Vorgehensmodelle für objektorientierte Software-Entwicklung

Wolfgang Hesse

Zusammenfassung

Das Aufkommen der objektorientierten (OO-) Entwicklungsmethoden und -werkzeuge hat einen Umbruch in der Software-Entwicklung bewirkt, von dem auch die Vorgehensmodelle und Management-Verfahren betroffen sind. Ein Vergleich einiger bekannter OO-Analyse- und Entwurfsmethoden zeigt, daß sie sich zwar strukturell und terminologisch von älteren Methoden absetzen, daß die zugehörigen Vorgehensmodelle jedoch noch mehr oder weniger stark in der Tradition des sogenannten Wasserfall-Ansatzes stehen. Damit sind sie den neuen Entwicklungszielen und -verfahren nicht mehr angemessen.

Im zweiten Teil dieses Beitrags wird ein vom Autor entwickeltes Modell (*EOS*, für: *Evolutionäre objektorientierte Software-Entwicklung*) vorgestellt und in den Vergleich einbezogen. Es knüpft Entwicklungszyklen und -tätigkeiten in systematischer Weise an die Bausteine der Software-Entwicklung (System, Komponenten, Klassen, Subsysteme). Management-Verfahren, die auf diesem Schema aufsetzen, erlauben eine besser angepaßte, flexiblere und differenzierte Projektplanung und -steuerung.

1 Einleitung

Im Laufe der fast 30-jährigen Geschichte der Softwaretechnik ist eine Vielzahl von *Vorgehens-, Phasen oder Prozeßmodellen* entwickelt worden, die den Software-Entwicklungsprozeß idealtypisch beschreiben und die als Leitfaden für die Abwicklung konkreter Projekte gedacht sind. In der industriellen Praxis dominieren heute noch die sogenannten *Wasserfall-Modelle*, die Ende der 70-er Jahre aufkamen und durch die Arbeiten von B. Boehm weithin bekannt wurden. Dies hat eine von uns zusammen mit Arbeitspsychologen und Soziologen durchgeführte empirische Studie bestätigt (vgl. Berichte über das IPAS-Projekt, u.a. [WeO 92], [HeW 94], [BHS 95]).

Das wasserfall-artige Vorgehen ist u.a. durch die folgenden Eigenschaften gekennzeichnet:

- Software-Projekte werden in „Phasen“ unterteilt, die vorwiegend sequentiell von einzelnen Mitarbeitern oder Teams abgearbeitet werden.

- Jede Phase beginnt mit einer Menge von Anforderungen oder Spezifikationen, die in der/den vorangegangenen Phase(n) erarbeitet wurden. Ihr Ziel besteht darin, eine festgelegte Menge von Resultaten (oft auch als „Produkte“ bezeichnet) zu erbringen, die entweder Bestandteil des Endresultats werden oder als Spezifikationen für Folgephasen verwendet werden.

- Während der mittleren Phasen werden große Systeme hierarchisch in kleinere Bausteine (bezeichnet z.B. als Subsysteme, Komponenten oder Module) unterteilt, diese werden separat entwickelt, getestet und (re-) integriert. Das integrierte System wird in der Zielumgebung installiert und nach einem Akzeptanztest zur Benutzung freigegeben.

Dieses Vorgehensschema hat in der Vergangenheit vielfältige Kritik erfahren - unter anderem deswegen, weil es bürokratische Tendenzen unterstützt, nicht flexibel genug ist, um mit instabilen und sich ändernden Anforderungen umgehen zu können und weil es zwischen den einzelnen Phasen leicht zu methodischen und terminologischen Brüchen führt.

Um diese Nachteile zu vermeiden, wurden immer wieder Modifikationen und Alternativen zum Wasserfall-Modell vorgeschlagen und erprobt. Zu den bekanntesten alternativen Vorgehensweisen gehören das *prototyping* sowie die *inkrementellen* und die *Spiral-Modelle* (vgl. [HMF 92]). Fast allen diesen Ansätzen ist die Betonung des zyklischen, iterierenden Vorgehens (gegenüber der traditionellen linearen Vorgehensweise) gemeinsam. Damit trägt man u.a. dem oben genannten Problem der nicht hinreichend bekannten oder instabilen Anforderungen an eine Software-Entwicklung Rechnung.

Beispiele für Vorgehensmodelle, die nicht nur eine iterierende (schleifen-förmige) Entwicklung vorsehen, sondern Entwicklungszyklen im Sinne einer *evolutionären Software-Entwicklung* (vgl. [Leh 80]) zur Grundlage des gesamten Software-Prozesses machen, sind das STEPS-Modell von Ch. Floyd et al. [FRS 89] und das in Abschnitt 4 behandelte EOS-Modell (vgl. [Hes 95, Hes 96].

Gemeinsam mit der Sprache *Smalltalk*, d.h. vor nunmehr fast 20 Jahren, wurde der Begriff der *objektorientierten Programmierung (OOP)* geprägt und verbreitet. Mit der Erweiterung dieses Ansatzes zum *objektorientierten Entwurf (OOD)* und zur *objektorientierten Analyse (OOA)* wurde Ende der 80-er Jahre der Grundstein gelegt zu einem durchgängigen Vorgehen in der Software-Entwicklung, das erstmalig auf einem einheitlichen Strukturprinzip beruht. Damit lassen sich Strukturbrüche zwischen den einzelnen Entwicklungsphasen vermeiden oder mindestens abmildern. Inzwischen haben zahlreiche Autoren neue Entwicklungsmethoden veröffentlicht, die auf dem objektorientierten Anasatz aufbauen. Stellvertretend für viele andere seien hier die Methoden (und Bücher) OOSA von Shlaer und Mellor [ShM 88], [ShM 91], OOA/OOD von Coad und Yourdon [CoY 90], [CoY 91], OOAD von G. Booch [Boo 91], [Boo 94], OMT von J. Rumbaugh et al. [Rum 91] und OOSE von I. Jacobson et al. [Jac 93] genannt.

Aus den speziellen Zielsetzungen und Leitlinien der OO-Methoden ergeben sich einige besondere Anforderungen für das Vorgehen, die teilweise von den traditionell gegebenen Anforderungen abweichen :

- Durch die veränderte Betrachtungsweise erhalten die "Objekte" der Software-Entwicklung (z.B. die einzelnen Klassen, aber womöglich auch Komponenten oder Subsysteme) ein höheres Gewicht und eine größere Eigenständigkeit. Entwicklungsprozesse sollten nicht nur auf Systemebene, sondern für Bausteine jeder Größenordnung betrachtet und geplant werden.

- Läßt man eigenständigere Bausteine zu und entkoppelt deren Entwicklung auch zeitlich, so verwischen sich die Phasengrenzen. Die Projektplanung muß sich dann mehr an der Software-Architektur und den an einzelnen Bausteinen auszuführenden Aktivitäten als an systemweit definierten Projektphasen orientieren.

- Mit Hilfe von OO-Methoden erstellte Software-Systeme sollen langlebig und an veränderte Bedürfnisse anpaßbar sein. Dazu muß man echte System-"Zyklen" betrachten, die nicht nach der (erstmaligen) Installation des Systems beim Auftraggeber enden, sondern Pflege- und Revisions-Tätigkeiten sowie ein mehrmaliges Durchlaufen wichtiger Phasen wie z.B. Analyse und Entwurf einbeziehen.

- Mit Hilfe von OO-Methoden erstellte Software-Bausteine sollen wiederverwendbar sein. Dafür sind Anpassungen, Erweiterungen und Weiterentwicklungen notwendig, die auch auf der Ebene der Bausteine eher ein zyklisches als ein sequentielles Vorgehen erfordern.

Mit dem Aufkommen der OO-Entwicklungsmethoden hat sich die Frage nach den dazu passenden Vorgehensmodellen neu gestellt. Von den meisten der oben genannten Methoden wird nicht nur eine neue Terminologie und Notation angeboten, sondern auch ein eigenes Vorgehensmodell, d.h. eine verallgemeinerte, idealisierte Anleitung für die Steuerung und Abwicklung von Projekten im OO-Umfeld. Diese Vorgehensmodelle sollen im folgenden näher betrachtet werden.

In einer kürzlich fertiggestellten Studie (vgl. [Hes 97a]) habe ich einige OO-Methoden daraufhin untersucht, inwieweit die dort vorgeschlagenen Vorgehensmodelle Anforderungen wie die oben genannten unterstützen. Die folgenden beiden Abschnitte enthalten eine verkürzte Darstellung der Ergebnisse. Im 4. Abschnitt wird das oben genannte EOS-Modell den betrachteten Vorgehensmodellen gegenübergestellt. Im abschließenden 5. Abschnitt werden Schlußfolgerungen im Hinblick auf das Projekt-Management gezogen.

2 Dimensionen des Software-Entwicklungsprozesses und Vergleichskriterien

In diesem Abschnitt werden unter dem Stichwort *"Dimensionen"* einige wichtige Gesichtspunkte zusammengestellt, unter denen sich der Software-Entwicklungsprozeß betrachten läßt. Sie bilden u.a. die Grundlage für die Kriterien, an denen sich der Methodenvergleich orientieren soll.

Vier Dimensionen für die Betrachtung von Vorgehensmodellen:

Die meisten bekannten Vorgehensmodellen sind hauptsächlich unter dem *zeitlichen* Aspekt strukturiert - d.h. der Software-Prozeß wird in Phasen oder Tätigkeitsschritte unterteilt, die sequentiell (in manchen Fällen auch teilweise parallel) zu durchlaufen sind. Ein Beispiel dafür ist das oben angeführte Wasserfall-Modell.

Zu einer differenzierteren Betrachtung gelangt man, wenn man weitere Dimensionen hinzunimmt. Ledgard and Marcotty haben schon früh vorgeschlagen, eine Dimension *Raum* zu betrachten und dabei einen *Problem-Raum* und einen *Lösungs-Raum* zu unterscheiden (vgl. dazu [HeE 90]). In seinem V-förmigen Modell grenzt B. Boehm durch die sogenannte Anforderungs-Linie (*requirements baseline)* einen *Anwendungs-Raum* vom *Entwicklungs-Raum* ab [Boe 79]. In meiner eigenen "Software-Technologie-Landschaft" (vgl. [Hes 84]) habe ich mit zwei Dimensionen gearbeitet: einer Raum-Dimension (mit den Ebenen "Problem", "Modell" und "Realisierung") sowie einer Dimension für die Formalität von Beschreibungen bzw. von Überprüfungen. Für die Klassifizierung

von Werkzeugen kam als dritte Dimension die Automatisierung hinzu. Henderson-Sellers and Edwards unterscheiden ebenfalls einen Problem-Raum *(problem space),* einen Modell-Raum *(solution space model)* und einen Lösungs-Raum *(solution space)* [HeE 90].

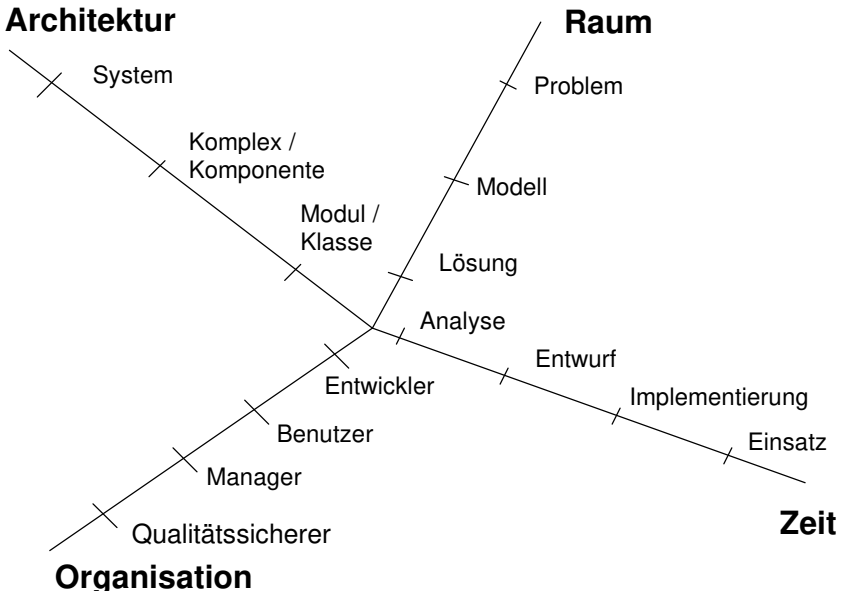

Abb. 1: Dimensionen des Software-Prozesses

Eine weitere Dimension bildet die Software-*Architektur*: Auf das Vorgehensmodell hat diese insofern Einfluß, als bestimmte Phasen oder Tätigkeiten an Elemente der Architektur, also z.B. an Bausteine wie Komponenten, Subsysteme oder Module geknüpft sein können. Vorkehrungen für die Wiederverwendung von Software setzen ebenfalls auf der Software-Architektur auf.

Die vierte Dimension betrifft die *Organisation*: An der Software-Entwicklung sind in der Regel verschiedene Gruppen beteiligt: zum Beispiel (gegenwärtige oder künftige) Benutzer, Entwickler, Manager, Qualitätssicherungs-Personal, Werkzeug-Verwalter. Einige Vorgehensmodelle wie das deutsche V-Modell enthalten separate Submodelle für solche Gruppen [Vmo 97].

Vergleichskriterien

Die folgenden Fragen sollen als Leitfragen für die folgenden Abschnitte dienen:

- Wie ist das Vorgehensmodell als Ganzes aufgebaut, welche Phasen, Abschnitte, Zyklen etc. sind vorgesehen?

- Inwieweit haben die betreffenden Autoren die Forderungen nach zyklischem und inkrementellen Vorgehen oder nach Prototyping berücksichtigt?

- Hat die System-Architektur einen Einfuß auf das Vorgehensmodell, inwieweit gehen die "Objekte" der Software-Entwicklung in dieses ein?

- Hat die Forderung nach Wiederverwendbarkeit Auswirkungen auf das Vorgehensmodell?

Durch die ersten beiden Fragen sind die Dimensionen Raum und Zeit angesprochen, während bei den anderen zwei Fragen Architektur-Gesichtspunkte im Mittelpunkt stehen. Die Organisation ist nicht direkt Gegenstand einer Leitfrage. Indirekt spielen allerdings Aspekte der Organisation bei allen vier Fragen mit hinein, vor allem bei der zweiten Frage, was das Verhältnis von Entwicklern und Benutzern anbetrifft. Da das Projekt-Management im besonderem Maße von den Vorgehensmodellen betroffen ist, ist diesem ein eigener Abschnitt (Nr. 5) gewidmet.

3 Ein Vergleich der Vorgehensmodelle einiger bekannter OO-Methoden

Für alle in den folgenden Vergleich einbezogenen OO-Methoden haben die jeweiligen Autoren Vorschläge für das Vorgehen gemacht. Diese "Vorgehens-modelle" sollen im folgenden unter dem Gesichtspunkt beleuchtet werden, ob und inwieweit sie den speziellen Zielsetzungen der Objektorientierung Rechnung tragen. Dieses schließt *keinen* umfassenden Methodenvergleich ein, d.h. Aspekte wie Notation, Aufbau und Beziehungen der Grundkonzepte, Werkzeug-unterstützung etc. bleiben unberücksichtigt. Für solche generellen Vergleiche sei auf die zahlreich dazu vorhandene Literatur, wie z.B. auf [M-O 95] oder [Ste 93] verwiesen.

Einen ausführlicheren Vergleich von OO-Vorgehensmodellen, der eine Reihe weiterer Kriterien berücksichtigt, liefert [Hes 97a]. Dort wurden fünf OO-Methoden (darunter die hier angesprochenen vier) bezüglich der folgenden neun Kriterien miteinander verglichen: *Gesamt-Modellstruktur, Objektorientierte Merkmale, Evolutionäre Aspekte*, *Vollständigkeit, Systematik und Kohärenz, Umgang mit Komplexität, Vorkehrungen für Wiederverwendung, Diversifizierung* und *Dokumentation.* Ein weiterer Vergleich, der sich auf Fragen der evolutionären System-Entwicklung konzentriert, findet sich in [Hes 97b].

Die Vorgehensmodelle und ihre Gesamtstruktur

(1) *Object Oriented Systems Analysis (OOSA)* von S. Shlaer und S. Mellor

Shlaer und Mellor haben ihre Methode in zwei Büchern ([ShM 88] und [ShM 91]) publiziert. Das erste Buch enthält ein sequentielles Phasenmodell mit vier Phasen:

- Problem-Analyse (kurz: AN, mit dem Ergebnis „Analysemodell"),
- Externe Spezifikation (ES, mit gleichnamigem Ergebnis),
- System-Entwurf (SD, mit gleichnamigem Ergebnis),
- Implementierung & Integration (II, mit dem Ergebnis „Akzeptiertes System").

Mit der Gliederung des Analysemodells in die drei Teilmodelle *Informationsmodell, Zustandsmodell und Prozeßmodell* nehmen die Autoren die traditionelle Einteilung in drei komplementäre, aber eigenständige Teilmodelle vor. Die zentrale Rolle des Informationsmodells entspricht der objektorientierten Denkweise. Es wird aber eher im traditionellen Sinne als Basis für den folgenden Datenstruktur-Entwurf genutzt denn als Kern für den gesamten Entwurf, der sowohl strukturelle als auch verhaltensmäßige Aspekte umfassen sollte. Daten-Abstraktion und Vererbung kommen kaum vor und die „Objekte" unterscheiden sich lediglich in ihrer Bezeichnung von den früheren „Entitäten". Dieser traditionelle Standpunkt spiegelt sich auch im Vorgehensmodell wider: Die verschiedenen Teilmodelle werden unabhängig voneinander entworfen und ein Eintritt in eine neue Phase bedeutet mehr ein neues Aufsetzen mit veränderten Zielsetzungen als ein Weiterbearbeiten bereits erbrachter Ergebnisse auf ein kontinuierlich verfolgtes Ziel hin. In ihrem zweiten Buch [ShM 91] führen die Autoren eine Hierarchie von sieben Entwicklungsebenen ein, bauen aber kein neues Vorgehensmodell darauf auf.

(2) Object-Oriented Modeling and Design von J. Rumbaugh et al.

In diesem Buch schlagen die Autoren ein sechs-phasiges Modell mit den folgenden Phasen vor (vgl. [Rum 91])::

- Problemstellung (IN)
- Analyse (AN)
- Entwurf (DS, später unterteilt in Systementwurf (SDS) und Objektentwurf (ODS))
- Implementierung (IM)
- Test
- Operationale Phase (einschl. Wartung und Anpassung)

Wie schon die Phaseneinteilung zeigt, steht auch dieses Vorgehensmodell in der Tradition des Wasserfall-Modells. Die Phasen werden von den Autoren allerdings - verglichen mit den traditionellen Modellen - ausdrücklich als weniger wichtig eingeschätzt. Wichtig ist vielmehr die einheitliche Klassen- und Objektstruktur, die sich durch den gesamten Entwicklungsprozeß wie ein roter Faden durchzieht und die Phasenübergänge eher zweitrangig macht. Brüche (wie z.B. der Übergang zu einer anderen Notation) werden ausdrücklich vermieden.

Von den sechs Phasen sind nur die Analyse- und Entwurfsphasen ausführlich, die letzten beiden Phasen dagegen überhaupt nicht beschrieben. Ähnlich wie bei Shlaer/Mellor besteht das Analysemodell aus drei Teilmodellen, dem Objektmodell, dem Dynamischen Modell und dem Funktionalen Modell. Aspekte der Objektorientierung sind auf den zweiten Teil der Entwurfsphase *(object design)* konzentriert. Sie werden erst dann sichtbar, wenn die Phasen in Schritte und Aktivitäten zergliedert werden. Das Objektkonzept zieht sich jedoch - stärker ausgeprägt als bei Shlaer/Mellor - als führendes, einheitliches Leitkonzept durch den gesamten Entwicklungsprozeß: Ihm sind alle anderen Konzepte wie Funktionen, Beziehungen, Ereignisse, Zustände, etc. zu- und untergeordnet.

(3) Object-Oriented Analysis and Design with Applications von G. Booch

Booch hat seine Methode zum OO-Entwurf erstmals 1991 in einem zusammenhängenden Buch beschrieben [Boo 91] und sie dann für die 2. Auflage zur Analyse- und Entwurfsmethode erweitert [Boo 94]. Im Gegensatz zu den vorgenannten Autoren unterscheidet Booch zwei Entwicklungsebenen und schlägt dafür zwei Vorgehensschemata vor: den Mikro- und den Makro-Prozeß (vgl. [Boo 94, pp. 234 ff]). Für seine Beschreibung wählt Booch einen *bottom up*-Ansatz und beginnt mit dem *Mikro-Prozeß.* Dieser dient als Rahmen für iterative and inkrementelle Entwicklungsschritte und weist eine gewisse Ähnlichkeit zu Boehm's Spiralmodell auf [Boe 88]. Er besteht aus den vier Schritten:

- Identifiziere Klassen und Objekte (ID)
- Identifiziere die Semantik von Klassen und Objekten (SM),
- Identifiziere die Beziehungen zwischen Klassen und Objekten (RE),
- Implementiere Klassen und Objekte (IM).

Damit konzentriert Booch alle typisch „objektorientierten“ Entwicklungstätigkeiten auf den Mikro-Prozeß. Normalerweise läuft dieser in einem Projekt mehrfach ab.

Dagegen ähnelt der Makro-Entwicklungsprozeß mit seinen fünf Phasen

- Konzeptualisierung (CO),
- Analyse (AN),
- Entwurf (DS),
- Evolution (EV),
- Wartung (MA)

den traditionellen Wasserfall-Modellen. Er bildet den steuernden Rahmen für die darin enthaltenen Mikro-Prozesse. Das bedeutet im besonderen, daß Makro-Aktivitäten wie Analyse und Entwurf einem gleichartigen (Mikro-) Schema folgen, in dem es immer wieder um das Identifizieren, Spezifizieren und Implementieren von Klassen und Objekten geht. Der Makro-Prozeß weist die folgenden Besonderheiten auf:

(a) Die „Konzeptualisierung" wird als als separate Phase eingeführt und damit wird ihre besondere Bedeutung für die Festlegung der Kern-Anforderungen hervorgehoben. Dazu gehört es beispielsweise, mit - möglicherweise nicht-weiterverwendbaren - Prototypen zu experimentieren.

(b) Anstelle der üblichen „Implementierungsphase“ steht bei Booch „Evolution“. Damit wird zum einen eine terminologische Überschneidung mit der "Implementierung" im Mikro-Prozeß vermieden, zum anderen aber auch der evolutionäre Charakter dieser Phase hervorgehoben.

(c) Das Ende des Makro-Zyklus bildet die „Wartungsphase“. Das erinnert zwar sehr an die herkömmlichen Phasenmodelle, doch betont Booch auch hier die Evolution, die noch nach der Auslieferung der Software stattfindet („postdelivery evolution“) und erwähnt explizit mögliche Wiederholungen des Makro-Zyklus - auch nach Auslieferung des Produkts. [Boo 94, p. 249].

Booch gibt eine ausführliche Begründung für sein Konzept der miteinander verschränkten Makro- und Mikro-Prozesse. Sein Zwei-Ebenen-Modell soll die Tatsache verdeutlichen, daß in den zentralen Makro-Phasen (Analyse, Entwurf, Evolution) die wesentlichen Entwicklungstätigkeiten darin bestehen, immer wieder Klassen und Objekte zu identifizieren, zu spezifizieren und zu implementieren. Makro- und Mikro-Prozesse müssen miteinander verknüpft werden, um einen vollständig rationalen Entwicklungsprozeß zu erreichen. Dies ist nach Booch eine Voraussetzung für die Erreichung eines Qualitätsstandards, der etwa dem Prädikat "defined" im Capability Maturity Model (CMM) des amerikanischen Software Engineering Instituts (SEI) entspricht (vgl. [Hum 89]).

(4) *Object-Oriented Software Engineering - A Use Case Driven Approach* von I. Jacobson:

Ähnlich wie Booch verwendet auch Jacobson zwei Arten von Zyklen, die allerdings in einem ganz anderen Verhältnis zueinander stehen als bei Booch. Jacobson versucht, die methodischen Aspekte stärker von den Management-Aspekten zu trennen und führt deshalb nebeneinander ein sogenanntes Prozeßmodell und Projektmodell ein [Jac 93].

Das *Prozeßmodell* (auch *System Life Cycle* oder *Software engineering life cycle*, kurz *SLC* genannt) deckt die methodisch-technische Seite ab und besteht aus einer Vorphase und vier Hauptphasen:

- Projekt-Auswahl und -Vorbereitung (PP, Vorphase)
- Anforderungs-Modellierung (RM)
- Objekt-Modellierung (OM)
- System-Konstruktion (SC)
- System-Test (ST)

Diese Phaseneinteilung unterstreicht Jacobson's Sicht von objektorientierter Systementwicklung: Aufbauend auf dem Anforderungs-Modell, das aufgrund von Anwendungs-Fallstudien *(use cases)* erstellt wird, erfolgt die Modellierung der Software als System kooperierender Klassen bzw. Objekte, die dann in den Phasen System-Konstruktion und -Test in ein lauffähiges Gesamtsystem umgesetzt werden. Der Schwerpunkt liegt dabei mehr auf der Revision und Weiterentwicklung bestehender Systeme als auf der Neuentwicklung. Inkrementelle Entwicklung wird als Vorgehensart besonders bevorzugt: Man sollte mit einer Reihe von *use cases* mit höchster Priorität starten und dann mit Inkrementen vernünftiger Größenordnung (d.h. ca. 5-20 *use cases*) fortfahren, die sukzessive definiert, implementiert und in das Gesamtsystem integriert werden.

Dagegen konzentriert sich das *Projektmodell* (*Project model,* kurz: *PM)* auf eine Phaseneinteilung und Definition von Meilensteinen gemäß den Bedürfnissen des Management. Die Phasen sind folgendermaßen benannt:

- Vorstudie *(pre-study)*
- Machbarkeitsstudie *(feasibility-study)*
- Etablierung *(establishment)*
- Ausführung *(execution)*
- Abschluß *(conclusion)*

Aus Management-Sicht läßt sich das Prozeßmodell in das Projektmodell einbetten, indem man den Abschluß bestimmter (SLC-) Teilprozesse mit PM-Meilensteinen identifiziert. Die Idee, für die Entwicklung und für das Management zwei getrennte Modelle anzugeben, ist eher unkonventionell. Einerseits lenkt sie den Blick auf die unterschiedlichen Institutionen und Tätigkeiten im Projekt, andererseits führt sie zu terminologischen Überschneidungen und einer gewissen Redundanz. Um zu einer abschließenden Bewertung zu kommen, müßte man Erfahrungen aus konkreten Projekten berücksichtigen, die Jacobson's Prozeß- und Projektmodell nebeneinander benutzt haben.

Damit können wir ein erstes Fazit ziehen:

Alle hier betrachteten Modelle folgen im Prinzip einem *hybriden* Ansatz: Objektorientierte Entwicklung findet im wesentlichen auf einer untergeordneten Klassen- und Objektebene statt und ist in einen übergeordneten, wasserfall-artigen Gesamtprozeß eingebettet. In der Ausprägung und der Verteilung der Gewichte finden sich jedoch deutliche, teilweise recht erhebliche Unterschiede.

Bei der ersten Gruppe von Autoren (Shlaer/Mellor und Rumbaugh) dominiert die Wasserfall-Struktur. Spezifika der objektorientierten oder der evolutionären Entwicklung haben (abgesehen von einigen terminologischen Besonderheiten) kaum Einfluß auf die überkommene sequentielle Struktur der Vorgehens modelle. Rumbaugh et al. konzentrieren sich auf die methodischen Aspekte der OO-Entwickung und lassen erkennen, daß sie den Vorgehens- und Management-Aspekten keine primäre Bedeutung zumessen.

Bei Booch finden wir ebenfalls den erwähnten hybriden Ansatz, zugleich aber auch den ernsthaften Versuch, die Besonderheiten der OO-Entwicklung im Vorgehensmodell zu reflektieren. Das belegt der Mikro-Prozeß, der als echter Zyklus angelegt und wiederholbar in den Makro-Prozeß eingebettet ist. Booch bringt damit den Gedanken eines hierarchisch aufgebauten Vorgehensmodells ins Spiel. Er unterstützt die „Objektorientierung im Kleinen", bleibt aber „im Großen" dem Wasserfall-Modell treu. Bei Jacobson's zwei Vorgehensmodellen handelt es sich nicht (wie bei Booch) um eine Hierarchie von Modellen, sondern diese richten sich an unterschiedliche Zielgruppen (Entwickler bzw. Management). Während Booch's Mikro-Prozeß iterativen Charakter hat, sind Jacobson's Modelle beide linear und laufen in der Regel genau einmal ab.

Unterstützung des zyklischen Vorgehens im Vorgehensmodell

Häufig wird der objektorientierte Ansatz mit der Forderung verknüpft, Software *zyklisch, inkrementell* oder als *Folge von Prototypen* zu entwickeln. Shlaer & Mellor geben in ihren ersten Buch kaum Hinweise zu zyklischen Entwicklungsformen. Das zweite Buch heißt zwar „Objekt-Lebenszyklen“, befaßt sich aber eher mit Aspekten der Modellierung dynamischer Systeme als mit Entwicklungszyklen. In der jüngsten Veröffentlichung [ShM 96] wird "Iteration" zwar erwähnt, soll aber ausdrücklich "kontrolliert", d.h. auf Analyse- bzw. Entwurfsschritte und dort auf einzelne Bereiche (domains) beschränkt werden.

Rumbaugh et al. heben hervor, daß der gesamte Entwicklungprozeß eher iterativ als sequentiell verläuft. Entwicklungsschritte sollen auf zunehmend verfeinerten Detaillierungsebenen wiederholbar sein. Bei der Objektmodellierung ist ausdrücklich ein Schritt zur Überarbeitung und Verfeinerung des Modells vorgesehen. Auch die darauffolgenden Phasen (Entwurf und Implementierung) sind zur Verfeinerung der Analysemodelle gedacht. Inkrementelle Entwicklung und System-Evolution werden weniger betont. Das mag damit zusammenhängen, daß sich das Buch insgesamt auf die frühen Phasen konzentriert.

Booch definiert Entwurf als einen „inkrementellen, iterativen Prozeß“. Er benutzt den Begriff „round trip gestalt design“, um damit auszudrücken, daß Systeme weder in reiner *top down*- noch in *bottom up*-Manier entworfen werden sondern auf inkrementelle, iterative Weise durch schrittweises Verfeinern verschiedener (logischer und physischer) Sichten auf das Gesamtsystem. Diese Sichtweise wird vor allem beim Mikro-Prozeß deutlich. Dort ist unter anderem vorgesehen, Alternativ-Entwürfe probeweise umzusetzen, aus mehreren konkurrierenden Lösungen die beste auszuwählen oder eine erfolgte Implementierung durch Erweiterungen und Änderungen fortzusetzen. Da die System-Integration durch ständiges, inkrementelles Anreichern erfolgt, erübrigt sich nach Booch eine separate Integrationsphase.

Für Jacobson ist System-Entwicklung ist ein Prozeß kontinuierlicher Veränderung. Er betont den langfristigen Entwicklungs-Prozeß wesentlich stärker als das isolierte Erzeugen eines bestimmten Produkts aufgrund festgelegter Vorgaben. Damit sind die Ideen von zyklischer und inkrementeller Entwicklung fundamental für den gesamten Ansatz. Anwendungs-Fallstudien *(use cases)* können als Ausgangspunkt für *System-Inkremente* dienen. In Jacobson’s Projektmodell wird inkrementelle Entwicklung speziell empfohlen, um verschiedene *Systemversionen* schrittweise aus einem gemeinsamen Kern abzuleiten.

Um Software-Applikationen besser zu verstehen, empfiehlt Jacobson *Prototyping*. Er sieht dessen Wert hauptsächlich in den frühen Projektphasen, z.B. um bestimmte (vorgesehene) System-Eigenschaften intensiver zu beleuchten. Dazu bevorzugt er weiterverwendbare Prototypen, äußert sich aber skeptisch über das sogenannte *rapid prototyping,* das er eher als ein gern genutztes Feigenblatt für *„quick and dirty programming“* ansieht.

Einfluß der System-Architektur auf das Vorgehensmodell

Die meisten phasenorientierten Vorgehensmodelle nehmen auf den Systemaufbau, die dabei definierten Bausteine und deren Zusammenbau zu lauffähigen (Sub-) Systemen keinen expliziten Bezug. Betrachten wir das Royce'sche, von B. Boehm populär gemachte Wasserfall-Modell [Boe 76], so finden wir dort Phasenbezeichnungen wie "Grobentwurf" und "Feinentwurf" (*preliminary and detailed design*), sowie "Test und Integration", die keine Festlegungen über eine etwaige System-Zerlegung in kleinere Bausteine, deren gesonderte Entwicklung und späteren Zusammenbau treffen. In anderen Modellen ist bisweilen von "Komponenten", "Subsystemen", von "Modul-Design" oder "Modultest" die Rede, wobei die Entstehung und der Zusammenhang solcher Bausteine aber meistens offengelassen wird. Eine weitere Detaillierung führt dazu, einzelne Tätigkeiten an die Bausteine zu binden (vgl. [Hes 84]).

In einer objektorientierten Entwicklungs-Umgebung stellt sich die Situation neu dar: Die Bausteine der Software-Entwicklung bekommen ein größeres Eigengewicht. Dafür können verschiedene Gründe maßgeblich sein:

- Klassen, Objekte und andere Software-Bausteine werden als eigenständig agierende Einheiten angesehen, die physisch verteilt sein können, miteinander durch Nachrichten kommunizieren, einzeln wiederverwendet oder weiterentwickelt werden können.

- Man möchte mehrere Klassen - z.B. für eine bestimmte Anwendung - in Gruppen (sog. *Komponenten* oder *Subsystemen*, engl.: *clusters*) zusammenfassen.

- Da einzelne Software-Bausteine möglicherweise zu verschiedenen Zeitpunkten benötigt werden, läßt sich ihre Entwicklung auch zeitlich entkoppeln. Der Zwang, alle Bausteine synchron in einer systemübergreifenden "Phase" zu bearbeiten, entfällt.

Die gegenwärtige Diskussion um eine "OO-Software-Architektur" spiegelt diese Argumente wider: Man ist auf der Suche nach einem möglichst universell zu

verwendenden Bauplan für OO-Software-Systeme, der es erlaubt; Bausteine unter den genannten Gesichtspunkten zu definieren, (weiter) zu entwickeln und (wieder) zu verwenden. Für ein OO-Vorgehensmodell lassen sich daraus - je nach Betrachtungsweise - unterschiedliche Konsequenzen ziehen: Entweder man betrachtet die Architekturfrage als ein dem allgemeinen Phasenschema untergeordnetes Entwurfsproblem (d.h. Fragen nach der Systemstruktur stellen sich erst beim Entwurf) oder man erhebt die Systemstruktur zum zentralen Gliederungsprinzip und betrachtet die Entwicklungs-Abläufe als an die jeweiligen Software-Bausteine geknüpft.

Alle vier in den Vergleich einbezogenen Vorgehensmodelle folgen - mit unterschiedlicher Ausprägung - der ersten Alternative. Im ersten Buch von Shlaer und Mellor [ShM 88] kommen "Objekte" zwar als wesentliche Elemente des Informations-, Zustands- und Prozeßmodells vor, doch haben sie auf die Struktur des Entwicklungsprozesses kaum Einfluß. Es gibt keine durchgehende, die Entwicklungsphasen überdauernde Baustein-Struktur. Aus neueren Veröffentlichungen ([ShM 96]) ist allerdings ein deutlicher Trend zur einem mehr architektur-getriebenen Entwicklungsprozeß ablesbar. Dabei ist das Bemühen nach Standardisierung und Automatisierung bestimmend: So gibt es z.B. vorgefertigte Rahmen-Bauteile ("structures"), die mit Elementen aus dem Anwendungsbereich auszufüllen sind.

Bei Rumbaugh finden wir wie bei Shlaer/Mellor ein dreigeteiltes Analysemodell, doch wird hier die zentrale Rolle des *Objektmodells* sehr viel stärker betont. Die Autoren weisen ausdrücklich auf den Vorrang der Datenstruktur vor funktionalen Aspekten hin, Funktionen, Beziehungen, Ereignisse etc. werden um die Objekte herum gruppiert. Objektorientierung bestimmt als oberstes Leitbild die Analyse- und Entwurfsphasen, hebt aber die traditionelle Phaseneinteilung nicht auf.

Booch betont ausdrücklich, daß die Objektorientierung nicht nur ein neues Paradigma für den Software- Entwurf darstellt, sondern daß sie beträchtliche Konsequenzen für den Software Engineering-Prozeß in seiner Gesamtheit hat. Das belegen u.a. die beiden miteinander verschachtelten Entwicklungszyklen. Im Prinzip schließt jede Phase des Makro-Prozesses die Definition, den Entwurf und die Implementierung von Klassen und Objekten (gemäß dem Mikro-Prozeßschema) ein. Zur Gruppierung logisch zusammengehöriger Klassen führt Booch den Begriff der Klassen-Kategorie ein. Dies geschieht jedoch lediglich auf der Darstellungsebene (in den Diagrammen), auf das Vorgehensmodell hat es keinen Einfluß.

Auch Jacobson stellt die "Objektmodellierung" in den Mittelpunkt seines Vorgehens. Das "Heraus-Destillieren" von Objekten bzw. Klassen und

Subsystemen aus Anwendungsfällen ist eine der zentralen Aufgaben zur Gewinnung des Analysemodells, das die Grundlage für den weiteren Entwicklungsprozeß bildet. Speziell zur Unterstützung von wiederverwendbaren Software-Bauteilen werden sogenannte Komponenten gebildet, für deren Entwicklung und Pflege Jacobson ein eigenes Vorgehensschema angibt.

Vorkehrungen für Wiederverwendung von Systemen und Bausteinen

Die Idee der objektorientierten Software-Entwicklung ist untrennbar mit der Frage nach der Wiederverwendung von Software-Bausteinen und -Dokumenten verbunden. Obwohl die Wiederverwendung zu den Kernthemen der Objektorientierung gehört, ist die Unterstützung der einzelnen Methoden dafür recht unterschiedlich. Bei den Vorgehensmodellen reicht das Spektrum von Nichtbeachtung dieser Frage bis zur höchsten Priorität.

Shlaer & Mellor gehen in ihrem ersten Buch nur am Rande auf Fragen der Wiederverwendung ein. In ihrem Vorgehensmodell wird kein Bezug darauf genommen. Im zweiten Buch wird die Wiederverwendung von Prozessen im Kontext von Prozeßmodellen kurz diskutiert. Der Einsatz von Bibliotheken für wiederverwendbare Software wird nicht thematisiert. In der Methodendarstellung im WWW werden die *domains* als Einheiten für die Wiederverwendung ausgezeichnet. Sie bilden nach Vorstellung der Autoren weitgehend voneinander unabhängige Teilsysteme, die sich nach Belieben in unterschiedliche Systeme einbauen lassen. Auf das Vorgehensmodell hat dies keine Auswirkungen.

Rumbaugh et al. erwähnen Wiederverwendung zweimal: Einmal im Kontext der Objektmodellierung, wo Module aus früheren Entwürfen auf Wiederverwendbarkeit geprüft werden. In der Methoden-Zusammenfassung (Kap. 11) kommt dieser Schritt allerdings nicht vor. Die zweite Referenz ist ein eigenes Teilkapitel über Wiederverwendbarkeit unter der generellen Überschrift „programming style“. Damit bleibt das Thema auf die Wiederverwendung von Code beschränkt. Es gibt Richtlinien und Hinweise zum Einsatz von Vererbung - aber wiederum keine Erwähnung in der Methoden-Zusammenfassung. Klassenbibliotheken werden in einem Tutorial-Kapitel über OO-Sprachen erwähnt (Kap. 15), sind aber kein OMT-Standard.

Booch behandelt die Wiederverwendung in einen eigenen Abschnitt. Jedes Stück Software kann wiederverwendet werden: Code, Entwürfe, Szenarien, Dokumentation. Bei den OO-Sprachen sind es primär Klassen, die mit Hilfe von Generalisierung und Spezialisierung wiederverwendet werden können. Auf höherem Abstraktionsniveau können das auch Muster von Klassen, Objekten und Entwürfen sein. Booch hebt hervor, daß Wiederverwendung nicht einfach von

selbst stattfindet, sondern institutionalisiert werden muß. Das schlägt sich in seiner Beschreibung des Makro-Prozesses in den Aktivitäten zum „Muster-sammeln“ (*pattern scavenging*) nieder. Weiter sieht Booch explizit eine Basis-Klassenbibliothek (*foundation class library)* vor. Dort können wiederverwendbare Teile in Form von sogenannten *frameworks* (Zusammenfassungen von Klassen, die Dienste für einen bestimmten Anwendungsbereich anbieten) bereitgestellt werden.

Auch Jacobson widmet dem Thema „Systementwicklung und Wiederverwendung“ ein eigenes einführendes Teilkapitel. Er fordert, daß Wiederverwendung zu einer Selbstverständlichkeit werden müsse. Sie findet auf der Code-Ebene statt, sollte aber auch bei der Analyse, beim Entwurf und beim Test als natürlich angesehen werden. Wiederverwendbare Einheiten sind Module und Komponenten, die in einer Bibliothek gespeichert werden. Komponenten sind vorfabrizierte Einheiten, die man als Zusatzangebot zu den Elementen der Programmiersprache oder als Standard-Bausteine bei der Entwicklung von Applikationen nutzen kann. Die Erstellung und Nutzung der Komponenten beschreibt Jacobson zwar außerhalb seines Vorgehensmodells, sie spielen aber für den Gesamtprozeß eine wichtige Rolle.

4 EOS: Ein Modell für evolutionäre, objektorientierte Software-Entwicklung

In diesem Abschnitt wird ein Modell zur evolutionären, objektorientierten Software-Entwicklung (kurz: EOS) vorgestellt. Es beruht auf den folgenden Leitgedanken (vgl. [Hes 95], [Hes 96]):

- Objektorientierte Software-Entwicklung ist ein *hierarchischer, zyklischer Prozeß*, der sich auf unterschiedlichen Entwicklungsebenen in jeweils analoger Weise vollzieht.

- Die Systemstruktur bestimmt den Gesamtprozeß und seine Teilprozesse: Jeder System-Baustein hat seinen eigenen Entwicklungszyklus. Bausteine sind durch ihre Bezeichnung den Ebenen der System-Hierarchie zugeordnet. In EOS tragen sie die Bezeichnungen (und Kürzel) System (S.), Komponente (oder Komplex, X.), Klasse (K.) und Subsystem (SS.).

- Jeder Entwicklungsprozeß ist als Zyklus ausgelegt und nach dem gleichen Schema aufgebaut (s. Abb. 2). Dahinter steht der Gedanke, daß für jeden Baustein die vier grundlegenden Schritte bzw. Tätigkeiten Analyse (.A), Entwurf (.E), Implementierung (.I) und Operativer Einsatz (.O) relevant sind.

Der letztgenannte Schritt umfaßt Einsatz, Pflege, „Wartung“ und Revisionen des betreffenden Bausteins.

- Zyklen und Tätigkeiten sind orthogonal zueinander und können beliebig miteinander gekoppelt werden. Damit entsteht ein Raster für die Identifizierung und Beschreibung von zyklenbezogenen Tätigkeiten (z.B. SA für System-Analyse, SE für System-Entwurf, XE für Komponenten-Entwurf, KI für Klassen-Implementierung), das auch für administrative und Planungszwecke eingesetzt werden kann.

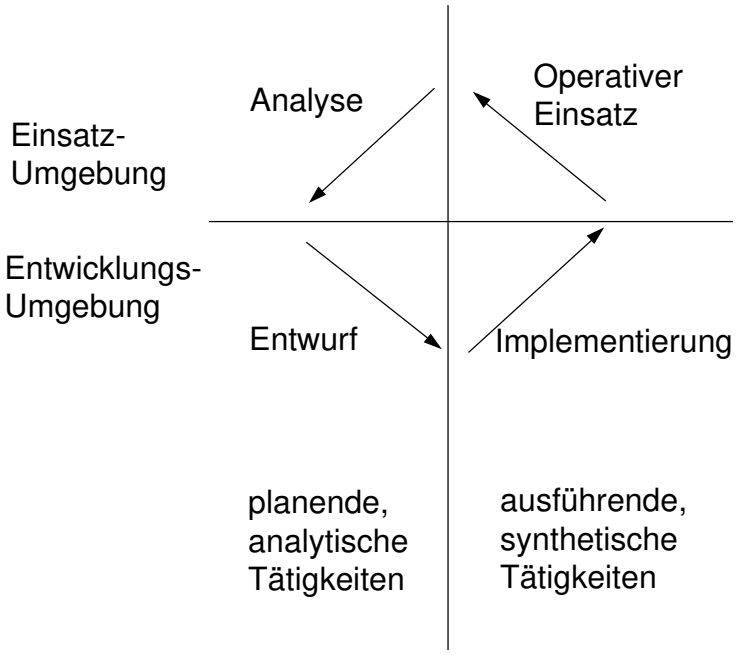

Abb. 2: Struktur eines EOS-Entwicklungszyklus

- Entwicklungszyklen für einzelne Systemteile und Bausteine sind nach dem Rekursionsprinzip definiert. Ihre zeitliche Abfolge wird nicht durch einen übergeordneten Phasenplan (Wasserfall) bestimmt, sondern aufgrund der jeweiligen aktuellen Projekt-Erfordernisse geplant und koordiniert (s. Abb. 3).

- Die herkömmlichen „Meilensteine“ werden durch sogenannte *Revisionspunkte* ersetzt. Diese sind mit den Referenzlinien des STEPS-Modells verwandt [FRS 89]. Sie sind allerdings stärker an der Bausteinstruktur des Systems und an der generellen Struktur der Entwicklungszyklen verankert. Unsere Revisionspunkte sind vom Management zu planende und jeweils an einen bestimmten Baustein gebundene Zeitpunkte im Projekt, die den jeweils zu erreichenden Entwicklungszustand eines Bausteins festlegen. Der Gesamtstatus des Projekts

ergibt sich aus der Zusammenschau der Zustände aller in Entwicklung befindlichen Bausteine.

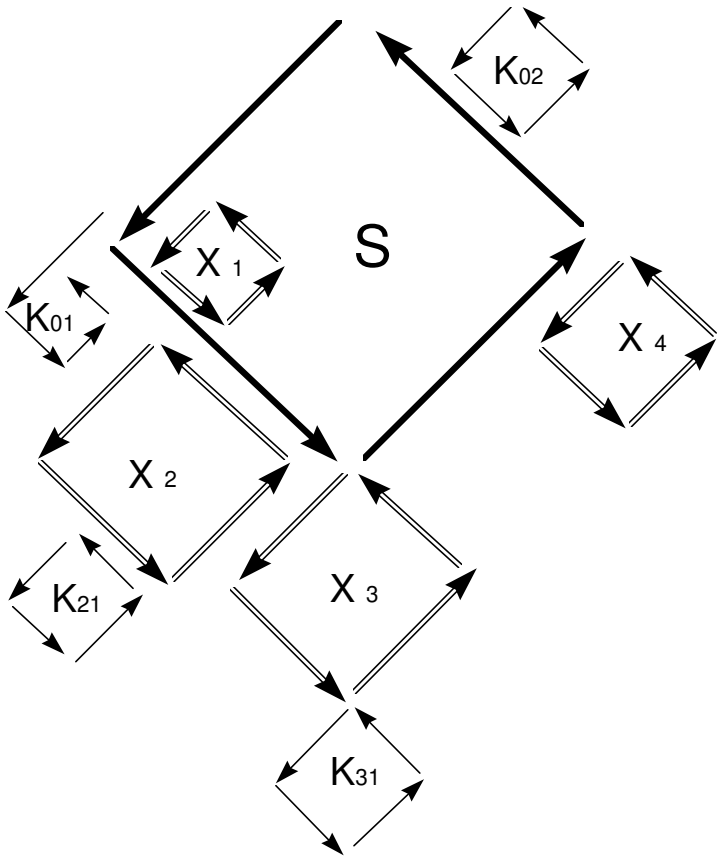

Abb. 3: Zeitlich verzahnte Entwicklungszyklen

Vergleich des EOS-Modells mit den oben genannten Modellen

Die oben betrachteten Vorgehensmodelle verfolgen einen hybriden Ansatz, bei dem objektorientierte Entwicklungsprozesse („im Kleinen“) in ein übergeordnetes Wasserfall-Modell („im Großen“) eingebettet sind. Dagegen verfolgt das EOS-Modell die Idee hierarchisch gestaffelter Entwicklungszyklen. Höchste Priorität genießen nicht irgendwelche Projektphasen, sondern die Bausteine der Systemstruktur, denen einzelne Tätigkeiten und Entwicklungsschritte zugeordnet sind.

Alle Zyklen haben die gleiche Grundstruktur und werden rekursiv definiert. Damit können sie beliebig tief gestaffelt werden. Der äußerste (System-) Zyklus steht an der Stelle des klassischen Wasserfall-Modells, unterscheidet sich aber in zweierlei Hinsicht davon: Erstens ist er selbst als echter Zyklus angelegt, d.h. *langfristige*

Evolution vollzieht sich als mehrfaches Durchlaufen dieses Zyklus. Zweitens spielen die Phasengrenzen nicht die dominierende Rolle für die untergeordneten Prozesse, sondern diese laufen weitgehend unabhängig davon ab. Zeitliche Koordination der Prozesse erfolgt mit Hilfe der Revisionspunkte.

Um Komplexität zu meistern, bietet das EOS-Modell die Konzepte der *Komponenten* und *Subsysteme*. Eine *Komponente* ist eine Zusammenfassung von Klassen oder anderen (kleineren) Komponenten unter logischen (Analyse- und Entwurfs-) Gesichtspunkten. Ein *Subsystem* ist eine Zusammenfassung von Klassen, Objekten oder anderen ausführbaren Einheiten mit dem Ziel der gemeinsamen Ausführung auf einem Rechner. Damit sind Subsysteme vor allem für die Phasen Implementierung und Operativer Einsatz wichtig. Im EOS-Vorgehensmodell spielen beide Konzepte eine hervorragende Rolle als Träger der mittleren Entwicklungsebenen und der daran hängenden Tätigkeiten. Die „fraktale“ Gesamtstruktur des Modells garantiert seine Anwendbarkeit auf Projekte beliebiger Größenordnungen.

Komponenten und Klassen sind die prädestinierten Einheiten für Software-Wiederverwendung. Zur Wiederverwendung ausgewählte und vorbereitete Bausteine werden in der Bausteinbibliothek (BBL) gespeichert und für anderen Nutzer zur Verfügung gestellt. Jeder Entwicklungszyklus (sei es auf System-, Komponenten- oder Klassen-Ebene) beginnt mit einer Bibliotheks-Suche nach wiederverwendbaren Bausteinen und endet mit einer Sichtung der erstellten Bausteine im Hinblick auf Aufnahme in die Bibliothek.

Komponenten, Klassen oder Subsysteme können auch als Inkremente aufgefaßt werden. Das heißt: sie werden in eigenen Zyklen entwickelt und dann im Rahmen eines übergeordneten Zyklus in die dort betrachtete Einheit (z.B. das Gesamtsystem) eingebaut. In ähnlicher Weise ordnet sich die Entwicklung von Prototypen in das EOS-Modell ein: Ein Prototyp wird (etwa als "Subsystem") in einem oder mehreren Zyklen entwickelt, wobei die Entwurfs- und Implementierungsschritte möglicherweise stark verkürzt sind. Das Resultat wird entweder in die laufende Systementwicklung eingebunden oder weggeworfen und durch eine alternative Entwicklung ersetzt.

5 Ausblick: Projekt-Management für objektorientierte Entwicklung

Vorgehensmodelle bilden die Grundlage dafür, Entwicklungs-Tätigkeiten zu strukturieren, zu koordinieren und zu dokumentieren. Das prädestiniert sie zu einem wesentlichen Instrument für das Projekt-Management. Die Grundstruktur des Vorgehensmodells liefert dem Projektleiter bzw. -manager ein Raster für die Planung von Tätigkeiten und Personaleinsatz, von Meilensteinen, Qualitäts-

sicherungs-Maßnahmen und Berichts-Zeiträumen. Damit üben die Vorgehensmodelle zweifellos einen deutlichen Einfluß auf die Denk- und Vorgehensweisen der Manager aus. Das haben u.a. auch die IPAS-Untersuchungen bestätigt: Die Mehrzahl der Projekte folgt - zumindest offiziell - einem explizit vom Management vorgeschriebenen Vorgehensmodell, dessen Struktur in der Regel wasserfall-artig ist [HeW 94].

Die Realität weicht allerdings nicht selten deutlich von dieser Vorgabe ab. Planungen lassen sich aufgrund von Verzögerungen, geänderten Anforderungen oder unerwarteten Ereignissen nicht aufrechterhalten oder werden bewußt unterlaufen. Nicht vorgesehene Entwicklungsschleifen oder Revisionen werden notwendig und machen die aufgestellten Pläne zur Makulatur. Oft gelingt es nicht oder nicht rechtzeitig, die Planung dem tatsächlichen Verlauf anzupassen und Projekte geraten in einem planlosen, schlimmstenfalls chaotischen Zustand.

Ein naheliegender scheinbarer Ausweg aus diesem Dilemma besteht darin, so grob wie möglich zu planen - dann fallen Abweichungen nicht so schnell auf. Wahrscheinlich haben die Wasserfall-Modelle unter anderem dieser Eigenschaft ihre Beständigkeit zu verdanken: Durch ihr i.a. recht grobes Phasen-Raster erleichtern sie die anfängliche Planung und ein möglichst langes Festhalten an fiktiven Plangrößen. Selbstverständlich ist damit auf die Dauer nichts gewonnen, sondern der Zeitpunkt des "bösen Erwachens" wird höchstens hinausgeschoben. Die Ergebnissen des IPAS-Projekts haben diese Thesen bestätigt: Am erfolgreichsten waren nicht die Projekte mit der rigidesten Planung und Planverfolgung, sondern diejenigen, bei denen man es verstand, sich schnell und unbürokratisch auf unerwartete Ereignisse und veränderte Projektsituationen einzustellen (vgl. [WeO 92], [HeW 94]).

Daraus leiten sich Forderungen nach einem flexiblen, *dynamischen Projekt-Management* ab. Eine Projektplanung sollte immer an veränderte Situationen und Bedingungen anpaßbar sein. Deshalb beginnt sie nicht notwendigerweise mit einem vordefinierten, formal verabschiedeten Plan, sondern dieser kann zunächst vorläufigen Charakter haben, muß aber ständig verfeinert und den aktuellen Projekterfordernissen angepaßt werden.. Anstelle eines einmaligen „Absegnens“ der Projektbedingungen zwischen Auftraggeber und Auftragnehmer tritt ein kontinuierlicher Kommunikationsprozeß.

Das Aufkommen der OO-Entwicklungsmethoden hat diese Situation noch verschärft. Dafür sind u.a. die veränderte System-Architektur mit sich verselbständigenden Bausteinen, die Tendenz zur Wiederverwendung und die gestiegene Bedeutung von zyklischen Vorgehensweisen verantwortlich. Wenn ein

Vorgehensmodell dem Management von OO-Projekten in dieser Hinsicht Hilfestellung leisten soll, so sollte es die folgenden Punkte unterstützen:

- die voneinander unabhängige Behandlung einzelner Bausteine (Komponenten, Subsysteme, Klassen),

- die Planung und Abwicklung von Entwicklungszyklen auf verschiedenen Ebenen (nicht nur auf der Ebene des Gesamtsystems) sowie

- die Planung und Koordination von Maßnahmen zur Wiederverwendung.

Ein Blick auf die im 3. Abschnitt betrachteten Vorgehensmodelle zeigt, daß sie die genannten Anforderungen nicht oder nur zu einem gewissen Teil erfüllen. Bei den Modellen von Shlaer/Mellor und Rumbaugh liegt das in erster Linie an ihrer Wasserfall-Struktur, die eine entsprechende Differenzierung nicht zuläßt. Im neueren Modell von Shlaer/Mellor wären eigenständige Entwicklungszyklen für die *domains* denkbar, doch lassen sich die Autoren nicht näher darüber aus.

Booch liefert mit seinem zweistufigen Prozeß-Modell einen Ansatzpunkt für eine ebensolche Planung und Planverfolgung: auf der Makro-Ebene ähnelt sie dem traditionellen Wasserfall, während auf der Mikro-Ebene ein zyklisches Vorgehen und damit die Notwendigkeit zur dynamischen Planung in den Vordergrund tritt. Es bleibt allerdings offen, inwieweit die Projekt-Manager in die konkrete Planung dieser teilweise doch detaillierten Prozesse eingebunden sind bzw. in welchem Maße man sie besser der Selbstorganisation der Entwickler überläßt.

Jacobson's Ansatz, dem Entwicklungsmodell ein eigenes Management-Modell gegenüberzustellen, ist unkonventionell und wegen der Hervorhebung der Management-Tätigkeien zunächst positiv zu bewerten. Allerdings wirft er die Frage nach der Koordination beider Zyklen auf. Interessant ist auch sein Ansatz, Komponenten (= wiederverwendbare Bausteine) in eigenen Entwicklungszyklen zu behandeln, doch gibt sein Modell keine Hilfestellung bei der Koordination solcher parallel laufender Entwicklungen.

Das EOS-Modell ist bewußt im Hinblick auf die Unterstützung eines dynamischen Projekt-Managements im Umfeld von OO-Projekten entwickelt worden. Dazu sollen u.a. die folgenden Eigenschaften dienen:

- An die Stelle der Phasenstruktur tritt die komplexere, aber weit flexiblere und der Systemstruktur besser angepaßte Zyklen- und Tätigkeitstruktur. Damit ist ein Schema gegeben, das Planungschritte jeder Größenordnung vorsieht und damit in allen Stadien einer Projektplanung anwendbar ist.

- Die Arbeitsteilung orientiert sich an den Bausteinen der Systemstruktur: Für Komponenten und Subsysteme sind Entwicklungs-Teams (falls notwendig, unter Einbeziehung von Anwendern), für Klassen einzelne Entwickler zuständig. Die Bindung einzelner Teams und Entwickler an „ihre“ Bausteine ist eine wesentliche Voraussetzung für deren „evolutionäre“ Entwicklung.

- Durch die "Revisionspunkte" ergibt sich ein klar vorgebenes Planungsraster als Hilfestellung für das Projekt-Management. Für alle in Betracht kommenden Bausteine muß deren geplanter Entwicklungszustand gemäß der EOS-Zyklen- und Tätigkeitsstruktur definiert werden. Revisionspunkte können nicht in allen Details zu Projektbeginn festgelegt werden. Sie werden vielmehr grob vorgeplant und dann jeweils gemäß dem aktuellen Projektstand verfeinert.

- Maßnahmen zur Qualitätssicherung setzen ebenfalls an der Zyklen-und Aktivitäten-Struktur an. Wichtige Reviews werden mit den Revisionspunkten kombiniert. Ihre Ergebnisse bestimmen die Planung weiterer Zyklen und die inhaltliche Festlegung künftiger Revisionspunkte.

- Software-Entwicklung wird als *kontinuierlicher Prozeß* angesehen, der im allgemeinen nicht mit der Installation eines „fertigen“ Produkts endet. Vielmehr gehen die Entwicklungstätigkeiten weiter - sei es als „Wartung“, als Weiterentwicklung oder als Wissenstransfer zu weiteren benachbarten Projekten.

Es ist klar, daß ein so vielschichtiges Vorgehensmodell hohe Anforderungen an die Projektführung stellt. An die Stelle der traditionellen, leicht überschaubaren Phasen treten komplexe und differenzierende Zyklen- und Tätigkeitsstrukturen, die entsprechende Management-Techniken erfordern. Das Management muß also zu Projektbeginn mehr in eine differenzierte Planung (und die dazu notwendigen Recherchen), in interne Abstimmungen und Überzeugungsarbeit beim Kunden investieren.

Auf das Ganze gesehen, ist es jedoch angemessener, ehrlicher und am Ende auch genauer und kostengünstiger, von vornherein differenziert, aber flexibel zu planen, und die Notwendigkeit von Plan-Anpassungen, Entwicklungsschleifen und Revisionen bei der Planung von Anfang an zu berücksichtigen. Neue Qualitätsstandards wie das CMM des SEI (vgl. Abschnitt 2) verlangen ebenfalls solch ein flexibles Management: nur so können die höheren Stufen "managed" oder "optimising" erreicht werden.

Noch ist nicht klar, ob und wie sich der objektorientierte Ansatz langfristig auf die Vorgehensmodelle auswirken wird. Vielleicht ist hier die Geschichte der *„unified*

method" symptomatisch: Das ursprüngliche, ambitionierte Vorhaben der drei Methoden-„Päpste“ Rumbaugh, Booch und Jacobson, eine standardisierte „Methode“ zu schaffen, wurde auf die Entwicklung einer gemeinsamen Begriffswelt und Notation - kondensiert in der "Unified Modelling Language" (vgl. [UML 97]) - eingeschränkt. Das signalisiert, daß man sich auf einen „unified process“, d.h. ein gemeinsames Vorgehensmodell nicht hat einigen können oder daß man zu der Erkenntnis gekommen ist, daß sich ein solcher Standard - der auch so subtile Bereiche wie Firmen-Organisation und -"kultur" berühren würde - zumindest zum gegenwärtigen Zeitpunkt nicht durchsetzen ließe.

Literaturverweise:

[BHS 95] U. Bittner, W. Hesse, J. Schnath: *Praxis der Software-Entwicklung, Methoden, Werkzeuge, Projektmanagement - eine Bestandsaufnahme.* Oldenbourg 1995

[Boe 76] B.W. Boehm: Software Engineering. IEEE Transact. on Computers, C-25.12, pp. 1216-1241 (1976)

[Boe 88] B.W. Boehm: A spiral model of software development and enhancement; *Computer,* May 1988, pp. 61-72

[Boo 91] G. Booch: *Object-Oriented Design with Applications.* Benjamin/ Cummings Publ. Comp. 1991

[Boo 94] G. Booch: *Object-Oriented Analysis and Design with Applications*. Second Edition, Benjamin/Cummings Publ. Comp. 1994

[CoY 90] P.Coad, E. Yourdon: *Object-oriented analysis.* 2nd edition, Yourdon Press 1990

[CoY 91] P. Coad, E. Yourdon: *Object-oriented design.*Yourdon Press 1991

[FRS 89] Ch. Floyd, F.-M. Reisin, G. Schmidt: STEPS to software development with users. in: *C. Ghezzi, J. McDermid (eds.): ESEC ‘89, Second European Software Engineering Conference.* LNCS 387, pp. 48-64; Springer 1989

[HeE 90] B. Henderson-Sellers, J.M. Edwards: Objects oriented software systems life cycle; *CACM* Vol. 33, No. 9 (1990)

[HMF 92] W. Hesse, G. Merbeth, R. Frölich: *Software-Entwicklung - Vorgehensmodelle, Projektführung und Produktverwaltung.* Handbuch der Informatik, Band 5.3, Oldenbourg 1992

[HeW 94] W. Hesse, F. Weltz: Projektmanagement für evolutionäre Software-Entwicklung. *Information Management* 3/94, pp. 20-33, (1994)

[Hes 84] W. Hesse: *A systematics of software engineering: Structure, terminology and classification of techniques.* In: P. Pepper (Ed.): *Program Transformations and Programming Environments*, pp. 97-125. Springer 1984

[Hes 95] W. Hesse: Evolutionäre Software-Entwicklung und Projektmanagement. In: *F. Huber-Wäschle, H. Schauer, O. Widmeyer (Eds.): GISI '95 - Herausforderungen eines globalen Informationsverbundes für die Informatik.* Informatik aktuell, Springer 1995

[Hes 96] W. Hesse: Theory and practice of the software process - a field study and its implications for project management; in: *C. Montangero (Ed.): Software Process Technology, 5th European Workshop, EWSPT 96.* Springer LNCS 1149, pp. 241-256 (1996)

[Hes 97a] W. Hesse: *Life cycle models of object-oriented software development methodologies*. In: A. Zendler et al.: *Advanced concepts, life cycle models and tools for object-oriented software development.* Reihe Softwaretechnik 7, Tectum Verlag Marburg 1997

[Hes 97b] W. Hesse: Wie evolutionär sind die objekt-orientierten Analysemethoden? Ein kritischer Vergleich. *Informatik-Spektrum* 20.1, S. 21-28 (1997).

[Hum 89] W. Humphrey: *Managing the Software Process.* Addison-Wesley 1989

[Jac 93] I. Jacobson: *Object-Oriented Software Engineering - A Use Case Driven Approach.* Revised Printing, Addison- Wesley 1993

[Leh 80] M.M. Lehman: Programs, life cycles, and laws of software evolution; *Proceedings of the IEEE,* Vol. 68, No. 9, pp. 1060-1076 (1980)

[M-O 95] J. Martin, J.J. Odell: *Object-oriented methods - A foundation.* Prentice Hall 1995

[Rum 91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: *Object-Oriented Modelling and Design.* Prentice Hall 1991

[ShM 88] S. Shlaer, S.J. Mellor: *Object-Oriented Systems Analysis: Modelling the World in Data.* Yourdon Press 1989

[ShM 91] S. Shlaer, S.J. Mellor: *Object Lifecycles. Modelling the World in States.* Yourdon Press 1991

[ShM 96] S. Shlaer, S.J. Mellor: *The Shlaer-Mellor Method.* http://www.projtech.com/smmethods/smmethod.html, Oct. 1997

[Ste 93] W. Stein: Objektorientierte Analysemethoden - ein Vergleich; *Informatik-Spektrum* 16, S. 317-332 (1993)

[UML 97] G. Booch, J. Rumbaugh, I. Jacobson: *Unified Modeling Language, Notation Guide and UML Semantics, Version 1.0,* Rational Software Corporation, Jan. 1997

[Vmo 97] V-Modell-Browser. http://www.scope.gmd.de/vmodel, Oct. 1997

[WeO 92] F. Weltz, R. Ortmann: *Das Softwareprojekt - Projektmanagement in der Praxis.* Campus-Verlag 1992

Adresse des Autors:

Prof. Dr. Wolfgang Hesse,
FB Mathematik/Informatik, Philipps-Universität
Hans Meerwein-Str.
D-35032 Marburg
Tel.: +49-6421-281515, Fax: +49-6421-285419,
email: hesse@informatik.uni-marburg.de