Wie evolutionär sind die objektorientierten Analysemethoden? Ein kritischer Vergleich

Wolfgang Hesse, Philipps-Universität, Marburg/Lahn

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

Zusammenfassung

Nach M.M. Lehman verläuft Software-Entwicklung häufig nicht geradlinig nach einem starren Phasenschema, sondern evolutionär, d.h. in einer Folge von Erweiterungs- und Anpassungszyklen als Reaktion auf die Erfahrungen der Benutzer. Mit dem Aufkommen der objektorientierten (OO-) Entwicklungsmethoden und deren Anspruch, wiederverwendbare Software zu schaffen, ist der evolutionäre Ansatz erneut ins Blickfeld geraten. Ein Vergleich bekannter OO-Analyse- und Entwurfsmethoden zeigt, daß deren Autoren zwar den evolutionären Charakter der Software-Entwicklung (unterschiedlich stark) betonen, daß dieser jedoch in den Vorgehensmodellen noch kaum zum Ausdruck kommt.

Um diese Lücke zu füllen, wird ein vom Autor entwickeltes Modell (*EOS*, für: *Evolutionäre objektorientierte Software-Entwicklung*) vorgestellt und in den Vergleich einbezogen. Es verzichtet vollständig auf die überkommene Phasenstruktur und knüpft Zyklen und Tätigkeitsschritte 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.

Abstract:

As M.M. Lehman pointed out, software development often does not follow a linear, rigid phase schema, but takes place in evolutionary manner, i.e. in a series of cycles aiming at extending and adapting the system according to the users' feedback. Since object oriented (OO-) development methods came into play, claiming and supporting the reuse of software, the evolutionary approach has again moved into focus. A comparison of known OO analysis and design methods shows that on the one hand side, their authors emphasise the evolutionary character of software development (with different rigor), but on the other hand, this is hardly reflected in their software process models.

In order to fill this gap, a process model for *evolutionary, object-oriented software development (short: EOS)* created by the author is sketched and added to the comparison. It discards the traditional phase structure completely, being based instead on development cycles and activities associated with the „objects“ of software construction: systems, subsystems, components and classes. Management procedures built upon this framework allow to plan and control projects in a more adapted, flexible, and differentiated way.

1 Einleitung: Der evolutionäre Ansatz in der Software-Entwicklung

Seit sich die Softwaretechnik als selbständiges Teilgebiet der Informatik etabliert hat, spielt die Diskussion um den *Software-Lebenszyklus* eine zentrale Rolle. Im Laufe der Zeit ist eine Vielzahl von *Vorgehens-, Phasen oder Prozeßmodellen* entwickelt worden, die den Software-Entwicklungsprozeß paradigmatisch beschreiben und die als Leitfaden für die Abwicklung konkreter Projekte gedacht sind. In der *Software Engineering*-Praxis wird das Feld auch heute noch von den sogenannten *Wasserfall-Modellen* dominiert, die Ende der 70-er Jahre aufkamen und durch die Arbeiten von B. Boehm weithin bekannt wurden. Für diese sind die folgenden Eigenschaften charakteristisch:

- Ein geeignetes Stück Software-Entwicklung wird zum „Projekt“ erklärt, formell ins Leben gerufen und (jedenfalls in solchen Fällen , in denen es zu einem erfolgreichen Ende kommt) auch formell abgeschlossen. Das Projekt wird in „Phasen“ unterteilt, die einzelnen Mitarbeitern oder Teams zur Bearbeitung zugeordnet und von diesen sequentiell abgearbeitet werden.

- Jede Phase beginnt mit einem festgelegten Katalog von Anforderungen oder mit einer Menge von Spezifikationen, die in der oder den vorangegangenen Phasen erarbeitet wurden. Das Ziel einer Phase 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, (re-) integriert, in der Zielumgebung installiert und nach einem Akzeptanztest zur Benutzung freigegeben.

Dieses lineare, phasenbasierte Vorgehensschema hat in der Vergangenheit viel Kritik erfahren:

- Es ist zu grob, unrealistisch und nicht flexibel genug, um mit instabilen und sich ändernden Anforderungen umgehen zu können.

- Es läßt modernere Gesichtspunkte wie objektorientierte Entwurfsmethodik, Wiederverwendung von Software und projektübergreifenden Wissenstransfer außer acht.

- Der Zwang, zu festgelegten Zeitpunkten immer neue Dokumente zu erzeugen, führt zu Redundanzen, Ineffizienzen und schlimmstenfalls zur Demotivation der Entwickler (mögliche Effekte von „Software-Bürokratie“, vgl. [Den 90])

- Zwischen einzelnen Phasen kommt es zu methodischen und terminologischen Brüchen, die sich nachteilig auf den kontinuierlichen Entwicklungsprozeß und die Benutzer-/ Entwickler-Kooperation auswirken.

Gegenvorschläge, wie man die Schwächen des Phasenansatzes überwinden könne, gibt es viele. Zu den wichtigsten alternativen Vorgehensweisen gehören das *prototyping,* die *inkrementelle und die evolutionäre* Systementwicklung.

Beim *prototyping* versucht man das Problem der instabilen Anforderungen dadurch zu entschärfen, daß man anstelle der (nicht-lauffähigen) anfänglichen Anforderungs-Kataloge und Spezifikationen lauffähige Prototypen setzt, mit deren Hilfe sich bestimmte Systemeigenschaften schon frühzeitig erproben (und, falls notwendig, potentiellen Benutzern vorführen) lassen (vgl. [BKM+ 84], [BKK+ 92]). Eines der bekanntesten Vorgehensmodelle, das diesen Ansatz berücksichtigt, ist das Spiralmodell von B. Boehm [Boe 88].

Inkrementelle Systementwicklung bedeutet, daß Anforderungen, Entwürfe und Lösungen sich nicht von Anfang an auf das zu entwickelnde Gesamtsystem beziehen müssen, sondern beginnend mit einem Teil- oder Kernsystem durch schrittweise Erweiterungen (sog. *Inkremente*) gewonnen werden. Schleifen im Entwicklungsprozeß werden immer dann notwendig, wenn Anforderungen sich nachträglich verändern oder wenn durchgeführte Entwicklungsschritte nicht zu zufriedenstellenden Ergebnissen geführt haben. Ein Beispiel für ein Vorgehensmodell, das solche Schleifen nicht nur vorsieht, sondern Entwicklungszyklen zur Grundlage des gesamten Software-Prozesses macht, ist das STEPS-Modell von Ch. Floyd et al. [FRS 89].

Alle genannten Argumente und Ansätze stellen Facetten des sogenannten *evolutionären Ansatzes* dar, der zuerst von M.M. Lehman in die Software Engineering-Diskussion gebracht wurde (vgl. [Leh 80]). Lehman vergleicht drei verschiedene Arten von Aufgabenstellungen miteinander, die in der Praxis der Software-Entwicklung vorkommen: Zum einen solche, für die sich eine vollständige formale Spezifikation angeben läßt und die sich nach dem Paradigma: „Erst spezifizieren - dann programmieren - dann verifizieren “ gut bearbeiten lassen. Dazu gehören z.B. mathematische Problemstellungen wie die Berechnung des kleinsten gemeinschaftlichen Vielfachen oder das Acht-Königinnen-Problem. Die zweite Gruppe umfaßt Problemstellungen aus der realen Welt mit einer Vielzahl von Neben- und Kontextbedingungen, für die es keine absolut „korrekte“ Lösung und daher auch keine vollständige formale Spezifikation geben kann. Ob die gefundene Lösung annehmbar ist, kann nicht bewiesen, sondern nur durch Erprobung und (qualitative) Bewertung bestätigt oder verworfen werden. Beispiele dafür sind Wettervorhersage- oder schachspielende Programme.

Diesen stellt Lehman die Aufgabenstellung für sogenannte „sozial eingebettete“ Systeme gegenüber, bei denen es darum geht, menschliche oder gesellschaftliche Aktivitäten zu automatisieren. Hier ist die Ausgangssituation wieder eine andere: Die Anforderungen sind häufig nicht klar, müssen zwischen Auftraggeber und Auftragnehmer erst noch ausgehandelt werden und sind möglicherweise während des Projektablaufs Änderungen unterworfen. Die erstellten Systeme verändern das Arbeits- oder Erlebnisumfeld ihrer Benutzer, sind deshalb erhöhter Kritik ausgesetzt und führen in der Regel zu Rückkopplungen, etwa in Form von Änderungswünschen, neuen oder veränderten Anforderungen, die Auslöser für weitere Entwicklungszyklen sein können. Beispiele solcher Systeme sind Steuerungs- und Kontroll-systeme (etwa im Luftverkehr), Lagerhaltungssysteme, Computer-Betriebs- und -Ober-flächensysteme sowie Software-Entwicklungswerkzeuge. Evolutionäre Systeme unterliegen gewissen Gesetzmäßigkeiten, die Lehman über Jahre hinweg studiert hat [Leh 96].

Eine vom Autor zusammen mit Arbeitspsychologen und Soziologen im Rahmen des IPAS-Projekts durchgeführte, zweiteilige Praxisuntersuchung von insgesamt fast 50 Software-Projekten hat u.a. erbracht, daß gerade in der Frage des Vorgehensmodells und der tatsächlichen Projektabwicklung Theorie und Praxis erheblich voneinander abweichen (vgl. [W-O 92], [H-W 94]). So waren zwar wasserfallartige Phasenmodelle in der Mehrzahl (61 %) der Projekte der ersten Untersuchung noch offizieller Standard, die Phasenplanung wurde jedoch nur in 24 % der Fälle auch wirklich eingehalten. Bei 41 % der Projekte wurden Phasen-Überlappungen, bei 13 % „Echternacher Springprozession“ (d.h. explizite Rücksprünge im Projektablauf), bei 9 % „informaler Ablauf“ und bei 13 % sogar „anarchisch“ als Vorgehensart angegeben. Bei einer weiteren Untersuchung, wurde in 29 Projekten von insgesamt 39 expliziten Schleifen (davon 11 sogar über 2 Projektabschnitte hinweg) berichtet (vgl. [BHS 95], [Hes 96c]).

Weiter haben unsere Untersuchungen gezeigt, daß Software-Entwickler von sich aus Vorkehrungen treffen, die die Langlebigkeit, Änder- und Erweiterbarkeit ihrer Produkte fördern, daß diese bei der Projektplanung und -bewertung aber kaum angemessen berücksichtigt werden. (vgl. [BHS 95]).

Als Fazit dieser Untersuchungen können wir also feststellen, daß die Praxis der Software-Entwicklung weit „evolutionärer“ ist als die offiziellen Vorgehensmodelle. Ursachen für das Festhalten an überkommenen, sequentiellen Projektstrukturen liegen vor allem in dem Bestreben des Projekt-Managements, Projekte in leicht planbare, überschaubare und kontrollierbare Abschnitte einzuteilen.

In den letzten Jahren sind viele Methoden zur *objektorientierten (kurz: OO-) Software-Entwicklung* vorgeschlagen worden, einige davon haben Eingang in die industrielle Praxis gefunden. Damit hat sich die Frage nach speziellen Vorgehensmodellen für die OO-Software-Entwicklung neu gestellt. Diese sollten aus folgenden Gründen „evolutionäre“ Elemente enthalten:

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

- Folgt die Software-Entwicklung einem einheitlichen, durchgängigen „Objekt-Paradigma“, so verwischen sich die Phasengrenzen und die Projektplanung muß sich mehr an der Software-Architektur und den an einzelnen Bausteinen auszuführenden Aktivitäten als an systemweit definierten Projektphasen orientieren.

In einer kürzlich fertiggestellten Studie (vgl. [Hes 96b]) hat der Autor unter anderem vier bekannte OO-Analyse- und Entwurfsmethoden daraufhin untersucht, inwieweit die dort vorgeschlagenen Vorgehensmodelle die Anforderungen an eine evolutionäre Entwicklung unterstützen. Es handel t sich um die Methoden OOSA von S. Shlaer & S. Mellor, OMT von J. Rumbaugh, OOAD von G. Booch und OOSE von I. Jacobson.

Im 2. Abschnitt folgt ein Vergleich der Vorgehensmodelle dieser Methoden in Bezug auf die darin enthaltenen evolutionären Elemente. Im 3. Abschnitt werden diese dem vom Autor entwickelten EOS-Modell gegenübergestellt. Im abschließenden 4. Abschnitt werden Schlußfolgerungen im Hinblick auf das Management evolutionärer Software-Entwicklung gezogen.

2 Ein Vergleich bekannter OO-Vorgehensmodelle im Hinblick auf evolutionäre Software-Entwickung

In diesem Abschnitt sollen die Vorgehensmodelle der oben angeführten OO-Methoden speziell unter dem Aspekt beleuchtet werden, ob und inwieweit sie den Zielsetzungen der evolutionären Software-Entwicklung Rechnung tragen. Damit ist *kein* umfassender Vergleich dieser Methoden zu verbunden, dafür sei auf die zahlreichen diesbezüglichen Veröffentlichungen verwiesen, z.B. auf [M-O 95] oder im deutschsprachigen Bereich auf [Ste 93].

Für das hier gestellte Thema ergeben sich die folgenden Fragen:

- Wie ist das Vorgehensmodell als Ganzes aufgebaut, ist es linear, zyklisch, hierarchisch?

- Ist der Aspekt der Wiederverwendbarkeit im Vorgehensmodell berücksichtigt?

- Wie stellen sich die betreffenden Autoren zum evolutionären Ansatz, inwieweit sind etwa iteratives, inkrementelles Vorgehen oder Prototyping vorgesehen?

Diese Fragen sollen als Leitfragen für die folgenden Abschnitte dienen. Für einen ausführlicheren Vergleich, der eine Reihe weiterer Kriterien berücksichtigt, sei auf [Hes 96b] verwiesen. Dort wurden die oben genannten OO-Methoden 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.*

Gesamt-Modellstruktur

S. Shlaer und S. Mellor haben ihre Methode *Object Oriented Systems Analysis (OOSA)* in zwei Büchern ([SM 89] und [SM 91]) publiziert. Das erste Buch enthält ein sequentielles Phasenmodell mit vier Phasen:

- Problem-Analyse (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 ihrer Aufteilung des Analysemodells in die drei Teilmodelle *Informationsmodell, Zustandsmodell und Prozeßmodell* nehmen die Autoren den traditionellen Standpunkt dreier komplementärer, aber eigenständiger Teilmodelle ein. 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 umfaßt. 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 zweiten Buch [S-M 91] wird eine Hierarchie von sieben Entwicklungsebenen eingeführt, es wird aber kein neues Vorgehensmodell darauf aufgebaut.

Rumbaugh et al. schlagen in ihrem Buch *Object-Oriented Modeling and Design* ein sechsphasiges Wasserfall-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)

Von diesen sind allerdings nur die Analyse- und Entwurfsphasen ausführlich, die letzten beiden Phasen dagegen überhaupt nicht beschrieben. Die Phasen des Vorgehensmodells werden von den Autoren ausdrücklich als weniger wichtig - verglichen mit den traditionellen Modellen - 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.

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 Entwurf sphase *(object design)* konzentriert. Sie werden hauptsächlich auf der zweiten Abstraktionsebene sichtbar, wenn die Phasen in Schritte und Aktivitäten zergliedert werden. Hier wird als wesentliches Merkmal der Objektorientierung die Präferenz *„Datenstruktur vor Funktion“* deutlich. Das Objektkonzept zieht sich als führendes, einheitliches Leitkonzept durch den gesamten Entwicklungsprozeß: Ihm sind alle anderen Konzepte wie Funktionen, Beziehungen, Ereignisse, Zustände, etc. zu- und untergeordnet.

G. Booch hat seine Methode zum OO-Entwurf erstmals 1991 in einem zusammenhängenden Buch [Boo 91] beschrieben 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 (*micro and macro process* [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]. Der Makro-Prozeß ähnelt dagegen mehr den traditionellen Wasserfall-Modellen. Er bildet den steuernden Rahmen für die darin enthaltenen Mikro-Prozesse.

Ein *Mikro-Entwicklungsprozeß* besteht aus 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. 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.

Dagegen ähnelt der Makro-Entwicklungsprozeß mit seinen fünf Phasen

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

den traditionellen Wasserfall-Modellen. Er weist die folgenden Besonderheiten auf: (1) Die „Konzeptualisierung" (conceptualization) wird als als separate Phase eingeführt und damit ihre besondere Bedeutung für die Festlegung der Kern-Anforderungen hervorgehoben. Dazu gehört es z.B. mit - möglicherweise nicht-weiterverwendbaren - Prototypen zu experimentieren. (2) Anstelle der üblichen „Implementierungsphase“ steht bei Booch „Evolution“. Damit wird zum einen eine terminologische Überschneidung mit dem Mikro-Prozeß vermieden, zum anderen aber auch der evolutionäre Charakter dieser Phase hervorgehoben. (3) 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].

Der Makro-Prozeß beschreibt die Aktivitäten eines Entwicklungs-Teams über Zeiträume von Wochen und Monate. Hierhin gehören z.B. auch das Konfigurations-Management, die Qualitätssicherung, Review- und Inspektions-Maßnahmen sowie die Dokumentation. Der Schwerpunkt liegt beim Risiko-Management und bei der Entwicklung der Gesamt-Architektur. Für die eingebetteten Mikro-Prozesse übernimmt der Makro-Prozeß die Steuerungsfunktion.

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, Klassen und Objekte zu identifizieren, zu spezifizieren und zu implementieren. Beide Prozesse müssen miteinander verknüpft werden, um einen vollständig rationalen ("fully rational") Entwicklungsprozeß zu erreichen. Dies ist nach Booch eine Voraussetzung für die Erreichung eines Qualitätsstandards, der etwa dem Prädikat "defined" im Software Capability Maturity (SCM) -Modell des amerikanischen Software Engineering Instituts (SEI) entspricht (vgl. [Hum 89]).

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 methodisch-technischen Aspekte stärker von den Management-Aspeten zu trennen und führt deshalb nebeneinander ein Prozeßmodell und ein 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 eine 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 Management-gemäße Phaseneinteilung und die Definition von Meilensteinen. 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.

Fazit: 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 einer ersten Gruppe von Autoren (Shlaer/Mellor und Rumbaugh) dominiert die Wasserfall-Struktur. Spezifika der objektorientierten und der evolutionären Entwicklung haben (abgesehen von einigen terminologischen Besonderheiten) keinen Einfluß auf die überkommene sequentielle Phasenstruktur. 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. Dies wird auch an der relativ knappen, tabellarischen Darstellung des Vorgehensmodells deutlich, die zudem auf die Analyse- und Entwurfsphasen beschränkt bleibt.

Bei Booch finden wir ebenfalls den erwähnten hybriden Ansatz (und hier sogar in ganz expliziter Form), aber zugleich 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. Zwischen dem Makro- und Mikro-Zykus wird eine gewisse Orthogonalität angestrebt, sie ist allerdings nicht durchgängig: Analysierende, entwerfende und implementierende Tätigkeits-Elemente finden sich in beiden Zyklen und für die ganz frühen und späten Makro-Phasen machen die Mikro-Zyklen keinen Sinn.

Obwohl auch Jacobson zwei ineinander eingebettete Zyklen benutzt, unterscheiden sich die Modelle der beiden Autoren ganz erheblich voneinander. Bei Jacobson handelt es sich nicht (wie bei Booch) um eine Hierarchie von Modellen, sondern seine Modelle 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.

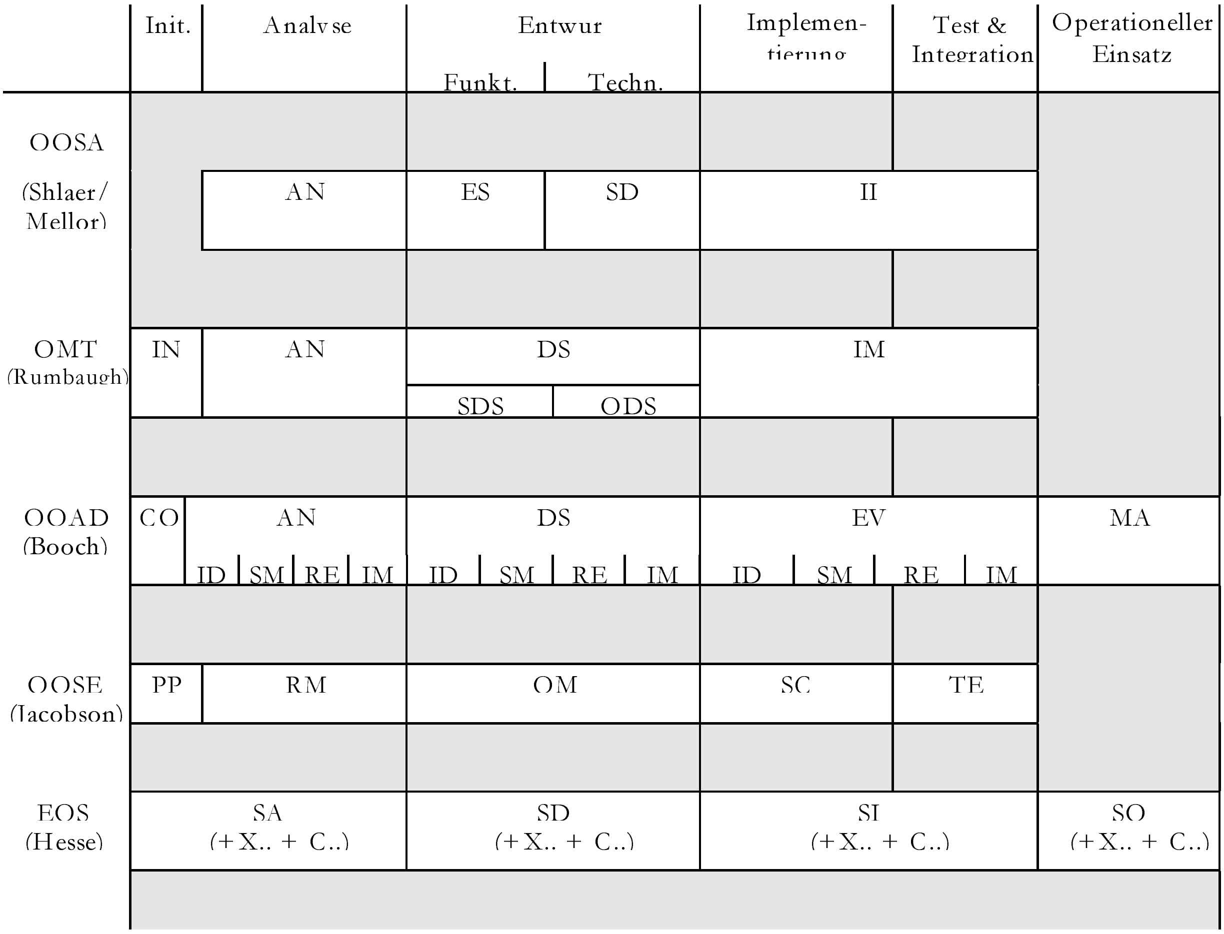

Abb. 1: Phasenstruktur ausgewählter Vorgehensmodelle

Eine Übersicht über die Phasenstrukturen liefert Abb.1. Sie schließt das weiter unten erläuterte EOS-Modell bereits mit ein.

Wiederverwendung und System-Evolution

Die Idee der evolutionären Software-Entwicklung ist untrennbar mit der Frage nach der Wiederverwendung von Software-Bausteinen und -Dokumenten verbunden. Bedeutet Evolution doch gerade, Vorhandenes weiterzuentwickeln, auf Bestehendem aufzubauen, Neues aus Bekanntem abzuleiten. Wir können daher die Eignung von Methoden zum evolutionären Vorgehen auch daraus ableiten, welche Unterstützung sie für die Software-Wiederverwendung bieten.

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.

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 dies allerdings nicht vor. Die zweite Referenz ist ein eigenes Teilkapitel über Wiederverwendbarkeit unter der generellen Überschrift „programming style“. Damit bleibt das Thema über 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 widmet der Wiederverwendung einen eigenen Abschnitt in seinem Buch. 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 „Mustersammeln“ (*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.

Unterstützung des evolutionären Vorgehens im Vorgehensmodell

Zum Schluß dieses Vergleichs soll ein Resümee darüber gezogen werden, wie sich die Autoren zum evolutionären Ansatz stellen und welche Elemente ihrer Vorgehensmodelle zur Unterstützung einer evolutionären Vorgehensweise herangezogen weden können. Zu diesen Elementen gehören z.B. auch iteratives, inkrementelles Vorgehen und *prototyping*.

Shlaer & Mellor geben in ihren beiden Büchern kaum Hinweise zu den genannten Themen. So werden evolutionäre Aspekte der Software-Entwicklung im ersten Buch gar nicht und im zweiten nur sehr eingeschränkt - bezogen auf „Objekt-Lebenszyklen“ behandelt. Entwicklungszyklen werden zwar nicht ausgeschlossen, sind aber im Vorgehensmodell nirgendwo explizit angesprochen und es bleibt unklar, wie sie sich in die sequentielle Phasenstruktur einfügen könnten.

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 während der Analysephase 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 das Buch sich 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.

Beim Makro-Prozeß benutzt Booch explizit den Begriff „Evolution“ für eine ganze Phase (an Stelle der sonst üblichen „Implementierungsphase“). Damit soll die oben erwähnte inkrementelle Vorgehensweise in dieser Phase betont werden. Dazu gehört es ebenfalls, Alternativ-Entwürfe probeweise umzusetzen, aus mehreren konkurrierenden Lösungen die beste auszuwählen oder eine erfolgte Implementierung durch Erweiterungen und Änderungen fortzusetzen. Wir können dies auch als „kurzfristige Evolution“ bezeichnen im Gegensatz zu M.M. Lehmans Evolution, die eher langfristig angelegt ist und auch mehrere hintereinandergeschaltete Makro-Zyklen zulassen würde. Bei Booch fällt das unter den Begriff *modification*. Er zitiert Lehman in diesem Zusammenhang, sieht aber keine großen Unterschiede zwischen diesen verschiedenen Formen von Evolution. 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-Aspekt (= Software-Evolution) wesentlich stärker als das isolierte Erzeugen eines bestimmten Produkts aufgrund festgelegter Vorgaben. Damit sind die Ideen von Software-Evolution und inkrementeller Entwicklung fundamental für den gesamten Ansatz. Anwendungs-Fallstudien *(use cases)* können als Ausgangspunkt für *System-Inkremente* dienen. Jedes Inkrement bringt neue Funktionalität in das System und bildet damit ein neues Entwicklungsstadium. 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. Jacobson bevorzugt weiterverwendbare Prototypen und äußert sich skeptisch über das sogenannte *rapid prototyping,* das er eher als ein gern genutztes Feigenblatt für *„quick and dirty programming“* ansieht.

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

Als ein Resultat des eingangs erwähnten IPAS-Projekts wurde vom Autor eine Methode zur evolutionären, objektorientierten Software-Entwicklung (EOS) entwickelt und an anderer Stelle publiziert (vgl. [H-W 94], [Hes 95]). Drei Argumente haben den Ausschlag dafür gegeben, mit EOS ein gänzlich neues Vorgehensmodell vorzuschlagen:

- Beim Vorgehen in der Software-Entwicklung haben sich Theorie und Praxis weit voneinander entfernt und sollten wieder besser in Einklang gebracht werden.

- Die Prinzipien der Objektorientierung sind zwar zumeist in den Methodenbeschreibungen gut wiedergegeben, haben jedoch die Vorgehensmodelle erst wenig beeinflußt. Dies belegen die nach wie vor dominierenden Wasserfall-Strukturen.

- Der evolutionäre Charakter der Software-Entwicklung wird bei den meisten Methoden noch zu wenig gewürdigt. Dies betrifft besonders die Vorgehensmodelle. Ein Wasserfall-Modell zu einem Zyklus „zusammenzubiegen“ reicht nicht aus, um ein evolutionäres Vorgehensmodell zu bilden.

Das EOS-Vorgehensmodell beruht auf den folgenden Leitgedanken:

- 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 Entwicklungszyklus ist 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.

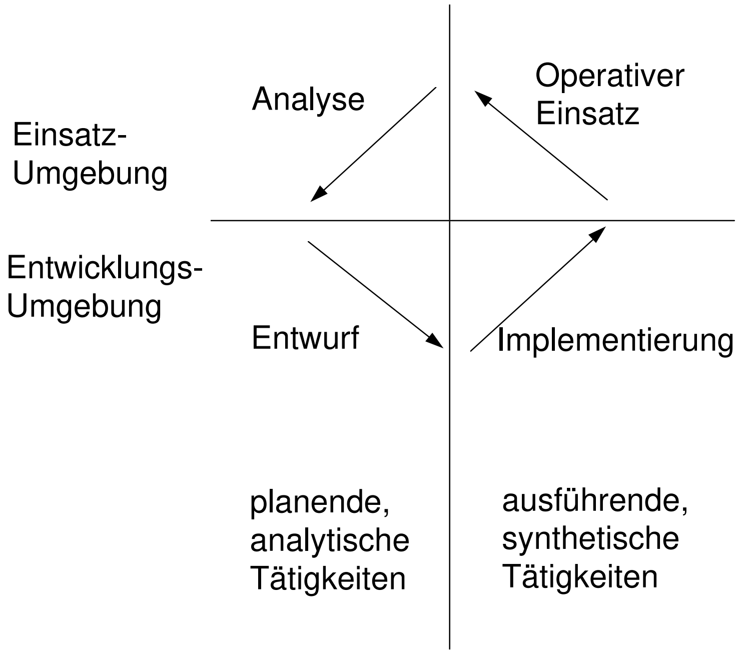

Abb.2: Struktur eines EOS-Entwicklungszyklus

- Entwicklungszyklen für einzelne Systemteile und Bausteine sind in rekursiver (fraktaler) Weise miteinander gekoppelt. 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. Das sind vom Management zu planende Zeitpunkte im Projekt, für die individuell für jeden Baustein der jeweilige Entwicklungszustand geplant und überprüft werden kann. Der Gesamtstatus des Projekts ergibt sich aus der Zusammenfassung dieser Entwicklungszustände.

Vergleich des EOS-Modells mit den oben genannten Modellen

Alle im vorigen Abschnitt betrachteten Vorgehensmodelle verfolgen einen hybriden Ansatz, bei dem objektorientierte Entwicklungsprozesse („im Kleinen“) in ein übergeordnetes Wasserfall-Modell („im Großen“) eingebettet sind. Dagegen wird im EOS-Modell die Idee hierarchisch gestaffelter Entwicklungszyklen konsequent verfolgt. Höchste Priorität genießen nicht die Phasen des Wasserfall-Modells, sondern die Bausteine der Systemstruktur, denen einzelne Tätigkeiten und Entwicklungsschritte zugeordnet sind.

Im Unterschied zu Booch’s Modell umfaßt die Hierarchie nicht nur zwei Ebenen, sondern Entwicklungszyklen können beliebig tief gestaffelt werden. Weiterhin haben alle Zyklen die gleiche Grundstruktur und werden rekursiv definiert. 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.

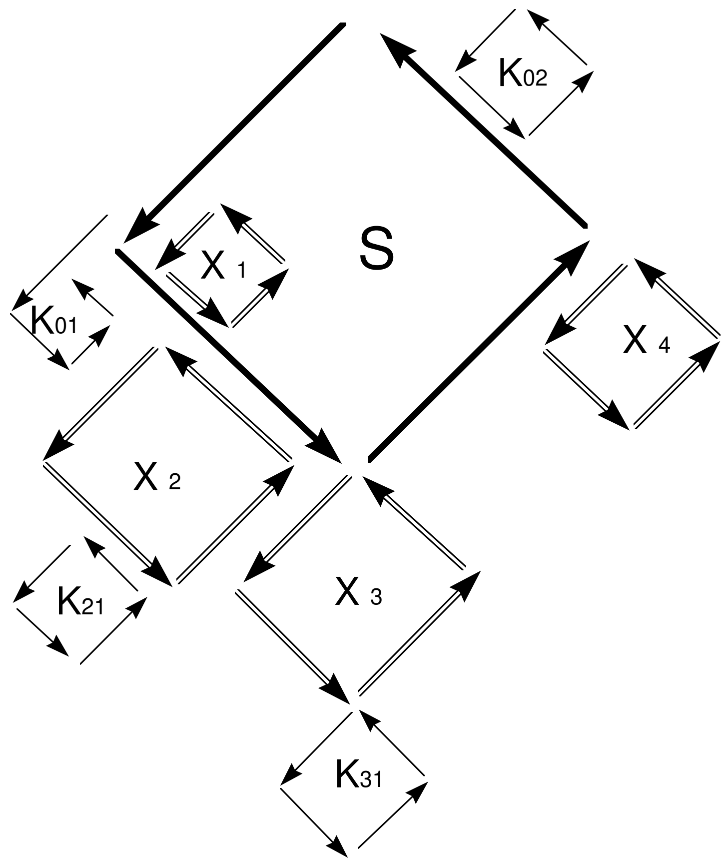

Abb.3: Zeitlich verzahnte Entwicklungszyklen

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.

Evolutionäre Systementwicklung gehört - wie der Name bereits andeutet - zu den obersten Zielen des EOS-Modells. Jeder Entwicklungsprozeß ist als Zyklus angelegt - was natürlich nicht ausschließt, daß Zyklen irgendwann auch einmal enden. Inkrementelle Entwicklung ist ebenfalls möglich: Inkremente werden in naheliegender Weise als Komponenten oder Subsysteme definiert. Ähnliches gilt für die Entwicklung von Prototypen. Diese verläuft typischerweise zyklisch, wobei die Entwurfs- und Implementierungsschritte möglicherweise verkürzt sind.

4 Ausblick: Projekt- Management für evolutionäre Software-Entwicklung

Eines der hervorstechenden IPAS-Ergebnisse ist die Feststellung, daß formale Planung und Projekterfolg nicht aneinander gekoppelt sind. 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. [W-O 92], [H-W 94]).

Daraus leiten sich Forderungen nach einem flexiblen, dynamischen Projekt-Management ab. Eine Projektplanung sollte vor allem realistisch und damit an veränderte Bedingungen und Situationen anpaßbar sein. So ist es angemessener, ehrlicher und am Ende auch genauer und kostengünstiger, von vornherein differenziert, aber flexibel zu planen, mögliche Plan-Anpassungen vorzusehen und die Erfordernisse evolutionärer Entwicklung bei der Planung von Anfang an zu berücksichtigen. Neue Qualitätsstandards wie das SCM des SEI (vgl. Abschnitt 2) verlangen ebenfalls solch ein flexibles Management: nur so können die höheren Stufen "managed" oder "optimising" erreicht werden.

Es ist klar, daß ein evolutionäres Vorgehen das Planen, Steuern und Kontrollieren von Software-Prozessen nicht einfacher macht. An die Stelle der simplen, überschaubaren Phasenstruktur des Wasserfall-Modells treten komplexere und differenziertere Zyklen- und Tätigkeitsstrukturen, die entsprechende Management-Techniken erfordern. Nur so läßt sich auch das Festhalten des Managements an den überkommenen Phasenmodellen erklären: es erleichtert scheinbar die Projektplanung und schiebt die Probleme auf einen späteren Zeitpunkt. Diese Strategie hat sich jedoch als kurzsichtig, wenig tragfähig und für die Qualität nicht förderlich erwiesen, oft führte sie später zu erheblichen Konflikten, Qualitäts-Einbußen oder kostspieligen Abweichungen von der ursprünglichen Planung.

Projekt-Management für *evolutionäre System-Entwicklung* stellt von Projektbeginn an besondere Anforderungen: Es beginnt nicht mit einem fertigen, vordefinierten, formalen Plan, sondern mit einem vorläufigen, der ständig verfeinert und den aktuellen Projekterfordernissen angepaßt werden muß. Anstelle eines einmaligen „Absegnens“ der Projektbedingungen zwischen Auftraggeber und Auftragnehmer tritt ein kontinuierlicher Kommunikationsprozeß, der auch Angleichungen der Planung an veränderte Anforderungen und Rahmenbedingungen erlauben muß.

Dafür liefert das EOS-Modell die notwendigen Grundstrukturen.

- 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, für Klassen einzelne Entwickler zuständig. Anwendungsnahe Komponenten können von gemischten Teams (d.h. mit Beteiligung der Benutzer) entwickelt werden. Die Bindung einzelner Teams und Entwickler an „ihre“ Bausteine ist eine wesentliche Voraussetzung für deren „evolutionäre“ Entwicklung.

- EOS-Revisionspunkte sind mit den Referenzlinien des STEPS-Modells verwandt [FRS 89]. Sie sind allerdings stärker an der System- /Bausteinstruktur und an den vorgegebenen Tätigkeiten der Entwicklungszyklen verankert. Dadurch 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-/Aktivitäten-Struktur an. Wichtige Reviews werden mit den Revisionspunkten kombiniert. Ihre Ergebnisse bestimmen die weitere Zyklen-Planung und die inhaltliche Festlegung künftiger Revisionspunkte.

- Software-Entwicklung wird als *kontinuierlicher, evolutionäerer Prozeß* angesehen, der im allgemeinen nicht mit der Installation eines „fertigen“ Produkts endet. Vielmehr gehen die Entwicklungstätigkeiten weiter - sei es als „Wartung“, als Erweiterung oder Weiterentwicklung, sei es als Wissenstransfer zu weiteren benachbarten Projekten. Das EOS-Modell bildet die Grundlage für solche Transferprozesse: von Tätigkeit zu Tätigkeit, von Zyklus zu Zyklus, von Projekt zu Projekt.

Wie sich der evolutionäre Ansatz langfristig auf die Vorgehensmodelle auswirken wird, läßt sich zur Zeit noch schwer absehen. Auf methodischem Gebiet bahnt sich mit der *„unified method“,* an der Rumbaugh, Booch und Jacobson zur Zeit gemeinsam arbeiten, eine gewisse Vereinheitlichung der Notation und Terminologie an. In jüngster Zeit lassen die Verlautbarungen der Autoren eine Konzentration auf eine „Unified Modelling Language“ erkennen. Das würde bedeuten, daß ein „unified process“, d.h. ein gemeinsames Vorgehens-modell zunächst nicht und möglicherweise überhaupt nicht angeboten werden soll. Damit endet ein wichtiger gegenwärtiger Versuch zur Standardisierung bei den Konzepten und Notationen und es bleibt offen, ob diese die Vorgehensmodelle jemals mit erfassen wird.

Literaturverweise:

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

[BKM+ 84] R. Budde, K. Kuhlenkamp, L. Mathiassen, H. Züllighoven (eds.) Approaches to prototyping, Berlin, Heidelberg, New York: Springer 1984:.

[BKK+ 92] R. Budde, K. Kautz, K. Kuhlenkamp, H. Züllighoven: Prototyping - an approach to evolutionary system development; Berlin, Heidelberg, New York: Springer1992.

[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

[Den 90] E. Denert: Software Engineering - Methodische Projektabwicklung; Springer 1990

[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

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

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

[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 96a] W. Hesse: From WOON to EOS: New development methods require a new software processs model; Technical Report no. 12/96, Fachbereich Mathematik, Philipps-Universität Marburg

[Hes 96b] W. Hesse: Life cycle models of object-oriented software development methodologies; Technischer Bericht Nr. 96-04, FAST e.V., München 1996

[Hes 96c] 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)

[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)

[Leh 96] M.M. Lehman: Laws of software evolution revisited; in: C. Montangero (Ed.): Software Process Technology, 5th European Workshop, EWSPT 96; Springer LNCS 1149, pp. 241-256 (1996)

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

[Mey 88] B. Meyer: Object-oriented software construction; Prentice Hall 1988

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

[Rum 96] J. Rumbaugh: To form a more perfect union: Unifying the OMT and Booch methods, JOOP Vol 8, No. 8 , pp. 14-18 (Jan. 1996)

[S-M 89] S. Shlaer, S.J. Mellor: Object-Oriented Systems Analysis: Modelling the World in Data; Yourdon Press 1989

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

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

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