsächlich verstrichene Zeit) aufgrund von Nacharbeiten zu anderen Projekten, Verwaltungsaufgaben etc. zum Teil wesentlich voneinander unterscheiden.

- Einrichten und Überwachen von Konfigurations- und Versionsmanagement;
- Schulen und Trainieren von Anwendern und Entwicklern (Repertoireanpassung);
- Teammotivation erhalten;
- Abstimmung mit anderen Projekten (Nutzen von Parallelen etc.);
- Erarbeiten von Notfallstrategien und Ausweichplänen;
- Teststrategie festlegen und Einhaltung überwachen;
- Kommunikationskoordination.

Die *»Anpassung des Vorgehens«* beschäftigt sich mit der Erweiterung, Anpassung und Dokumentation der Vorgehensweise selber, auch um die Erfahrungen für andere Projekten nutzbar zu machen. Wichtig ist in diesem Rahmen auch, von Anfang an die Kommunikationsstrukturen zu klären und auf ihre Nutzbarkeit hin zu untersuchen. Wenn möglich sollte für ein gezieltes Feedback zum Vorgehen (vor allem wenn die Vorgehensweise neu eingeführt wird) ein regelmäßiger Termin zusätzlich zu informellen Gesprächen vereinbart werden[102]. Zusätzlich sollte die Einführung einer neuen Vorgehensweise unbedingt von einem nicht im täglichen Projektgeschäft involvierten Mitarbeiter, dem Vorgehens-Coach, begleitet werden (vgl. auch Abschnitt 5.3).

Projektinitialisierung

Die Aktivität *»Projektinitialisierung«* markiert klassischer Weise den Beginn des Projektes und dient dazu, einen Konsens über den groben Inhalt und die Form der Zusammenarbeit zwischen Auftraggeber und Entwicklern zu erzielen.

Wesentliche Aufgaben umfassen dabei:

- Einigen über erste Projektziele und Erfolgskriterien;
- Budget, groben Zeitplan und feste Termine dokumentieren;
- Erste Projektrisiken aufdecken;
- Erste Ideen zum Umfang der Iterationen (fachlich und zeitlich) sammeln;
- Erster Überblick über Systemanforderungen *mit Priorisierung* durch den Kunden / Anwender.

[102] denkbar ist eine Periodizität je nach Projekterfahrung der Teammitglieder von ca. 3 Wochen.

- Art der Zusammenarbeit festlegen (Vor-Ort-Arbeit, über Mitwirkungsgrad und Aufgaben der Kunden/Anwender einigen);
- Teamzusammensetzung beschreiben und Verantwortlichkeiten klären;
- Vorhandene/benötigte Ressourcen besprechen und abgleichen;
- Vorgehensweise festlegen/Abstimmen und ggf. Teammitglieder schulen;
- Aufbau der Infrastruktur (Beschaffen von Hardware, Bereitstellen von Räumlichkeiten und Konfiguration von Software);

Besonders die ersten fünf Punkte sollen in dieser Aktivität zu Projektbeginn nur *kurz* angesprochen und im Sinne des iterativen Ansatzes im Verlauf des Projektes verfeinert werden. Als Orientierung läßt sich für die »Projektinitialisierung« ein Zeitraum von etwa 2-5 Wochen veranschlagen. Dieser Zeitraum hängt natürlich extrem von der Projektgröße und anderen projektspezifischen Faktoren ab. Auch in Extremfällen sollte aber, um ein iteratives Vorgehen umsetzten zu können, eine Spanne von 2 Monaten nicht überschritten werden.

Projektdurchführung

Die Projektdurchführung und Entwicklung des Systems geschieht in Iterationen, die jeweils aus drei Aktivitäten bestehen. Diese sind im Prozeßmodell kreisförmig in einem Mehrfachpfeil angeordnet, um das iterative Vorgehen und das damit verbundene mehrfache Durchlaufen dieser Aktivitäten zu betonen.

Zunächst wird der *»Umfang für die nächste Iteration«* in Zusammenarbeit von Kunden und Entwickler festgelegt. Dabei liefert der Kunde über eine Priorisierung der Funktionalitäten fachliche Informationen dazu, was als erstes benötigt wird. Durch konsequentes Schätzen der Entwickler für ihre verschiedenen Aufgaben kann geschätzt werden, welches Ergebnis in der nächsten Iteration im gegebenen Zeitraum (2-4 Wochen) realisierbar ist.

Das System selbst wird in der Aktivität *»Evolutionäre Systementwicklung«*[103] (mit »OO-Analyse«, »OO-Design«, »OO-Implementierung« und »Testen«) realisiert. Dabei gilt es besonders die vereinbarten Entwicklungsrichtlinien inklusive des durchgängigen Testens

[103] Als Evolutionär wird die Systementwicklung dann bezeichnet, wenn Erfahrungen der Anwender mit den ausgelieferten und eingesetzten Versionen oder Teilsystemen explizit in die Weiterentwicklung des Softwaresystems einfließen und auf erste prototypische Entwicklung konsequent aufgebaut wird (vgl. Floyd 1994b).

einzuhalten (zur Verschmelzung von »OO-Analyse«, »OO-Design« und »OO-Implementierung« vgl. auch Abschnitt 5.2.3).

Am Ende jeder Iteration steht ein *»Anwender Review«*, in dem der Anwender Gelegenheit bekommt, die entstandenen Systemteile zu beurteilen. Außerdem kann hier über grundsätzliche Änderungen beraten werden. Mit dieser Aktivität bekommt der Kunde also die Möglichkeit, seine eigenen Lernefahrungen und neue Ideen in die Entwicklung mit aufzunehmen. Um diesen Austausch zu gewährleisten, ist von Beginn an darauf zu achten, daß auch auf Kundenseite Zeit für diese Reviews und Tests eingeplant werden muß.

Projektabschluß

Die Wartung, weitere Anwenderschulung und die Evaluation möglicher Folgeaufträge sind Gegenstand dieser Aktivität. Ein weiterer Punkt ist die Aufarbeitung der im Projekt gewonnen Erfahrungen. Dabei kann es sich um Erfahrungen mit der Vorgehensweise selbst, der Teamstruktur, des Lebenszyklusmodells oder auch der Verwendung bestimmter Softwarewerkzeuge handeln.

5.2.5. Dokumente und Vorlagen

Wie im vierten Prinzip erwähnt, soll sich das Team auf die Erstellung der notwendigen Dokumente beschränken. Ausschlaggebend dafür, ob die Empfehlung eines Dokumentes in der Vorgehensweise sinnvoll ist, sind folgende Fragen:

- Wird das Dokument als Orientierungshilfe, Überblick, Detailerklärung oder Verwaltungshilfe in der Zukunft für andere Personen wichtig sein?
- Sind die Abhängigkeiten zu anderen existierenden Dokumenten minimal, so daß sich Änderungen an einem anderen Dokument nicht wesentlich auf die Relvanz und Konsistenz von diesem auswirken?
- Ist die Form des Dokumentes auf andere Projekte anwendbar bzw. leicht übertragbar?

Nur wenn diese Fragen eindeutig bejaht werden können, ist die Aufnahme einer Vorlage für das betreffende Dokument gerechtfertigt. Für das begleitende Projekt haben sich aus diesen Überlegungen heraus folgende zentrale Dokumenttypen gefunden, für die Vorlagen erstellt wurden:

- **Testkonzept** – inkl. erster Vereinbarung, welche fachlichen Tests und Testdaten vom Kunden bereitzustellen sind[104];
- **Grobkonzept** – ein *grober* Überblick über das zu erstellende System, der während der »Projektinitialisierung« entwickelt wird;
- **Handbücher** – typischerweise getrennte System- und Benutzerhandbücher;
- **Glossar** – fachlich und technisch zur Etablierung gemeinsamer und eindeutiger Begriffe;
- **Iterationspläne** – zeitlicher und inhaltlicher Iterationsplan;
- **Projektüberblick** – als Einstieg in das Projekt, z.B. eine einfache HTML-Seite;
- **Analyse von Risiken & Chancen** – inkl. grober Alternativlösungen bei Eintreten besonders großer Risiken;
- **Entwicklungsrichtlinien** – inkl. Programmierrichtlinien, Systemspezifikation, Versionsmanagementkonzept, Strategien/Prinzipien, etc. und
- **Standard-Vorlagen** – für Präsentationen, Statusberichte, Gesprächsprotokolle etc.

Zusätzlich können im Projektverlauf aufgrund der spezifischen Situation weitere Dokumente erforderlich werden. Vor dem Hintergrund der oben aufgeführten Fragen wird dann beurteilt, ob diese Dokumente als persönliche Notiz ohne längerfristige Aufbewahrung, projektspezifische Zusatzdokumente oder in die Vorgehensweise zu integrierende Vorlagen realisiert werden. Innerhalb der Entwicklungsrichtlinien sollte zusätzlich ein generelles Konzept zum einheitlichen Aufbau aller Dokumente (u.a. ein Dokumentkopf mit Firmen- und Projektname, Logo, Datum, Autor, Status, Inhaltsverzeichnis etc.) beschlossen werden.

5.3. Erfahrungen aus dem Einsatz

Wie bereits erwähnt, kam eine erste grobe Version der hier angedeuteten OPEN-Instanz in einem Praxisprojekt zum Einsatz. Die Erfahrungen, die sich dabei ergeben haben, wurden bereits in die beschriebene Version der Instanz eingearbeitet, sollen in diesem Abschnitt aber noch einmal explizit zusammengestellt werden.

---

[104] Die Erfahrung aus vergangenen Projekten hat gezeigt, daß dies ein kritischer Punkt ist. Zu häufig weisen die Testdaten und fachlichen Tests der Kunden zu große Mängel auf, um ein produktionsreifes System angemessen zu realisieren. Daher sollte das Testkonzept frühzeitig mit dem Kunden entwickelt und besprochen werden.

5.3.1. Erfahrungen mit dem Kunden

Die wichtigste und zugleich unangenehmste Erfahrung des Praxisprojektes war das Drängen des Kunden auf eine frühzeitige und möglichst genaue Einschätzung der Gesamtkosten. Primärer Kontaktpartner des Entwicklerteams war die EDV-Abteilung des Kunden. Der Leiter und auch einzelne Mitglieder dieser Abteilung standen einem konsequent iterativen Ansatz und dazugehöriger Abrechnung »nach Aufwand« (vgl. auch Abschnitt 5.1) durchaus positiv gegenüber. Die zunehmende Näherung des Termins, an dem das Projekt dem Vorstand der Kundenorganisation vorgestellt werden sollte, bewirkte aber die Forderung nach immer detaillierteren Kosten- und Zeitschätzungen. Denn die EDV-Abteilung mußte dem Vorstand gegenüber begründen, warum nur durch große Investition in eine neue maßgeschneiderte Software die gewünschte Verbesserung erreichen konnte und welche Funktionen der Software mit welchen Kosten veranschlagt sind, um ggf. zumindest Teillösungen durchsetzen zu können. Konsequenz dieser absehbaren Forderung war die Entscheidung, wie oben beschrieben, eine ca. zweimonatige Vorstudie durchzuführen.

Dabei wurde es dennoch erreicht, an der Abrechnung nach Aufwand festzuhalten. Ein Festpreisprojekt kam aufgrund der vielfältigen unklaren Anforderungen und ungesicherten Vorstellungen, was das Neusystem eigentlich leisten soll, nicht in Frage. Als Kompromiß zum notwendigen Kostenüberblick für den Vorstand, wurde als Ergebnis der Vorstudie eine detailliertere Kostenschätzung erstellt, als in der Vorgehensweise zunächst vorgesehen war. Diese Kostenschätzung wurde aber durch mehrfache Hinweise darauf relativiert, daß zu diesem Zeitpunkt keine verläßliche Aussage über die genauen Kosten möglich war, die Aufstellung also nur eine grobe Idee war, die auf dem augenblicklichen Erkenntnisstand beruhte. Der Vorstand schien dieser Argumentation folgen zu können, da der Auftrag zur Projektdurchführung nach dem vorgeschlagenen Konzept gegeben wurde. Dabei wurde allerdings große Priorität auf die Einhaltung des Zeitrahmens gelegt, da eine Einführung des Systems z.B. ein bis drei Monate später als zunächst geschätzt aus geschäftspolitischer Sicht nicht sinnvoll wäre.

Ein wesentliches Argument zur Durchsetzung der iterativen Vorgehensweise bei der Vorstandspräsentation war, daß schon innerhalb der kurzen Vorstudie, Lernerfahrungen auf beiden Seiten (vor allem auch auf Seite des Kunden) dazu geführt haben, daß die initialen Systemanforderungen und die Priorität verschiedener Funktionen drastisch verändert wurden. So war z.B. die Anbindung von externen Anwendern über das Internet zunächst als eine optionale Erweiterung für eine spätere Ausbaustufe vorgesehen und hat sich im Laufe

der Vorstudie mit dem Erkenntnisgewinn der Kunden über die Leistungsfähigkeit eines solchen Moduls zur höchsten Priorität entwickelt. Die Neuimplementierung der relationalen Datenbank hingegen, wurde auf eine spätere Version verschoben, da sich herausstellte, daß die vorhandene Datenbanklösung mit der neuen Technologie durchaus sinnvoll nutzbar war.

Mit dem Wunsch, ein unabhängiges Teilsystem zur Kalkulation in der Entwicklung vorzuziehen und schon parallel zur Vorstudie mit der Implementierung zu beginnen, gab der Kunde ein weiteres Argument für die beschriebenen Prinzipien. Es bestand der Bedarf, eine Teillösung möglichst frühzeitig benutzen zu können, um zu beurteilen, inwieweit die gewünschten Ergebnisse erreicht werden. Die gewählte iterative Vorgehensweise unterstützt dies durch die inkrementelle Auslieferung.

Ein typisches Problem tauchte vor allem in Gesprächen mit den Anwendern und Mitarbeitern der EDV-Abteilung des Kunden immer wieder auf: Die Entwickler berichteten, daß Gespräche über die groben Konzepte und Abläufe des Systems schwer zu führen waren. Immer wieder verloren sich Anwender und Kunden in Details zu Dingen, die für ein erstes Gesamtverständnis und die generelle Lokalisierung der Stärken und Schwächen des Systems eher hinderlich waren. Zur Klarstellung des augenblicklichen Fokus im Projekt half das benutzte Lebenszyklusmodell als erste Orientierungshilfe und speziell die Aktivität »Umfang für nächste Iteration« zur Identifikation der aktuellen Schwerpunkte. Mit zunehmender Kenntnis vom System konnten die Entwickler diesen Detailbeschreibungen zunehmend besser folgen und auch daraus neue Erkenntnisse ziehen.

5.3.2. Erfahrungen mit Entwicklern und Management

Die im vorigen Abschnitt erwähnte Forderung nach einer detaillierten Kosten- und Aufwandsschätzung hatte auch innerhalb der Entwicklungsorganisation Auswirkungen. Das Management der Firma erwog verschiedenste Lösungen (Outsourcing von Systemteilen, extreme Erweiterung des Projektteams durch externe Entwickler sowie Zukauf von fertigen Komponenten und Technologieplattformen), um vor allem den zeitlichen Umfang des Projekts zu verkürzen und Engpässen in der Bereitstellung von Personal zu begegnen. Nach kurzer Evaluierung stellte sich heraus, daß voraussichtlich keine der verschiedenen Möglichkeiten eine solche Produktivitätssteigerung mit vertretbarem Risiko erzielen würde.

Bei Gesprächen mit den Entwicklern des Teams über die einzusetzende Vorgehensweise kamen immer wieder die verwendeten Softwarewerkzeuge und Dokumentvorlagen ins Gespräch. Dies lag sicherlich vor allem daran, daß die Programme und Dokumente die Arbeitsobjekte sind, mit denen Entwickler täglich zu tun haben. Folglich sind sie daran interessiert, daß diese sinnvolle Unterstützung für typische Aufgaben liefern. Häufig genannte Wünsche der Entwickler umfaßten dabei

- eine klare Struktur und Ordnung der Verzeichnisse,
- Checklisten zu dem, was in einer Aktivität oder Aufgabe zu tun ist,
- Checklisten innerhalb der Dokumentvorlagen sowie
- konkrete Anleitungen was wann wie zu erledigen ist.

Generelle Konzepte und Prinzipien interessierten die beteiligten Entwickler zwar auf einer theoretischen Ebene auch, in der Projektpraxis standen die Programme und Dokumente aber einfach im Vordergrund; das theoretische Gerüst verlor vor dem Hintergrund des allgegenwärtigen Zeitdrucks im Projektalltag an Bedeutung.

Diese Erfahrungen wurden in die verwendete Vorgehensweise mit aufgenommen: Checklisten wurden integriert, Vorlagen für zentrale Dokumente mit aufgenommen und in den Beschreibungen von Aufgaben und Techniken wurde versucht, konkrete Anleitungen als Ressource zur Verfügung zu stellen, die gerade für die komplette Abwicklung von Standardaufgaben sowie die Unterstützung unerfahrener Mitarbeiter genutzt werden könnten. Durch die projektspezifische Zusammenstellung wurde, den Teammitgliedern zugleich ein weitreichender und flexibler Handlungsspielraum belassen. Außerdem wurde versucht, durch ein übersichtliches Lebenszyklusmodell eine erste Orientierung zu vereinfachen und gleichzeitig die grundlegenden Prinzipien für alle Teammitglieder ständig präsent zu halten.

Um die Vorgehensweise erfolgreich einzusetzen, ist es allerdings zwingend erforderlich, daß sich alle Teammitglieder auf die Vorgehensweise und darin vor allem auf die Aktivität »Anpassung des Vorgehens« einlassen. Nur durch das Feedback der beteiligten Personen zum Vorgehen und die daraus resultierende Integration von neuen Ideen in die Vorgehensweise selbst, kann eine angemessene Unterstützung gewährleistet werden.

Alle Überlegungen zu einer Vorgehensweise sind darüber hinaus obsolet, wenn die Einführung nicht durch ein klares Konzept gestützt und gefördert wird. In allererster Linie sind

hier Schulungen sowie die Integration eines Vorgehens-Coaches für die ersten Projekte notwendig. Zusätzlich muß die Vorgehensweise durch das Management (und natürlich die Projektleiter im Projekt) vertreten und gefördert werden, und es sollte regelmäßige Reviews zur Verbesserung der Vorgehensweise geben. Schließlich gilt es, die Vorgehensweise möglichst zügig in weiteren Pilotprojekten einzusetzen und auszuprobieren, um auf weitere Probleme der Praxis aufmerksam zu werden und diese in der Vorgehensweise betrachten zu können.

Ein Schulungskonzept könnte in folgenden Schritten verfolgt werden:

1. Einigkeit durch Kurzvorträge und anschließende Diskussion mit Vorstand und Geschäftsführung über Vorgehensweise herstellen;
2. Schriftliche Ausarbeitung und Vortrag für Projektbereichsleitungen[105];
3. Vortrag oder Kurzschulung für Projektleiter und einen zusätzlichen Mitarbeiter – dem Vorgehensverantwortlichen – aus dem Projekt oder dem Projektbereich;
4. Schulung der restlichen Entwickler durch Unterstützung des Vorgehensverantwortlichen.

Die Aufwände zur Schulung der Mitarbeiter sind dabei abhängig von ihrer Vorbildung und der Verfügbarkeit von bereits geschulten Ansprechpartnern im Projekt und im Arbeitsumfeld. Grundsätzlich geht es aber bei den Schulungen nicht darum, alle Detailüberlegungen der Vorgehensweise oder von OPEN weiterzugeben. Vielmehr wird sich auf die wesentlichen Aspekte, die zur Anwendung der Vorgehensweise notwendig sind beschränkt, und weitere Informationen werden bei Bedarf zur Verfügung gestellt.

Zusätzlich sollten projektspezifisch Überlegungen angestellt werden, ob die Vorgehensweise komplett oder evtl. schrittweise eingeführt werden soll. Die Zusammenstellung eines Teams mit Mitarbeitern, die in der Verwendung der grundlegenden Prinzipien (vgl. Abschnitt 5.2.2) erfahren sind, ermöglicht die volle Nutzung der Vorgehensweise. Auf der anderen Seite kann es in einem Team mit einigen unerfahreneren Entwicklern oder einem

[105] evtl. ist auch ein Workshop zur Klärung von Unstimmigkeiten zwischen den beteiligten Personen hilfreich. Hier kann dann auch eine Grund-Vorgehensweise gefunden werden, auf der projektspezifische Versionen aufbauen.

Projektleiter, der beispielsweise mit den Prinzipien der iterativen und inkrementellen Entwicklung nicht vertraut ist, sinnvoll sein, zunächst einige wenige Teile unter der Beratung eines Vorgehens-Coaches einzuführen. Hierbei sollte die generelle Umsetzung der Prinzipien eine deutlich höhere Priorität haben, als die Verwendung von Beschreibungen zu Aktivitäten, Aufgaben, Techniken oder Vorlagen aus OPEN oder der Einsatz bestimmter vereinbarter Tools.

5.4. Kritische Diskussion

Auch zu den entwickelten Grundideen für eine OPEN-Instanz sollen nun die in Kapitel 2 aufgestellten Fragen beantwortet werden:

(1a) Wie flexibel kann innerhalb der Vorgehensweise mit Veränderungen umgegangen werden?

Der flexible Umgang mit Veränderungen war einer der Hauptkritikpunkte an den bestehenden OPEN-Instanzen und damit Ausgangspunkte für die Erstellung einer neuen Instanz. Dieses Manko und die Erfahrungen aus der Betrachtung der Software-Expedition in Kapitel 3 spiegeln sich besonders deutlich in den zugrundeliegenden Prinzipien wider: Nach dem ersten Prinzip stehen die Personen und ihre Erfahrungen im Mittelpunkt. Die Teammitglieder werden also als »kreative Individuen« aufgefaßt, denen u.a. ein Lebenszyklusmodell *zur Unterstützung* bereitgestellt wird. Das fünfte Prinzip formuliert das in der Frage (1a) beschriebene Kriterium direkt: »Flexibilität in Planung, Projektstruktur und Vorgehen« wird zum Grundprinzip, daß es im Lebenszyklusmodell und im Projektalltag umzusetzen gilt. Das vorgestellte Lebenszyklusmodell reagiert darauf vor allem mit dem konsequent iterativen Vorgehen, der Aktivität »Anpassung des Vorgehens« und dem dynamischen Zusammenstellen der Repertoires. Das vierte Prinzip stellt die Dokumente in ihrer Bedeutung hinter den erfahrungsbasierten Aktivitäten der Teammitglieder zurück; zu viele Dokumente würden die situative Arbeit behindern und damit die Flexibilität einschränken.

Es soll hier aber nicht verschwiegen werden, daß mit der Forderung nach einem definierten Lebenszyklusmodell und Zusammenstellungen von konkreten Aufgabenbeschreibungen in den Repertoires doch bewußt eine leichte Einschränkung der vollkommenen Flexibilität stattgefunden hat. Völlige Neuorientierung ist zwar im Projektverlauf möglich, aber erstens nicht ohne weiteres zu jedem Zeitpunkt (eventuell muß beispielsweise das Ende einer Iteration abgewartet werden) und zweitens nur verbunden mit dem zusätzlichem Aufwand der

Reorganisation der Projektstruktur und evtl. des Lebenszyklusmodells. Ich halte diese Einschränkungen aber, gerade auch vor dem Hintergrund der weiter unten beantworteten Frage (2), für erforderlich und sinnvoll.

(1b) Inwieweit ermöglicht die Vorgehensweise Anpassungen an die spezifischen Bedingungen und die beteiligten Personen zu Projektbeginn?

Wie in Kapitel 3 beschrieben, liefert OPEN einige gute Konzepte, um zu Projektbeginn eine Vorgehensweise auszuwählen, anzupassen oder neu zu erstellen, die dann den spezifischen Bedingungen des zu unterstützenden Projektes gerecht wird. Diese Konzepte sind hier im Wesentlichen aufgenommen und ergänzt. Mit der Beschreibung der Grundideen für eine weitere OPEN-Instanz in diesem Kapitel, steht nun eine weitere Quelle für die Erstellung einer eigenen OPEN-Instanz zur Verfügung.

(2) Welche praktische Unterstützung liefert die Vorgehensweise für die täglichen Aufgaben im Projekt?

Die praktische Relevanz der OPEN-Instanz wurde vor allem durch die Diskussion und Kritik der Entwickler und Manager aus der *CTH Consult Team Hamburg GmbH* gewährleistet, die klarstellten, welche Aspekte ihnen in der alltäglichen Arbeit hilfreiche Unterstützung sein konnten. Auch in eigenen Erfahrungen aus der Unterstützung des die Arbeit begleitenden Projektes, wurde deutlich, daß für die Nutzung in der Praxis die Beschränkung auf wenige Kernaussagen und aufschlußreiche, einprägsame Grafiken unerläßlich ist. Als grundlegende Elemente für die OPEN-Instanz wurden in dem vorliegenden Kapitel daher die Prinzipien und das nach ausführlichen Gesprächen mit Projektbeteiligten, Management und Grafikern vielfach überarbeitete Lebenszyklusmodell vorgestellt. Als konkrete Unterstützung bei der täglichen Arbeit stehen darüber hinaus die Repertoires aus Aufgaben und Techniken von OPEN sowie Vorlagen für wenige, zentrale Dokumente zur Verfügung.

6. Schluß

Abschließend werde ich in diesem Kapitel noch einmal auf die Arbeit zurückblicken, die wesentlichen Punkte zusammenfassen und in einem Ausblick betrachten, wie die Gedanken dieser Arbeit weitergeführt werden können.

6.1. Zusammenfassung

In der vorliegenden Arbeit habe ich versucht, notwendige Aspekte der Flexibilität in Vorgehensweisen zur Softwareentwicklung herauszustellen, in einer Anpassung von OPEN zumindest ansatzweise umzusetzen und Erfahrungen aus der Praxis dieser Umsetzung aufzuzeigen. Dabei habe ich auf das Spannungsverhältnis zwischen maximaler Flexibilität in der Softwareentwicklung (zur Reaktion auf unausweichliche Veränderungen) auf der einen und maximaler Handlungsanleitung (zur Unterstützung der alltäglichen Arbeit von Entwicklern) auf der anderen Seite hingewiesen. Letztendlich gilt es in einem Softwareentwicklungsprojekt zu erkennen, welcher Grad an Flexibilität oder Handlungsanweisung für die spezifische Situation passend ist. Dazu habe ich in Kapitel 2 die wesentlichen Einflußfaktoren vorgestellt, die in ihrer Kombination jedes Projekt zu einem Einzelfall machen, der individuelle Auswahl oder Anpassung einer Vorgehensweise erfordert.

Aus der Diskussion der Vor- und Nachteile definierter Vorgehensweisen habe ich daraufhin drei Fragestellungen mit dazugehörigen Kriterien entwickelt, anhand derer eine Untersuchung von Vorgehensweisen in Bezug auf ihre

(1a) Flexibilität im Projektverlauf,

(1b) Anpaßbarkeit vor Projektbeginn und

(2) praktische Unterstützung für alltägliche Aufgaben

möglich war. Mit Hilfe dieser Fragestellungen wurden dann OPEN und die Softare-Expedition untersucht. Dabei stellte sich OPEN als eine Umgebung zur Erstellung von Vorgehensweisen heraus, die durch ihre Strukturierung in frei kombinierbare und erweiterbare Teile eine gute Voraussetzung zur Entwicklung flexibler Vorgehensweisen bietet. Eine kurze Analyse der in der OPEN-Literatur angebotenen Lebenszyklusmodelle (vor

allem des CDLM[106]) zeigte jedoch auch, daß dort die Flexibilität durch relativ starre Schemata eingegrenzt wurde. Als besonders positiv konnte bei OPEN aber die reiche Unterstützung im laufenden Projekt sowie die Vielfalt an Beispielen für die Erstellung von Lebenszyklusmodellen bewertet werden.

Die Software-Expedition bietet an dieser Stelle eine gute Ergänzung, da sie im Gegensatz zum CDLM, die schematische Gliederung und Beschreibung auf ein Minimum begrenzt, um gleichzeitig die situative Flexibilität im Projektalltag zu maximieren. Daher habe ich einige für die vorliegende Arbeit relevante Aspekte der Software-Expedition herausgegriffen und vorgestellt. Dabei ergaben sich als besonders wertvolle Aspekte der Flexibilität in Vorgehensweisen das Personen-orientierte Management, Aufbauen auf erfahrungsbasierter Flexibilität und das Auffassen von Planung als (veränderbare) Ressource.

Auf diesen Erkenntnissen, der Diskussion von Vorgehensweisen im Allgemeinen sowie der Software-Expedition und OPEN im Besonderen habe ich dann meine Vorstellungen für eine flexible Vorgehensweise als Instanz von OPEN beschrieben. Dabei habe ich mich nicht auf die rein theoretische Auseinandersetzung beschränkt, sondern versucht, Praxiserfahrungen aus einem die Arbeit begleitenden Projekt einzuarbeiten. Als Ergebnis habe ich sechs zugrundeliegende Prinzipien erarbeitet:

1. Beteiligte Personen stehen im Mittelpunkt;
2. Qualitativ hochwertige Software durch iterative Entwicklung, inkrementelle Auslieferung und kontinuierliches Testen sowie Einbeziehen von Lernerfahrungen durch häufige Reviews;
3. Ineinandergreifen von Analyse, Design, Implementierung und Testen statt Trennung in sequentielle Phasen;
4. Erfahrungsbasierte Aktivitäten und Aufgaben statt feste Dokumente als treibendes Moment – Bereitstellen von Vorlagen für nur wenige zentrale Dokumente;
5. Flexibilität in Planung, Projektstruktur und Vorgehen;
6. Realisierung der Prinzipien in einem übersichtlichen Lebenszyklusmodell mit strukturierenden Aktivitäten.

---

[106] CDLM steht für »contract driven lifecycle model« - vgl. Kapitel 3 sowie Graham et al. (1997).

Diese Prinzipien wurden dann gemäß dem sechsten Prinzip in einem Lebenszyklusmodell als übersichtliche Orientierungshilfe im Projekt umgesetzt. Dabei hat sich u.a. auch herausgestellt, daß in Vorgehensweisen, in denen eine weitreichende Unterstützung der Entwickler im Projektverlauf angeboten werden soll, leichte Einschränkungen der Flexibilität hingenommen werden muß. Dies bestätigt die zuvor beschriebene Theorie von Vorgehensweisen im Spannungsverhältnis zwischen maximaler Flexibilität und maximaler Handlungsanleitung. Allein das Aufstellen und Befolgen eines Lebenszyklusmodells birgt eine gewisse Einschränkung oder zumindest einen Zusatzaufwand, der mit einer Restrukturierung des Modells verbunden wäre. Sowohl die theoretische Diskussion in der Arbeit als auch die Anwendung in der Praxis haben aber ergeben, daß zumindest im die Arbeit begleitenden Projekt die Vorteile von zusätzlicher Orientierung, Strukturierung und Übersichtlichkeit diese leichte Abschwächung vollkommener Freiheit und Flexibilität ausgleichen.

So wurde für das begleitende Praxisprojekt das Kernstück einer flexiblen OPEN-Instanz geschaffen, das auch in anderen Projekten einsetzbar ist. Zur Optimierung und Verallgemeinerung sollte diese Vorgehensweise allerdings noch an verschiedenen Stellen kontinuierlich weiterentwickelt werden (vgl. Abschnitt 6.2).

Dabei kann auch der Blick zu anderen Vorgehensweisen in der Softwareentwicklung helfen[107]. Besondere Aufmerksamkeit erregen aktuell vor allem *Extreme Programming* (XP) als besonders flexible, leichtgewichtige Vorgehensweise mit Beschränkung auf wenige grundlegende Konzepte und Praktiken sowie der *Raional Unified Process* (RUP) als hochschematische und durchstrukturierte Vorgehensweise mit dem Anspruch, konkrete Arbeitsunterstützung für nahezu alle Bereiche der Softwareentwicklung zu liefern. Die *Software-Technik für Evolutionäre Partizipative Systementwicklung* (STEPS) hat schon früh flexible Ansätze zur Gestaltung von Vorgehensweisen eingeführt und vor allem den Lerneffekt, der durch die Zusammenarbeit von Kunden und Entwicklern auf beiden Seiten entsteht (und auch in der vorliegenden Arbeit berücksichtigt wurde) als wesentlich für die moderne Softwareentwicklung charakterisiert. Der *Werkzeug und Material-Ansatz* (WAM) stellt u.a. „eine Sammlung bewährter Konstruktions-, Analyse- und Dokumentationstechniken“ (vgl. Züllighoven et al. 1998, S. 10) zusammen, die bei der Entwicklung einer passenden Vorgehensweise helfen können. Mit dem *Feature Driven Development* (FDD)

[107] Zur Literatur der einzelnen Vorgehensweisen s. Fußnote 89 auf Seite 77.

wurde eine weitere leichtgewichtige Vorgehensweise entwickelt, die durch die Orientierung an Programmfunktionalitäten (den sog. »Features«) und Bereitstellung weniger kurz beschriebener Aufgaben vor allem die akkurate Schätzung des verbleibenden Aufwandes zu jedem Zeitpunkt ermöglichen soll. Das *V-Modell 97* schließlich stellt eine weitere schematische und sehr umfangreiche Vorgehensweise dar, die vor allem in Großprojekten die Koordination und Anleitung der Entwickler vereinfachen soll.

6.2. Ausblick

Zur weiteren Nutzung der in Kapitel 5 beschriebenen Vorgehensweise auch in anderen Projekten wird eine kontinuierliche Weiterentwicklung und Anpassung unumgänglich sein. Vor allem die Erweiterung und Aktualisierung der Beschreibungen von Aufgaben und Aktivitäten in OPEN wird durch das Aufkommen neuer Technologien und Verfahren ständiger Überarbeitung bedürfen. Die beste Möglichkeit diesen fortlaufenden Überarbeitungsprozeß praktisch umzusetzen ist, neuen Projekten einen Vorgehensberater zur Verfügung zu stellen, der mit dieser Aufgabe betraut wird. So wird das Projektteam von zusätzlichen Aufgaben befreit und kann sich dem Kerngeschäft widmen. Der Vorgehensberater kann weiterhin die Projektleitung in der Anpassung und Durchsetzung der Vorgehensweise im Projekt unterstützen und spezielle Erweiterungen an der Vorgehensweise vornehmen. So entsteht nach und nach und *aus der Praxis heraus*[108] eine breite Basis an Kurzbeschreibungen zu immer wiederkehrenden Aktivitäten und Aufgaben, die im Stil von OPEN in einer OPEN-Instanz zu einer passenden Vorgehensweise kombiniert werden können. Für diesen Weg der Weiterentwicklung hat sich auch die CTH Consult Team Hamburg GmbH entschieden, bei der das begleitende Praxisprojekt durchgeführt wurde[109].

Bei der CTH beginnen noch im Jahr 2001 weitere Projekte, die in weiten Teilen nach derselben Vorgehensweise ablaufen sollen. Dadurch wird es zu weiteren erfahrungsgeleiteten Anpassungen der Prinzipien oder des Lebenszyklusmodells kommen. Eine systematische Untersuchung dieser Projekte und evtl. eine durch Befragungen gestützte Erfolgsanalyse könnten weiterhin zur Verbesserung der Vorgehensweise beitragen. Zur Überprüfung der

---

[108] Ich habe diesen Punkt hervorgehoben, um auf die Wichtigkeit hinzuweisen, eine Vorgehensweise nicht nur theoretisch zu fundieren, sondern vor allem praktisch nutzbar zu machen.

[109] Vgl. CTH (2000).

übergreifenden Relevanz wäre außerdem eine Verwendung in anderen Organisationen und Kontexten wünschenswert.

Eine etwas andere Herangehensweise an die in der vorliegenden Arbeit betrachteten Ansätze könnte auch OPEN als »Handwerkszeug« für Software-Expeditionen betrachten, während hier ja umgekehrt Aspekte aus der Software-Expedition aufgegriffen wurden, um spezielle OPEN-Instanz zu schaffen. Mack (2001) stellt hierzu im Abschnitt 6.5 mit XP, UP und UML selbst einige softwaretechnische Lösungen vor, die den Ansatz der Software-Expedition um das softwaretechnische »Handwerkszeug« ergänzen sollen. Es wäre also zu prüfen, in wie weit sich auch OPEN zu diesem Zweck eignet.

Literatur

Balzert, H. (1996)

Lehrbuch der Softwaretechnik: Software-Entwicklung. Bd. 1. Heidelberg, Berlin, Oxford: Spektrum Akad. Verlag. (mit CD-ROM.)

Bassett, P.G. (1997)

Framing Software Reuse: Lessons From the Real World. Upper Saddle River, NJ: Prentice Hall, Inc.

Baudoin, C., Hollowell, G. (1996)

Realizing The Object-Oriented Life Cycle. Upper Saddle River, NJ: Prentice Hall, Inc.

Beck, K. (1999)

Extreme Programming Explained: Embrace Change. 1st ed. Reading, Mass.: Addison Wesley Longman.

Boehm, B.W. (1988)

A Spiral Model of Software Development and Enhancement. Computer (IEEE Publications), Vol. 21, No. 5, May 1988, pp. 61-72.

Booch, G. et al. (1999)

The Unified Modeling Language User Guide. Reading, Mass.: Addison-Wesley Longman.

Brooks, F.P. Jr. (1986)

No Silver Bullet Essence and Accidents of Software Engineering. Information Processing: H. J. Kugler (Ed.), Elsevia Science Publishers B.V. (North-holland) IFIP.

Coad, P. et al. (1999)

Java modeling in color with UML: enterprise components and process. Upper Saddle River, NJ [u.a.]: Prentice Hall PTR.

Constantine, L.L., Lockwood, L.A.D. (1994)

Fitting practices to the people. American Programmer 7(12): 21-27.

CTH (2000)

Vorgehen CTH. Papier zur Beschreibung der Vorgehensweise der Firma CTH Consult Team Hamburg GmbH, Osterbekstraße 90c, 22083 Hamburg.

CTH (2001)

Projektbericht des begleitenden Projektes bei der Firma CTH Consult Team Hamburg GmbH, Osterbekstraße 90c, 22083 Hamburg.

DeMarco, T. (1997)

Der Termin: Ein Roman über Projektmanagement. München, Wien: Hanser.

Derniame, J.C. (1999)

Software Process: priciples, methodology and technology. Berlin [u.a.]: Springer.

Dröschel, W. (Hrsg.) (1998)

Inkrementelle und objektorientierte Vorgehensweisen mit dem V-Modell 97. München [u.a.]: Oldenbourg.

Duden (1986)

Duden. Rechtschreibung der deutschen Sprache und der Fremdwörter. 19., neu bearb. Und erw. Aufl. Mannheim [u.a.]: Dudenverlag.

Duden Informatik (1993)

Ein Sachlexikon für Studium und Praxis. Mannheim [u.a.]: Dudenverlag.

Finkelstein, A. (Hrsg.), Kramer, J., Nusibesh, B. (1994)

SoftwareProcess Modelling and Technology. New York: Wiley.

Firesmith, D. G. et al. (1998)

Documenting a complete Java™ application using OPEN. Edinburgh Gate: Addison-Wesley Longman Limited.

Floyd, C., Reisin, F-M., Schmidt, G. (1989)

STEPS to software development with users. In Ghezzi, C.; McDernid, J.A. (Hrsg.): ESEC ´89. Lecture Notes in Computer Science Nr. 387. Berlin: Springer.

Floyd, C. (1995)

Theory and Practice of Software Development – Stages in a Debate. In: TAPSOFT´95. Theory and Practice of Software Development. 6/1995,
S. 25-41 Mosses, M. et al. (Ed.). Berlin: Springer.

Floyd, C. (1994a)

Software engineering – und dann? In: Informatik Spektrum 17/1 (1994), S. 29-37.

Floyd, C. (1994b)

Evolutionäre Systementwicklung und Wandel in Organisationen. In: Der GMD-Spiegel 3/94 (September 1994), S. 36-40.

Floyd, C., Züllighoven, H. (1999)

Softwaretechnik. In: Rechenberg, P.; Pomberger, G. (Hrsg.) 1999. Informatik Handbuch. 2. Auflage. München, Wien: Hanser.

Fowler, M., Scott, K. (2000)

UML Distilled. Second Edition. A brief guide to the standard object modelling language.2000. Reading, Mass. [u.a.]: Addison-Wesley.

Gamma, E. et al. (1995)

Design Patterns: Elements of Reusable Object-Oriented Design. Reading Massachusetts [u.a.]: Addison-Wesley Longman.

Goldberg, A., Rubin, K.S. (1995)

Succeeding with Objects. Decision Frameworks for Project Management. Reading, Mass.: Addison-Wesley Longman.

Graham, I. et al. (1997a)

The OPEN Process Specification. Edinburgh Gate. Addison-Wesley Longman.

Graham, I. (1997b)

Some problems with use cases... and how to avoid them. In OOIS'96, eds. D. Patel, Y. Sun and S. Patel, p. 18-27. London: Springer.

Graham, I. (1998)

Requirements Engineering and Rapid Development: an object-oriented approach. Harlow: Addison-Wesley Longman.

Hansel, J., Lomnitz, G. (2000)

Projektleiter Praxis: Erfolgreiche Projektabwicklung durch verbesserte Kommunikation und Kooperation. 3., neu bearb. Aufl. (1. Aufl. 1987.) Berlin, Heidelberg: Springer.

Henderson-Sellers, B. (1992)

A book of object-oriented knowledge: object-oriented analysis, design and implementation; a new approach to software engineering. New York, London: Prentice Hall.

Henderson-Sellers, B. (1995)

Object-Oriented Metrics: Measures of Complexity. New York, London: Prentice Hall.

Henderson-Sellers, B. et al. (1998)

The OPEN Toolbook of Techniques. New York: ACM-Press.

Henderson-Sellers, B., Bulthuis, A. (1998)

Object oriented metamodels. New York: Springer.

Henderson-Sellers, B., Unhelkar, B. (2000)

OPEN Modeling with UML. Harlow [u.a.]: Addison-Wesley.

Henderson-Sellers et al. (2001)

Offizielle Website des Object Oriented Process, Environment and Notation (OPEN): http://www.open.org.au/index.html, Stand: Januar, 2001.

Hohmann, L. (1997)

Journey of the Software Professional: A Sociology of Software Development. Englewood Cliffs, NJ: Prentice Hall, Inc.

Humphrey, W.S. (1997)

Managing Technical People: Innovation, Teamwork, And The Software Process. Reading, Mass.: Addison-Wesley Longman.

Jacobson, I. et al. (1992)

Object Oriented Software Engineering: A Use Case Driven Approach. Wokingham [u.a.]: Addison-Wesley [u.a.].

Jacobson, I. (1997)

Software Reuse: Architecture, Proces and Organization for Business Success. New York: ACM Press.

Jacobson, I. (1999)

The Unified Software Development Process. Reading, Mass.: Addison Wesley Longman.

Kruchten, P. (1999)

The Rational Unified Process: An Introduction. 2nd ed. Reading, Mass.: Addison Wesley Longman.

Lehman, M.M. (1980)

Programs, life cycles, and laws of software evolution. In: Proc. Of IEEE 68 (1980) 1060-1076.

Mack, J. (2000a)

Softwareentwicklung als Expedition. In: Ges. f. Informatik (Hrsg.): Informatiktage 1999. Fachwissenschaftlicher Informatik-Kongreß, 12.-13. November 1999 im Neuen Kloster Bad Schussenried. Leinfelden-Echterdingen: Konradin Verlag Robert Kohlhammer.

Mack, J. (2000b)

Software-XPedition – eine gelungene Verbindung aus Expeditionssicht und Extreme Programming. In: Mayr, H.C. et al. (Hrsg.): Software-Management 2000. Fachtagung 2.-3. November 2000 an der Philipps-Universität Marburg. Proceedings. OCG-Schriftenreihe, Bd. 149. Wien: Österreichische Computer Gesellschaft.

Mack, J. (2001)

Software als Expedition. Vorgelegt als Dissertation an der Univ. Hamburg: Selbstverlag. Vorabversion vom Dezember 2000. Voraussichtlich veröffentlicht im ersten Halbjahr 2001.

Madhavji, N. H. (1992)

Environment evolution: The prism model of changes. In: IEEE Trans. On Software Engineering, SE-18(5): 380-392, Mai 1992.

Mayrshofer, D., Kröger, H. A. (1999)

Prozeßkompetenz in der Projektarbeit. Hamburg, Windmühle GmbH.

McDermid, J.A. (1994)

Software engineer´s reference book. Oxford: Butterworth-Heinemann.

Meyer, B. (1994)

The Base Object-Oriented Component Libraries. Upper Saddle River: Prentice-Hall.

Meyer, B. (1995)

Object Success. Hemel Hempstead: Prentice Hall.

Meyer, B. (1997)

Object-Oriented Software Construction. 2nd ed. Upper Saddle River: Prentice-Hall.

Oestereich, B. (1999)

Erfolgreich mit Objektorientierung. München: Oldenbourg Wissenschaftsverlag GmbH.

Pasch, Jürgen. (1991)

Dialogischer Software-Entwurf. Dissertation im Fachbereich Informatik an der TU Berlin.

Pfleeger, S.L. (1991)

Software Engineering. The Production of Quality Software (2nd ed.). New York: Macmillan Publishing Co.

Piemont, C. (1999)

Komponenten in Java. Einsatz und Entwicklung von JavaBeans mit Visual Age for Java. Heidelberg: Dpunkt-Verlag.

Royce, W. (1970)

Managing the Development of Large Scale Software Systems. In: Proceedings of the IEEE WESCON, August 1970, S. 1-9.

Siegel, D. (1996)

Creating killer web sites: the art of third-generation site design. Indianapolis: Hayden.

Sommerville, I. (1995)

Software Engineering. Harlow: Addison Wesley.

Standish Group (1995)

The CHAOS Report. The Standish Group International, Inc.
http://www.standish-group.com, Stand: Januar 2001.

Unhelkar, B. (2000)

Persönliche Korrespondenz per e-mail.

Versteegen, G. (2000)

Projektmanagement mit dem Rational Unified Process. Berlin [u.a.]: Springer.

Waldén, K., Nerson, J.M. (1995)

Seamless object-oriented Software Architecture: Design and Analysis of Reliable Systems. New York [u.a.]: Prentice Hall.

Weltz, F., Ortmann, R.G. (1992)

Das Software-Projekt: Projektmanagement in der Praxis. Frankfurt / M., New York: Campus.

Wiegers K. (1999)

Software Requirements. Redmond, WA: Microsoft Press.

Wirfs-Brock, R. et al. (1990)

Designing Object-Oriented Software. Englewood Cliffs, NJ: Prentice Hall.

Wulf, V. (1994)

Anpaßbarkeit im Prozeß evolutionärer Softwareentwicklung. In: Der GMD-Spiegel 3/94 (September 1994) S. 36-40.

Züllighoven, H. (1998)

Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material-Ansatz. Heidelberg: Dpunkt Verlag für digitale Technologie.

Anhang A – Erklärungen zur OPEN-Terminologie

In diesem Anhang sollen die zentralen OPEN-Begriffe noch einmal aufgeführt werden, um die Referenz zur OPEN-Literatur zu erleichtern.

activity – OPEN-Begriff für *Aktivität.*

Aktivität – Strukturierungseinheit zur Gliederung des *Lebenszyklusmodells* (in OPEN: *Activity*).

Aufgaben – Hier: Kleinere Strukturierungseinheit für *Aktivitäten*, die beschreibt, was zu tun ist (in OPEN: *Tasks*).

CDLM – Abk. für *contract driven lifecycle model.*

contract driven lifecycle model – Standard- *Lebenszyklusmodell* in OPEN.

Deliverables – OPEN-Begriff für *auslieferbares Ergebnis.*

Ergebnis – Hier: Dokumente, Source Code, Pläne etc., die dem Kunden vorgelegt werden. In erweiterter Sicht werden in der vorliegenden Arbeit (nicht in OPEN) auch Erfahrungen als Ergebnisse betrachtet.

Instanz von OPEN – s. OPEN-Instanz.

Instanziierung – Hier: Erstellen einer Instanz (Vorgehensweise) aus einer Umgebung zur Definition von Vorgehensweisen (z.B. OPEN). Dazu gehört auch die Adaption oder Definition eines Lebenszyklusmodells.

Lebenszyklusmodell – (in OPEN: *life cycle model)* Gröbste Strukturierung und Orientierungshilfe in einer Vorgehensweise. Besteht i.A. aus *Aktivitäten* oder *Phasen.*

lifecycle model – OPEN-Begriff für *Lebenszyklusmodell.*

OML – OPEN Modeling Language; Alternative zur *UML.*

OPEN – **O**bject-oriented **P**rocess, **E**nvironment and **N**otation; Umgebung zur Definition von Vorgehensweisen (vgl. Kapitel 3).

OPEN-Instanz – OPEN-konforme *Vorgehensweise*; s. *Instanziierung.*

software engineering process – OPEN-Begriff die Beschreibung einer *Vorgehensweise.*

tasks – OPEN-Begriff für *Aufgaben.*

tailoring – OPEN-Begriff für das Zurechtschneiden einer Vorgehensweise innerhalb eines gegebenen Rahmens (z.B. OPEN). Als Ergebnis entsteht eine OPEN-Instanz mit entsprechendem Lebenszyklusmodell.

Techniken – beschreiben wie *Aufgaben* ausgeführt werden können (in OPEN: *Technique*).

techniques – OPEN-Begriff für *Techniken.*

Tools &Technologies – OPEN-Begriff für Softwarewerkzeuge und Technologien.

Vorgehen – Art, in der Anwendungssoftware in der Praxis tatsächlich konzipiert, entwickelt und betreut wird.

Vorgehensweise – Schriftlich fixierte oder mündlich tradierte Beschreibung, die als Vorlage für das Vorgehen dient und als solche Teil des Vorgehens selbst wird (in OPEN: *software engineering process*).

Anhang B – Aufgaben aus OPEN

Table 5.4 OPEN tasks in logical/temporal groupings.

(i) *User interactions and business issues*
Identify context
Identify source(s) of requirements
Identify user requirements
    Define problem and establish mission and objectives
    Establish user requirements for distributed systems
    Establish user DB requirements
Analyze user requirements
Model and re-engineer business process(es)
    Build context (i.e. business process) model
    Build TOM
    Convert TOM to BOM
    Do user training
    Prepare ITT
Undertake feasibility study
Obtain business approval
Develop BOM
Write manuals and prepare other documentation
Deliver product to customer

(ii) *Large-scale architectural issues*
Undertake architectural design
    Develop layer design
    Establish distributed systems strategy
    Select database/storage strategy
Construct frameworks (subtask)
Optimize the design

(iii) *Project management issues*
Develop software development context plans and strategies
    Tailor the lifecycle process
    Develop capacity plan
    Develop security plan
    Establish change management strategy
    Develop contingency plan
    Establish data take-on strategy
    Integrate with existing, non-OO systems
Undertake feasibility study
Develop and implement resource allocation plans
    Choose project team
    Identify project rôles and responsibilities
    Choose toolset
    Choose hardware
    Specify quality goals
    Specify individual goals
    Decompose programme into project
    Use dependencies in the BOM to generate first-cut project plan
    Develop timebox plan
    Develop iteration plan
    Set up metrics collection programme
    Manage subsystems
    Develop education and training plan

Quality issues
Test
    Perform class testing
    Perform cluster testing
    Perform regression testing
    Perform acceptance testing
Write manuals and prepare other documentation
Evaluate quality
    Analyze metrics data
    Evaluate usability
    Review documentation
Maintain trace between requirements and design
Undertake in-process review
Undertake post-implementation review

(iv) *Database issues*
Identify user database requirements (subtask)
Select database/storage strategy (subtask)
Identify persistent classes (subtask)
Map logical database schema
Design and implement physical database
    Distribution/replication design
    Operational and performance design
    Performance evaluation

(v) *Distribution issues*
Identify user requirements for DCS (subtask)
Establish distributed systems strategy (subtask)
Develop security plan (subtask)

(vi) *Modeling/building the system*
Analyze user requirements
Identify CIRTs
    Determine initial class list
    Identify persistent classes
    Identify rôles
    Refine class list
Construct the object model
Design user interface
Map rôles on to classes
Optimize the design
Undertake usability design
Code
Write manuals and other documentation

(vii) *Reuse issues*
Optimize reuse ('with reuse')
Create and/or identify reusable components ('for reuse')
    Construct frameworks
    Optimize for reuse
Manage library of reusable components

Anhang C – Techniken aus OPEN

Table 3.1 OPEN techniques in logical/temporal groupings.

(1) *User requirements*
Action research
Active listening
Activity grid construction
Analysis of judgements
Brainstorming
Concept maps
Context modeling
CRC card modeling
Domain analysis
Expected value analysis
Interviewing
JAD
Kelly grids
Power analysis
Questionnaires
RAD
Record and playback
Rich pictures
Roleplay
Scenario development
Simulation
Social systems analysis
Soft systems analysis
Storyboarding
Throwaway prototyping
Videotaping
Workshops

(2) *Project management and business issues*
Approval gaining
BPR
Business process modeling
Configuration management

Table 3.1 (Continued.)

Cost–benefit analysis
Cost estimation
CPM charts
Critical success factors
Customer (on-site) training
Envisioning
Gantt charts
Hierarchical task analysis
Impact analysis
Impact estimation table
Library management
Package construction
Package coordination
PERT charts
Pistols at dawn
PLanguage
Priority setting
Project planning
Prototyping
PSP
Risk analysis
Role assignment
Scenario development
Simulation
SMART goals
Social systems analysis
Systems acceptance
Systems audit
Team building
Throwaway prototyping
Timeboxing
TQM
Versioning (DBMS)
Walkthroughs

Table 3.1 (Continued.)

Genericity specification
Granularity
Hierarchical task analysis
Interaction modeling
Layering
Mixins
Petri nets
Qua-types
Refactoring
Relationship modeling
Responsibility identification
Rule modeling
Scenario development
Service identification
State modeling
Stereotyping
Time-threads
Visibility analysis

Other
Abstraction utilization
Application scavenging
Blackboarding
Business process modeling
Completion of abstractions
Connascence
CRC card modeling
Dataflow modeling
ER modeling
Event charts
Exception handling
Formal methods
Function points
Fuzzy logic and fuzzy modeling

Table 3.1 (Continued.)

Other
CIRT indexing
Class naming
Contract specification
Exception handling
Formal methods
GQM
Hierarchical task analysis
Literate programming
Quality templates
Refactoring
Rule modeling
Standards compliance

(3) *Modeling techniques*

Concepts/philosophy
Abstraction
Classification
Encapsulation and information hiding
Generalization (see also Classification)
Responsibilities and responsibility-driven design

Technical/low-level OOA/D
Abstract class identification
Class internal design
Class naming
Collaborations analysis
Contract specification
Creation charts
Delegation analysis
Design templates
Event modeling
Generalization and inheritance identification

Table 3.1 (Continued.)

Workflow analysis

(2a) *Quality*

Testing
Acceptance testing
Beta testing
Dependency-based testing
Integration testing
Package and subsystem testing
Regression testing
System acceptance
Unit testing
Usability testing

Inspections
Defect detection
Inspections
Reviews
Walkthroughs

Metrics
Cohesion measurement
Complexity measurement
Cost estimation
Coupling measurement
Function points
Law of Demeter
Metrics collection
Reuse measurement
Statistical analysis of data
Task points
Thresholds

Table 3.1 (Continued.)

Interfacing to relational DBMS and legacy code
Mapping to RDB
Normalization
Object replication
Object retention requirements
Path navigation
Query optimization
Reliability requirements
Storage of derived properties
Tuning of database
Versioning (DBMS)
Volume analysis

(5) *DCS*
DCS classification
DCS optimization
Distributed system partitioning and allocation
Implementation of distributed aspects of system
Scenario development

(6) *User interface*
Dialog design in UI
Screen painting
Usability testing

(7) *Reuse*
Access analysis
Application scavenging
CIRT indexing
Collaborations analysis
Completion of abstractions
Framework creation
Genericity specification

Table 3.1 (Continued.)

Hierarchical task analysis
Hotspot identification
Hypergenericity
Information engineering
Intelligent agent identification
Object lifecycle histories
Ownership modeling
Pattern recognition
Protocol analysis
PS–PPS
Refinement
Reverse engineering
Revision of inheritance hierarchies
Role modeling
Roleplay
Screen scraping
Simulation
Textual analysis
Transformations of the object model
Viewpoints
Visualization techniques
Zachman frameworks

(4) *Database*
Access analysis
Audit requirements
Clustering (DB)
Collapsing of classes
Database authorization
Database fragmentation
DBMS product selection
DBMS type selection
Indexing

Table 3.1 (Continued.)

Granularity
Layering
Library class incorporation
Library management
Pattern recognition
Reuse measurement
Revision of inheritance hierarchies
Scenario development
Variant analysis

(8) *Coding*
Class internal design
Creation charts
Generalization and inheritance identification
Inspections
Implementation of distributed aspects of system
Implementation of rules
Implementation of services
Implementation of structure
Law of Demeter
Mixins
Screen scraping
Storage of derived properties
Wrappers

(9) *Training and education*
Computer-based training
Games
Group problem solving
Internet and web technology
Lectures
Roleplay
Simulation
Train the trainer
Videotaping
Workshops

Erklärung

Ich erkläre hiermit, die vorliegende Arbeit selbständig durchgeführt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Hamburg, im Februar 2001

_____

Henrik Ortlepp