Heft 78

Helge Heß

Vergleich von Methoden zum objektorientierten Design von Softwaresystemen

August 1991

1 Einleitung

Eine Vielzahl von objektorientierten Programmiersprachen ist mittlerweile auf einer umfangreichen Palette von Hardwareplattformen effizient einsetzbar und hat ihre grundsätzliche Fähigkeit bewiesen, die Produktivität und Qualität der Softwareentwicklung unter gewissen Voraussetzungen erheblich zu steigern.
Obwohl beim objektorientierten Lifecycle eines Softwaresystems die Grenzen zwischen Analyse, Design und Implementierung verwischt sind (vgl. Abschnitt 2), ist die Existenz von Methoden, die eine Beschreibung eines Systems auf einer höheren abstrakteren Ebene als der der Programmiersprache erlauben, zwingend notwendig, um große Anwendungen in planvoller Art und Weise zu realisieren.
Die häufig beklagte, kaum noch zu überblickende Methodenvielfalt zur Analyse und zum Design von Softwaresystemen[1] scheint sich auch im objektorientierten Bereich fortzusetzen, wo in den letzten Jahren eine ganze Reihe von Methoden vorgeschlagen wurden, die allerdings noch nicht so ausgereift sind, wie die zum Quasi-Standard erhobenen Methoden traditioneller Entwicklung. Ungeachtet der damit verbundenen Ausweitung der Methodenvielfalt rechtfertigt und fordert die der Objektorientierung eigene Sichtweise jedoch die Definition neuer, geeigneter Methoden, um die Darstellung eines Systems als miteinander kommunizierende Objekte schon in den frühesten Phasen der Entwicklung zum Ausdruck zu bringen. Auch wenn sich einige der Basiskonzepte objektorientierter Programmierung in Verfahren zur Daten- bzw. Funktionsmodellierung wiederfinden[2], sind lobenswerte Versuche, die Ergebnisse konventioneller Designmethoden wie z. B. des Structured Designs als Ausgangspunkt einer Transformation hin zu objektorientiertem Design zu verwenden, lediglich als sinnvoll anzusehen, um schon für Analyse und Design eines Systems geleisteten Aufwand zu sichern; für Neuentwicklungen ist diese Vorgehensweise nicht geeignet, die von der Objektorientierung erwartete einfache Abbildung der Realität zu leisten.[3]

Nachfolgend werden existierende objektorientierte Designmethoden vorgestellt sowie ihre Ursprünge aufgezeigt, um dann aufbauend auf einer vergleichenden Bewertung Prognosen für die zukünftige Bedeutung derselben abzuleiten und Hinweise zur Vermeidung einer möglicherweise unnötigen Ausweitung der Methodenvielfalt zu geben. Auf die Erläuterung objektorientierter Basiskonzepte und die zugehörige Terminologie wird im

---

[1] Vgl. Scheer, A.-W.: Architektur integrierter Informationssysteme. Berlin et al. 1991, S. 1ff.

[2] Vgl. Scheer, A.-W.: a.a.O., S. 123.

[3] Vgl. PANEL: Structured Analysis and Object Oriented Analysis, in: Proceedings of ECOOP '90, S. 135-139.

folgenden nicht näher eingegangen, hier wird auf die entsprechende Grundlagenliteratur verwiesen.[4,5]

2 Lifecycle objektorientierter Softwaresysteme

Um die Aufgaben und Bedeutung objektorientierter Analyse- und Designmethoden beurteilen zu können, werden zunächst kurz die existierenden Ansätze zum objektorientierten Software-Lifecycle vorgestellt:
Das klassische **Wasserfall-Modell**[6] beschreibt grundsätzlich eine vernünftige Abfolge der Phasen einer Softwareentwicklung. Die seit langem kritisierten Mängel beruhen im wesentlichen auf der Starrheit und Inflexibilität dieser Vorgehensweise.[7] Neben Verbesserungen im Sinne eines evolutionären Prototypings, um sich ändernde Anforderungen in einer wohldefinierten Weise auch in spätere Phasen der Produktentwicklung einfließen zu lassen, versuchen objektorientierte Lifecycle-Modelle insbesondere, den monolithischen Ansatz eines traditionellen Vorgehensmodells aufzubrechen. Wiederverwendbarkeit von Teilkomponenten, leichte Änder- und Erweiterbarkeit als die immer wieder postulierten Vorteile objektorientierter Entwicklung können nur dann erreicht werden, wenn nicht nur das eine zu entwickelnde Produkt Endziel der Entwicklung ist, sondern zum einen wiederverwendbare Komponenten ganz bewußt als Ergebnis jeder Entwicklung angestrebt werden und zum anderen natürlich auch schon existierende Design- und Codebausteine in die Entwicklung einfließen.
Das von Meyer vorgeschlagene **Cluster-Modell** zielt darauf ab, für die Funktionalität eines Systems notwendige, zusammengehörige Klassen zu Clustern zusammenzufassen, die nicht nur Relevanz für das aktuelle Projekt besitzen, sondern projektübergreifende Generalisierungsschritte durchlaufen und langfristig eine Bottom-Up-orientierte Entwicklung unterstützen.[8] Die möglicherweise parallele Entwicklung jedes dieser Cluster orientiert sich an einem modifizierten Wasserfall-Modell, wobei die bisherigen Phasen zu den drei Entwicklungsschritten SPEC (specification), DESIMPL (design and

---

[4] Vgl. Schlüter, P. et al.: Objektorientierte Software-Entwicklung: Konzepte und Terminologie, in: Softwaretechnik-Trends, 10 (1990) 2, S. 22-44.

[5] Vgl. Meyer, B.: Object-Oriented Software Construction. Hemel Hempstead 1988.

6 Vgl. Boehm, B.W.: Software Engineering Economics. Englewood Cliffs 1981.

[7] Vgl. Fleischer, P. et al.: Der objektorientierte Software-Entwicklungsprozeß und seine Unterstützung durch Werkzeuge, in: Softwaretechnik-Trends, 11 (1991) 1, S. 23-53.

[8] Vgl. Meyer, B.: From Structured Programming to Object-Oriented Design: The Road to Eiffel, in: Structured programming, 10 (1989) 1, S. 19-39.

implementation) und VALGEN (validation and generalization) zusammengefaßt werden (vgl. Abb. 1). Die Reihenfolge der Clusterentwicklung ist vom allgemeinen zum speziellen hin gerichtet, da die spezielleren Klassen sich der allgemeineren Klassen bedienen, die in Bausteinbibliotheken abgelegt sind und von jedem neu durchzuführenden Projekt durch eventuelle Verallgemeinerungen profitieren.

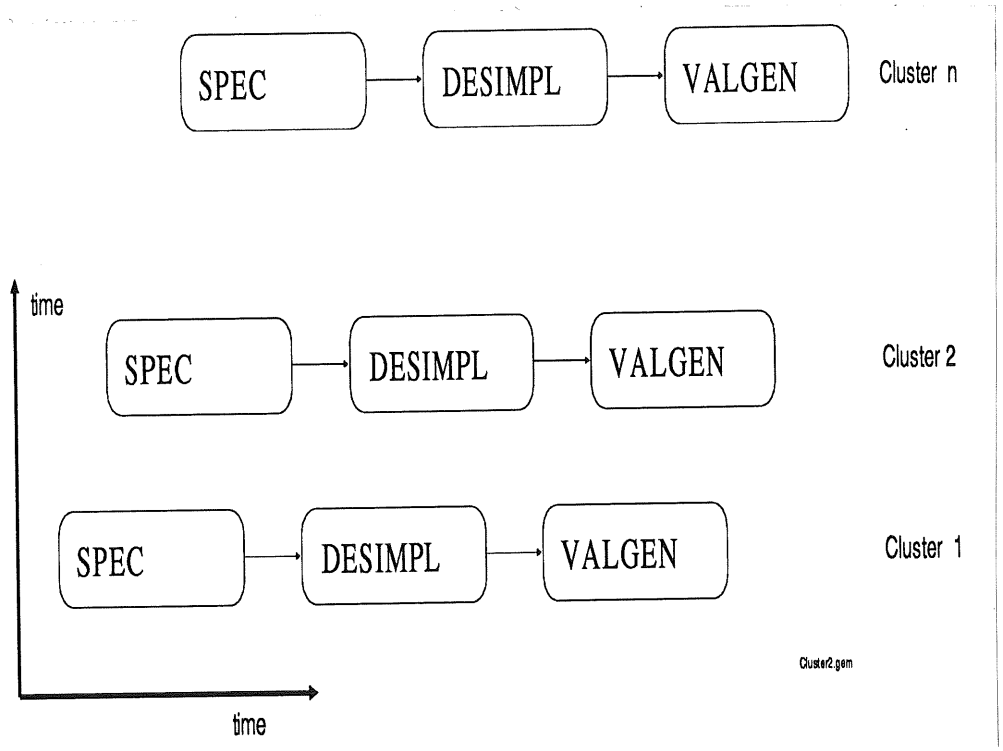


Abb. 1: Cluster-Modell (Quelle: Meyer, B., 1989)

Die geänderte Zielsetzung objektorientierter Softwareentwicklung spiegelt sich auch im **Fontänen-Modell** von Henderson-Sellers und Edwards wider (vgl. Abb. 2):[9] Neben der hohen Überdeckung der einzelnen Phasen besteht die wesentliche Neuerung darin, daß jede Phase mit dem Ziel angesteuert wird, nicht das erreichte Zwischenziel zu konservieren, sondern sofort wieder mit den anfänglich definierten Anforderungen abzugleichen und den gesamten Prozeß auf der Grundlage dieser Erfahrungen von vorne zu beginnen. Die Folge ist eine prototypähnliche Entwicklung mit hoher Iterationsanzahl,

9 Vgl. Henderson-Sellers, B.; Edwards, J.M.: The Object-Oriented Systems Life Cyle, in: Communications of the ACM, 33 (1990) 9, S. 142-159.

die in sinnvoller Weise auch auf die Entwicklung der von Meyer definierten Cluster anzuwenden ist. Dieser Lebenszyklus erfährt auch explizite Anwendung beim Entwurf autonomer Bausteine, die mit dem Ziel der Wiederverwendung in einer Bibliothek abgelegt werden.

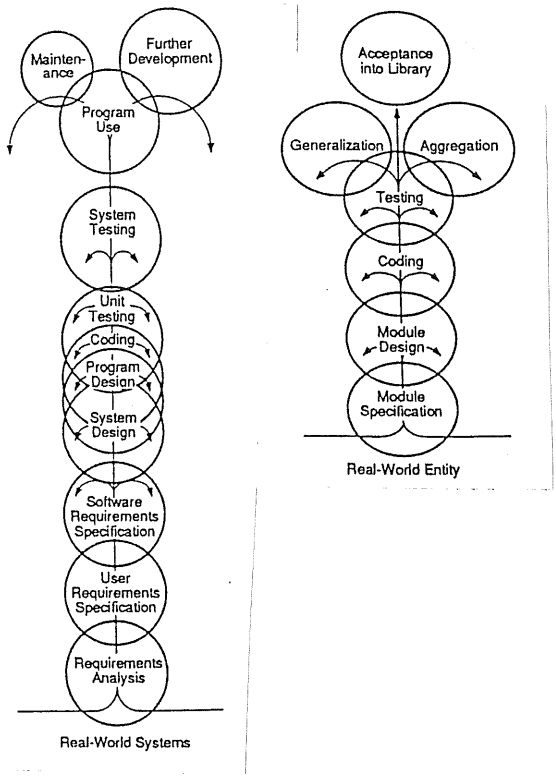


Abb. 2: Fontänen-Modell (Quelle: Henderson-Sellers, Edwards, 1990)

Im Hinblick auf die Rollen von Analyse und Design im objektorientierten Lifecycle wird aus diesen beiden Ansätzen deutlich, daß die traditionellen Aufgaben des Softwareentwicklers eine Veränderung erfahren:

Wenn Wiederverwendbarkeit als zentrales Paradigma angestrebt werden soll, ist es wichtig, möglichst früh die Benutzeranforderungen mit den Merkmalen existierender Teilkomponenten abzugleichen. Die bisher aufwendigen Transformationen eines Fachkonzeptes in ein DV-Konzept und weiter in die Implementierung werden dadurch reduziert, daß auf allen Ebenen die Terminologie der Domäne verwandt werden kann und auch die in der Realität angetroffenen Strukturen weitestgehend unverändert bis auf die Implementierungsebene abgebildet werden können. Die Aufgaben verlagern sich also weg von der Transformation zwischen unterschiedlichen Abstraktionsebenen hin zu einem effizienten Umgang mit der Bibliothek wiederverwendbarer Bausteine, die im Laufe der Zeit an Allgemeinheit und Umfang zunehmen wird.

3 Objektorientierte Designmethoden

In diesem Abschnitt soll ein Überblick über existierende objektorientierte Entwurfsmethoden gegeben und diese Methoden in die von Scheer definierte ARIS-Architektur[10], die sich als Rahmenkonzept zur Entwicklung von Informationssystemen versteht, eingeordnet werden.

3.1 Überblick

Von einer Methode wird nicht nur erwartet, eine Notation zur Dokumentation der Designergebnisse anzubieten, die meisten Methoden umfassen auch eine Beschreibung des Designprozesses mit mehr oder weniger exakten Designrichtlinien und einige wenige bieten Metriken zur Bewertung der Ergebnisse an.[11] Reine Notationen werden im folgenden der Übersicht halber nicht berücksichtigt, sofern sie nicht durch häufigen Einsatz eine gewisse Bedeutung erreicht haben. Ebensowenig werden Vorgehensweisen oder Designempfehlungen aufgeführt, denen keine graphische Notation zugrunde liegt.

Die vorgestellten Entwurfsmethoden resultieren im wesentlichen aus zwei unterschiedlichen Vorgehensweisen:

- Abstrahierung der Konstrukte objektorientierter (bzw. objektbasierter) Programmiersprachen,
- Erweiterung traditioneller Entwurfsmethoden um objektorientierte Konzepte.

Diese beiden Ursprünge werden aus der Übersichtsdarstellung (Abb. 3) deutlich, wo die Zusammenhänge der wesentlichen objektorientierten Designmethoden aufgezeigt werden. Die dargestellten Abhängigkeiten repräsentieren die gegenseitige Beeinflussung und Weiterentwicklung von Methoden (d.h. die Berücksichtigung von Konzepten bzw. die Übernahme von Notationen; eine schwache Beeinflussung ist gestrichelt dargestellt).
Insbesondere Smalltalk als "reinste" objektorientierte Programmiersprache und Ada mit überragender Bedeutung in der amerikanischen Forschung sind als Ausgangspunkte der Entwicklung einer Reihe von Methoden anzusehen. Die Gliederung traditioneller Analyse- und Designmethoden gemäß der Betonung eher datenorientierter bzw. funktionaler (functional decomposition) Aspekte spiegelt sich auch in deren Weiterentwicklung

---

10 Zur generellen Zielsetzung der ARIS-Architektur vgl. Scheer, A.-W.: a.a.O., S. 1-3.

11 Vgl. Wirfs-Brock, R.J.; Johnson, R.E.: Surveying current research in object-oriented Design, in: Communications of the ACM, 33 (1990) 9, S. 104-124.

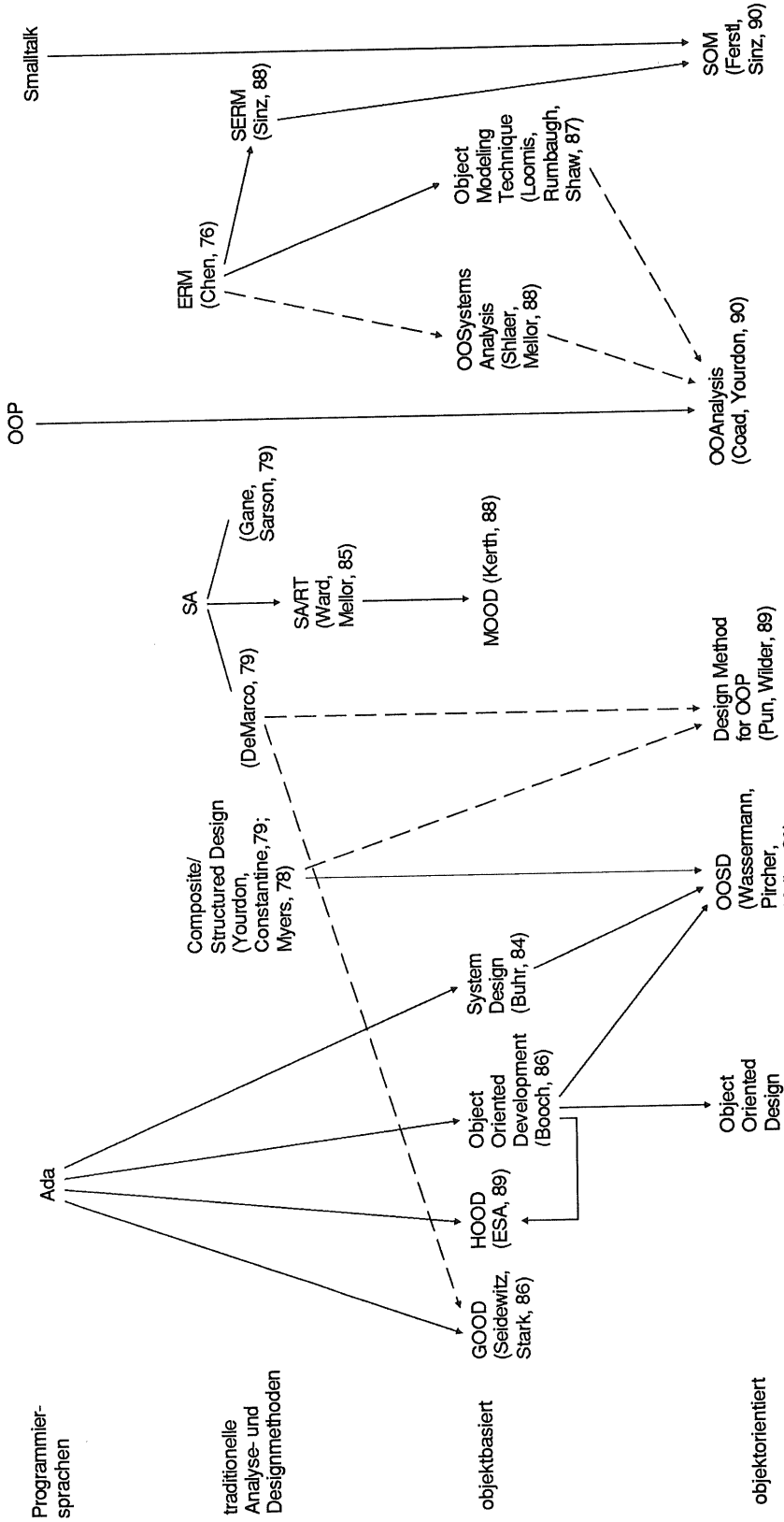


Abb. 3: Stammbaum objektorientierter Designmethoden

wider:[12] Objektorientierte Methoden, die auf datenorientierten Methoden aufbauen, haben ihren Schwerpunkt in der Definition von Klassen und ihren Beziehungen untereinander, wogegen Weiterentwicklungen funktionsorientierter Methoden eher das Methodenprotokoll von Klassen und die Kommunikation über Nachrichten in den Mittelpunkt der Betrachtung stellen.

Analog zur Klassifizierung von Programmiersprachen nach Wegner bzgl. der Berücksichtigung objektorientierter Basiskonzepte[13] werden die Methoden in objektbasierte bzw. objektorientierte Verfahren unterteilt. Hier wird deutlich, daß eine Reihe von Methoden - insbesondere die aus dem Umgang mit Ada entstandenen - nur eine Teilmenge der objektorientierten Basiskonzepte unterstützen. Auch Methoden, die reine Datenmodellierungstechniken, wie z. B. das Entity-Relationship-Modell um einige semantische Konstrukte erweitern, repräsentieren durch den Mangel an Darstellungsmöglichkeiten der funktionalen Aspekte nicht den vollen objektorientierten Anspruch.
Interessant sind insbesondere die Methoden, die alle Basiskonzepte objektorientierter Entwicklung umfassen, so daß die folgenden Designansätze einer genaueren Bewertung unterzogen werden sollen:

- Object-Oriented Design (Booch, 1991)[14],
- Object-Oriented Structured Design (Wasserman, Pircher, Muller, 1989)[15],
- Design Method for Object-Oriented Programming (Pun, Wilder, 1989)[16],
- Object-Oriented Analysis (Coad, Yourdon, 1990)[17],
- Semantisches Objektmodell (Ferstl, Sinz, 1990)[18].

Obwohl Object-Oriented Analysis vornehmlich die Analysephase unterstützt und ihren Schwerpunkt in der Darstellung des Fachkonzeptes und der Spezifikation von

---

12 Zur Klassifizierung von Designmethoden vgl. Meyer, B.: Object-Oriented Software Construction. Hemel Hempstead 1988, S. 41ff.

13 Vgl. Wegner, P.: Learning the Language, in: BYTE, 14 (1989) 3, S. 245-253.

14 Vgl. Booch, G.: Object-Oriented Design with Applications. Reading et al. 1991.

15 Vgl. Wasserman, A.I.; Pircher, P.A.; Muller, R.J.: The Object-Oriented Structured Design Notation for Software Representation, in: Computer, 23 (1990) 3, S. 50-63.

16 Vgl. Pun, W.W.Y.; Winder, R.L.: A Design Method for Object-Oriented Programming, in: Proceedings of ECOOP '89, S. 225-240.

17 Vgl. Coad, P.; Yourdon, E.: Object-Oriented Analysis. Englewood Cliffs 1990.

18 Vgl. Ferstl, O.K.; Sinz, E.J.: Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM), in: Wirtschaftsinformatik, 32 (1990) 6, S. 566-581.

Benutzeranforderungen hat, wird sie in den Kreis der untersuchten Methoden mitaufgenommen, da ihre Konstrukte auch geeignet sind, DV-technische Lösungen zu repräsentieren.

3.2 Einordnung in die ARIS-Architektur

Wenn im vorherigen Abschnitt das Inventar an existierenden Methoden gewissermaßen in einer Bottom-Up-Vorgehensweise aufgelistet wurde, sollen demgegenüber nun diese Ansätze in ein Rahmenkonzept eingeordnet werden, das weniger auf empirischen Erkenntnissen als auf theoretischen Ableitungsschritten beruht. Die von Scheer anhand der Anforderungen betrieblicher Informationssysteme abgeleitete ARIS-Architektur zerlegt, um Komplexität und Redundanz zu reduzieren, die Beschreibung eines Systems in unterschiedliche Sichtweisen (Daten-, Funktions- und Organisationssicht), deren Verbindungen in einer Steuerungssicht wieder eingeführt werden. Diese Perspektiven werden jeweils nochmal in die Beschreibungsebenen Fachkonzept, DV-Konzept und Implementierung unterteilt (vgl. Abb. 4).[19]

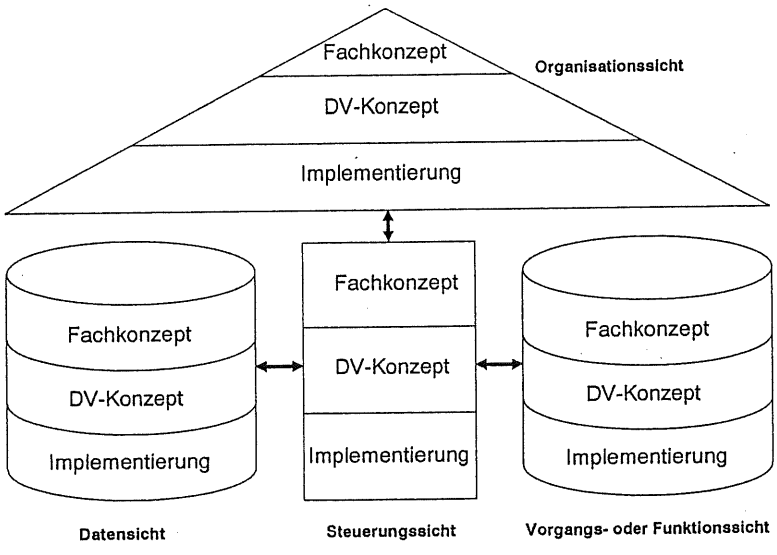


Abb. 4: ARIS-Architektur (Quelle: Scheer, A.-W. 1991)

[19] Vgl. Scheer, A.-W.: a.a.O., S. 13ff.

Vergleich von Methoden zum objektorientierten Design von Softwaresystemen

Dieser sehr allgemeine Anspruch, der diesem Ansatz zugrunde liegt, erlaubt es, auch objektorientierte Methoden innerhalb dieser Architektur zu integrieren: Die der Objektorientierung eigene, sehr enge Kopplung von Daten und Funktionen kann innerhalb der Steuerungsebene abgebildet werden. Diese auf einer Metaebene vollzogene Verbindung kann auch anhand der historischen Entwicklung bei einigen objektorientierten Methoden verfolgt werden, wo reine Datenmodellierungsmethoden um Konstrukte zur funktionalen Spezifizierung ergänzt werden.[20]

Die Eignung dieses allgemeinen Ansatzes auch zur Spezifizierung objektorientierter Systeme weist Scheer explizit durch die Angabe eines Metamodells nach, das den Zusammenhang zwischen sichten- und objektorientiertem Entwurf abbildet (vgl. Abb. 5).[21] Hier wird die Zuordnung zwischen Informationsobjekten und Klassen sowie die Bildung von Hierarchien zwischen Klassen dargestellt und Nachrichten durch die sendenden und empfangenden Objekte, die als Parameter verwendeten Datenelemente und die durchzuführenden Methoden beschrieben. Eine konkrete Modellierung ist dann auf der Ausprägungsebene dieses Modells (d. h. durch die Angabe der Entities) möglich.

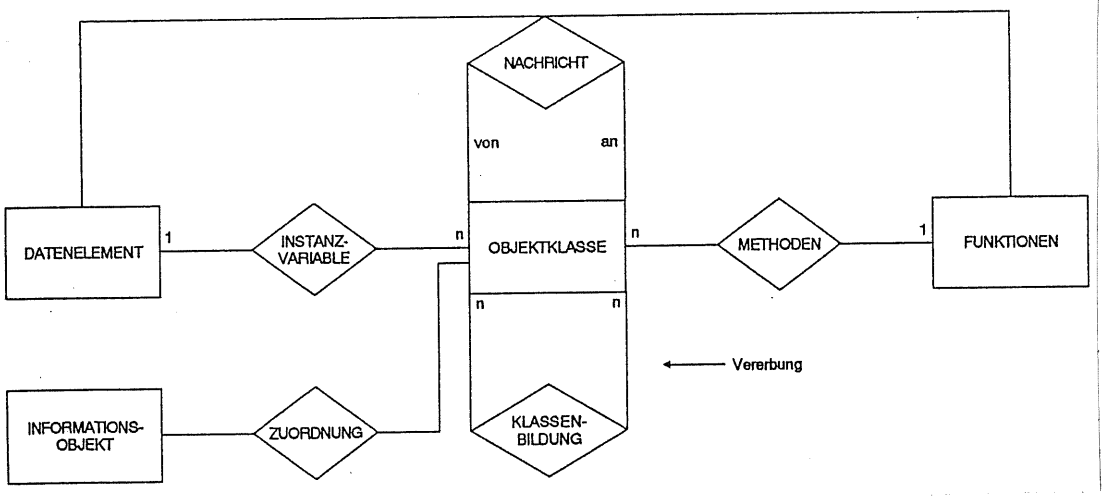


Abb. 5: Metamodell Objektorientierter Entwurf (Quelle: Scheer, A.-W. 1991)

---

[20] Als Beispiel siehe hierzu die Weiterentwicklung vom SERM zum SOM, vgl. Ferstl, O.K.; Sinz, E.J.: a.a.O.

[21] Vgl. Scheer, A.-W.: a.a.O., S. 124f.

Hieraus wird deutlich, daß die ARIS-Architektur geeignet ist, zur einer Vereinheitlichung der Methodenvielfalt im allgemeinen - und der objektorientierten Methoden im speziellen - beizutragen, was allerdings natürlich die Suche nach einer geeigneten objektorientierten Designmethode mit einer intuitiveren Notation als die der Ausprägungsebene des Metamodells nicht überflüssig macht.

4 Bewertung objektorientierter Designmethoden

Nachfolgend sollen von den oben aufgeführten Aspekten insbesondere die methodische Unterstützung beim Designprozeß und die Notation zur Dokumentation der Designergebnisse zum Vergleich der genannten Methoden herangezogen werden.

4.1 Notation

Die Akzeptanz einer Methode hängt in entscheidendem Maße von der zugehörigen graphischen Notation ab, in vielen Fällen wird die Methode sogar fälschlicherweise mit der Notation gleichgesetzt. Nur mit Hilfe einer graphischen Abbildungsmöglichkeit ist es möglich, auch komplizierte Strukturen in einer übersichtlichen Art und Weise zu präsentieren.

4.1.1 Vergleichende Bewertung

Die graphische Darstellung hat den Zwiespalt zu lösen, einerseits alle für die nachfolgenden Schritte möglicherweise relevanten Informationen repräsentieren zu können, andererseits aber auch diese Komplexität mit hoher Übersichtlichkeit, Flexibilität und leichter Erlernbarkeit zu kombinieren. Einige Methoden ergänzen die reine Diagrammdarstellung um formalisierte textuelle Beschreibungen (templates), in denen für das Gesamtverständnis weniger wichtige Detailinformationen abgelegt werden.

In Tabelle 1 wird das Kriterium der Vollständigkeit weiter detailliert und die Notation auf folgende Darstellungsmöglichkeiten hin untersucht:

- **Existenz von Klassen**: Als Grundlage weiterer Designaktivitäten müssen die zu definierenden Klassen eines Systems dargestellt werden.

- **Beziehungen zwischen Klassen**: In objektorientierten Systemen gibt es einige

wichtige Arten von Beziehungen zwischen Klassen, die schon auf der Designebene dargestellt werden sollten: eine Klasse kann Unterklasse (bzgl. der Vererbungshierarchie) einer anderen Klasse sein, Instanzvariablen einer Klasse können auf Instanzen anderer Klassen verweisen oder eine Klasse kann sich der Dienste einer anderen Klasse bedienen (d.h. Nachrichten an diese Klasse schicken). Diesen generell relevanten Beziehungstypen, die durch die Angabe der Kardinalität ergänzt werden können, stehen Beziehungen gegenüber, die nicht in allen objektorientierten Programmiersprachen abbildbar sind: Klassen können selbst wieder als Objekte aufgefaßt werden, die dann als Instanzen einer zugehörigen Metaklasse verstanden werden (z.B. in Smalltalk). Desweiteren erlauben einige getypte Programmiersprachen die Definition generischer Klassen, die von einem Typenparameter abhängig sind, so daß es Sinn macht, auch die Instantiierung solcher Klassen als weitere Beziehung aufzunehmen.

- **Zuordnung von Attributen zu Klassen**: Instanzvariablen repräsentieren den inneren Zustand eines Objektes. Neben einer reinen Auflistung kann es nötig sein, die Verweise dieser Variablen auf andere Klassen sowie Initialisierungs- und sonstige Nebenbedingungen festzuhalten.

- **Zuordnung von Methoden**: Die Methoden definieren die Möglichkeiten zur Manipulation und zum Umgang mit Objekten dieser Klasse. Neben der Nennung der Methoden (mit zugehörigen Parametern) sollte es eine Möglichkeit geben, den Sichtbarkeitsbereich (wer darf diese Methoden anstoßen ?), sowie eventuell notwendige Pre- und Postbedingungen und Ausnahmebehandlungen zu definieren.

- **Zustandsübergänge eines Objektes**: Zum Verständnis der Funktionalität einer Klasse ist es oftmals sinnvoll, die Zustände, die eine Instanz dieser Klasse von der Instantiierung bis zur Löschung durchlaufen kann, graphisch zu veranschaulichen.

- **Nachrichtenaustausch zwischen Objekten**: Die eigentliche Funktionalität eines Systems wird durch den Autausch von Nachrichten zwischen Objekten und der entsprechenden Reaktion darauf realisiert, so daß es dringend notwendig ist, eine Möglichkeit zur Darstellung des Nachrichtenflusses anzubieten. Darüberhinaus können Informationen, wie die Art der Sichtbarkeit zwischen Objekten, die Synchronisationsverfahren beim Ablauf paralleler Prozesse und die Darstellung der zeitlichen Abfolge der Nachrichtenkommunikation für das Verständnis hilfreich sein.

Methoden	OOD	OOSD	OOA	SOM	OOP
Darstellung von					
❒ Existenz von Klassen	+	+	+	+	+
❒ Beziehungen zwischen Klassen					
- Subklasse	+	+	+	+	+
- Redefinition von Methoden	-	+	+	-	-
- Part-Of	+	-	+	+	-
- Uses	+	+	+	+	+
- Metaklasse	+	-	-	-	-
- Instantiierung generischer Klassen	+	+	-	-	-
- Angabe der Kardinalität	+	-	+	+	-
❒ Zuordnung von Attributen (Instanzvariablen)					
- Auflistung	+ (template)	+	+	+	-
- Instanzreferenzen	+ (template)	-	+	(+)	-
- Initialisierung	+ (template)	-	-	-	-
- Nebenbedingungen	+ (template)	-	+ (template)	-	-
❒ Zuordnung von Methoden zu Klassen					
- Auflistung d. Methoden	+ (template)	+	+	+	+
- Sichtbarkeitsbereich	+ (template)	(+)	-	-	-
- Auflistung der Parameter	+ (template)	+	-	+	-
- Pre-/Postbedingungen	+ (template)	-	+ (template)	-	-
- Ausnahmebehandlungen	+ (template)	+	+ (template)	-	(+)
❒ Zustandsübergänge eines Objektes	+	-	+	-	-

Methoden	OOD	OOSD	OOA	SOM	OOP
Darstellung von					
❒ Nachrichtenaustausch					
- Auflistung der Nachrichten zwischen Objekten	+	+	-	-	+
- Art der Sichtbarkeit der kommunizierenden Objekte	+	(+)	(+)	(+)	-
- Synchronisationsverfahren paralleler Prozesse	+	+	-	-	-
- Zeitliche Abfolge der Nachrichtenkommunikation	+	-	-	-	(+)
❒ Modularisierung					
- Bildung von Teil-systemen	+	+	+	-	+
- Zuordnung von Modulen zu verschiedenen Prozessoren	+	-	-	-	-

Erklärung:

+	vorhanden
-	nicht vorhanden
(+)	nicht in vollem Umfang vorhanden
OOD	Object-Oriented Design
OOSD	Object-Oriented Structured Design
SOM	Semantisches Objektmodell
OOA	Object-Oriented Analysis
OOP	Design Method for Object-Oriented programming

Tabelle 1: Darstellungsmöglichkeiten objektorientierter Notationen

- **Modularisierung (Clusterbildung)**: Bei großen Systemen ist es dringend erforderlich, zur Reduzierung der Komplexität eine Zerlegung in Teilkomponenten vorzunehmen, um die Übersichtlichkeit zu gewährleisten und die sinnvolle Zusammenarbeit einer Entwicklungsgruppe zu unterstützen. Diese Modularisierung sollte graphisch abbildbar sein, darüberhinaus ist eine eventuelle Darstellung der Verteilung der Prozesse bei Mehrprozessorsystemen als zusätzliches Feature denkbar.

Eine vergleichende Betrachtung der Methoden ergibt eine fast vollständige Abdeckung aller betrachteten Kriterien durch Object-Oriented Design (unter Berücksichtigung der zugehörigen Templates). Alle anderen Methoden können nur einen Ausschnitt der aufgelisteten Aspekte repräsentieren, wobei der jeweilige Schwerpunkt auf den Ursprung der Methoden zurückzuführen ist: Object-Oriented Structured Design weist Darstellungsmängel bei der näheren Spezifizierung von Instanzvariablen und der Sichtbarkeitsbeziehungen zwischen Klassen auf, wohingegen die eher von der Datenmodellierung abstammenden Methoden (Object-Oriented Analysis, Semantisches Objektmodell) fast keine Features vorsehen, den Nachrichtenaustausch zwischen Objekten zu veranschaulichen. Hier beschränkt man sich darauf, den funktionalen Aspekt lediglich durch die isolierte Auflistung von Methoden jeder Klasse darzustellen. Der Ansatz von Pun, Wilder, der sich allerdings auch als noch zu verbessernder Vorschlag versteht, hat seinen Schwerpunkt nicht in der graphischen Darstellung der Ergebnisse, sondern versucht, die Strukturierung des Entwurfsprozesses und die damit notwendigen Aktivitäten in den Mittelpunkt zu stellen.

Alle betrachteten Notationen sind - wenn auch durch die Nähe zu einer Programmiersprache geprägt - programmiersprachenunabhängig und somit in allen objektorientierten Entwicklungsumgebungen sinnvoll einsetzbar.

Bewertungskriterien wie Einfachheit, Übersichtlichkeit und Erlernbarkeit sind einer objektiven Beurteilung schwer zugänglich zu machen, da sie in hohem Maße von den gewohnten Arbeitstechniken und individuellen Vorlieben des Benutzers abhängen. Eine differenzierende Operationalisierung dieser Aspekte ist somit wenig sinnvoll, auch wenn diese Kriterien eng mit der individuellen Anpaßbarkeit und Flexibilität einer Methode zusammenhängen. Dies wird in hohem Maße durch die Modularität einer Notation bestimmt, d. h., ob der Benutzer jeweils nur die für ihn in diesem Stadium interessanten Aspekte darstellen kann, oder ob nur eine vollständige Darstellung als korrektes, verständliches Diagramm gilt. Booch löst dieses Problem durch die Definition sechs verschiedener Arten von Diagrammen, wobei dann natürlich die Konsistenz dieser

Darstellungen sichergestellt werden muß. Pun, Wilder verwenden zwei Arten von Diagrammen, nämlich Object Interaction Diagrams zur Darstellung des Nachrichtenaustauschs zwischen Klassen und Class Struture Charts zur Spezifizierung der Funktionalität einer bestimmten Operation (Methode). Alle anderen betrachteten Notationen basieren auf einer einzigen Diagrammdarstellung, erlauben jedoch jeweils die Projektion des Gesamtzusammenhangs auf einen zu betrachtenden Aspekt.

4.1.2 Beispiele

Um einen Eindruck der unterschiedlichen Darstellungskonzepte zu geben, wird ein sehr einfaches Beispiel mit Hilfe der vorgestellten Notationen repräsentiert: In einem Industrieunternehmen sollen zwei Arten von Teilen (Zylinder, Kugeln), die auf einer Packliste aufgeführt sind, einzeln in Kartons verpackt werden. Ein Verpackungsautomat übernimmt die Durchführung der Verpackung, benötigt allerdings zu jedem Teil den passenden Karton, dessen Ausmaße von denen des zu verpackenden Teils und dessen Typ vom Gewicht des Teils abhängen.

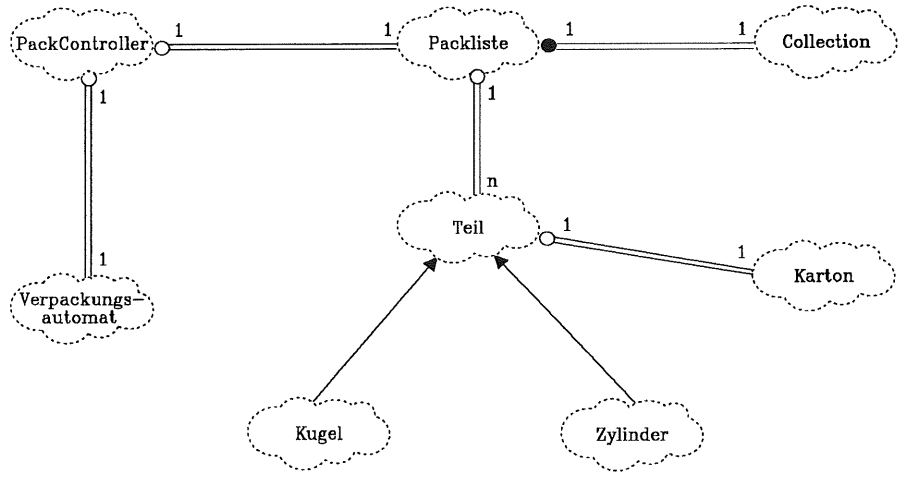
Den nachfolgenden Modellierungen liegt die Überlegung zugrunde, einen "Verpackungscontroller" mit der Überwachung des gesamten Vorgangs zu betrauen. Dieser fordert alle in der Packliste aufgeführten Teile auf, aus ihren Abmessungen Länge, Breite und Höhe des benötigten Kartons zu bestimmen und aus dem Gewicht, das sich aus dem Produkt von Volumen und spezifischem Gewicht ergibt, den Typ des Kartons abzuleiten. Letztendlich sendet der Verpackungscontroller dann die Nachricht an den Verpackungsautomat, alle Teile mit Hilfe des individuell bestimmten Kartons zu verpacken. Die Klasse Teil wird dabei als abstrakte Klasse verstanden, wo alle Variablen und Methoden, die für beide Teiletypen relevant sind, definiert werden. Polymorphismus kommt an dieser Stelle zum Einsatz, wenn für Kugel und Zylinder die Methode 'berechneVolumen' definiert wird, die Ausprägungen aber entsprechend den geometrischen Eigenarten der Körper vollkommen unterschiedlich aussehen.

Ziel dieses Abschnitts ist nicht ein vollständiges Design des Beispiels mit Hilfe jeder vorzustellenden Methode, sondern die Vermittlung eines Eindrucks von den Eigenarten jeder Notation, so daß im folgenden auf eventuell detaillierende Templates verzichtet wird und sich die Erläuterungen der Notation auf das absolut Notwendigste zum Verständnis des Beispiels beschränken.

4.1.2.1 Object-Oriented Design

Class Diagrams stellen Beziehungen zwischen Klassen dar, die als gestrichelte Wolken symbolisiert werden. Vererbungsbeziehungen werden als durchgezogene Pfeile notiert, wogegen Doppellinien die Verwendungs-Beziehung zum einen bei der Implementierung (gefüllter Kreis), zum anderen bei der Definition des Interfaces (leerer Kreis) ausdrücken. Object Diagrams zeigen den Nachrichtenaustausch zwischen Objekten, die Namen der angestoßenen Methoden werden an den Verbindungskanten notiert.

Class Diagram:



Object Diagram:

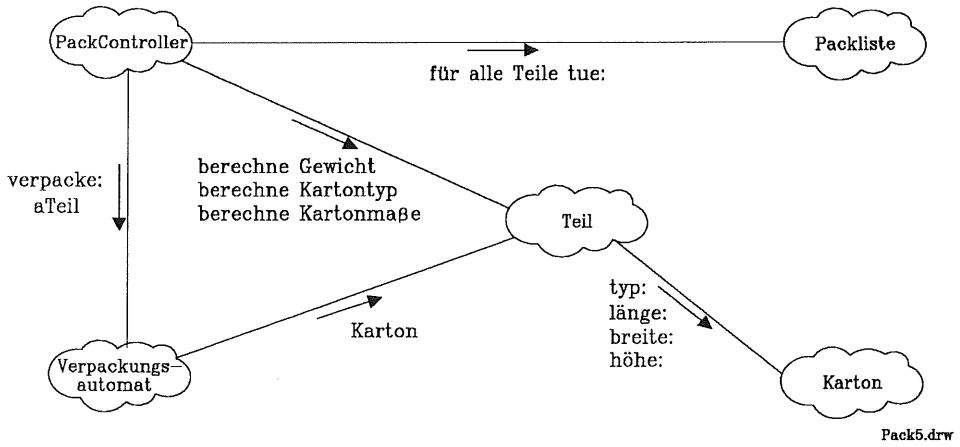


Abb. 6: Modellierung des Beispiels nach Object-Oriented Design

4.1.2.2 Object-Oriented Analysis

Klassen werden als Rechtecke dargestellt, die zugehörigen Instanzvariablen und -methoden sind darin aufgelistet. Repräsentiert werden die Sichtbarkeitsbeziehungen der Klassen untereinander durch einfache Verbindungslinien (versehen mit Symbolen zur Kennzeichnung der Kardinalitäten), sich gabelnde Verbindungslinien stellen die Bildung von Klassenhierarchien dar, graue Pfeile zeigen den Nachrichtenfluß auf.

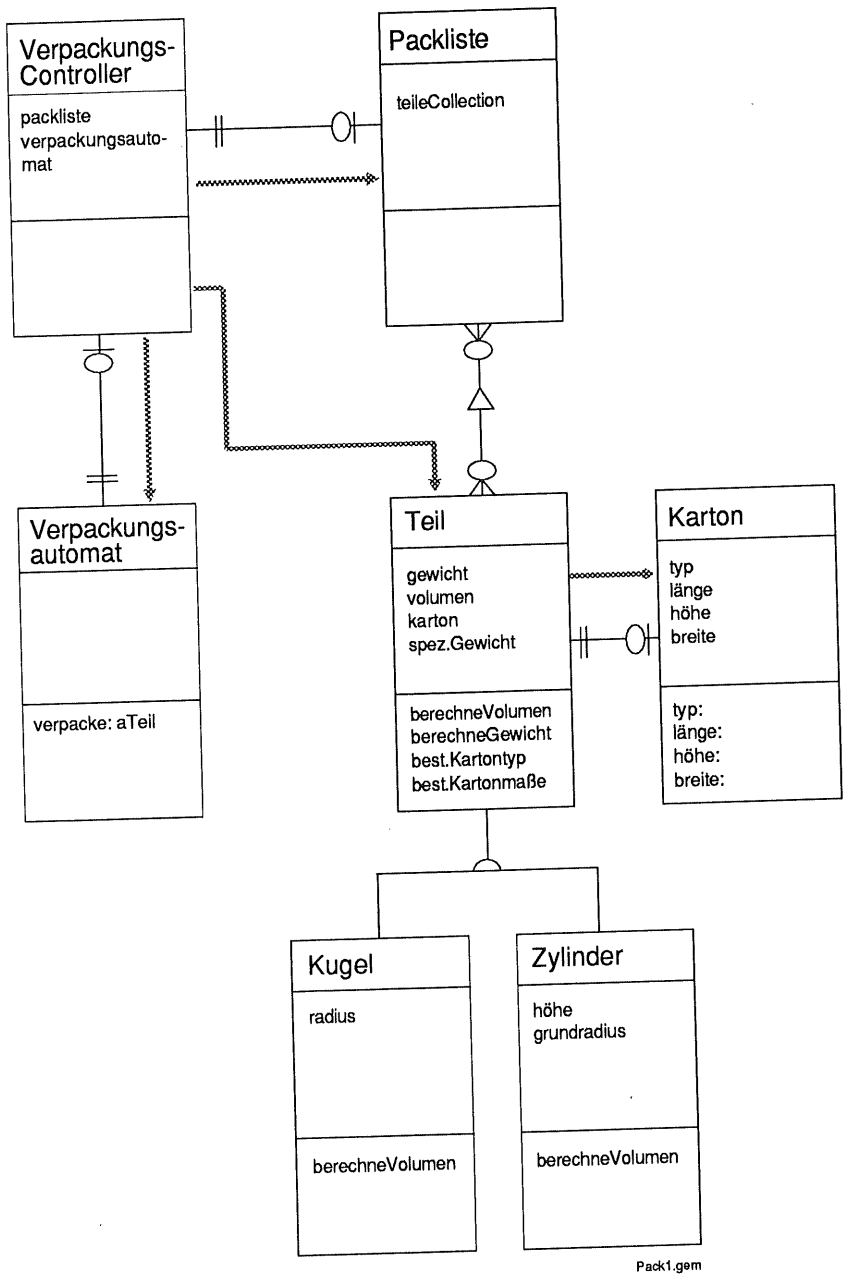


Abb. 7: Modellierung des Beispiels nach Object-Oriented Analysis

4.1.2.3 Object-Oriented Structured Design

Klassen werden als Rechtecke dargestellt; nach außen sichtbare Methoden werden durch überlappende Rechtecke repräsentiert, Instanzvariablen innerhalb von abgerundeten Rechtecken (innerhalb des Klassensymbols) aufgelistet. Durchgezogene Pfeile stellen den Nachrichtenaustausch zwischen Klassen dar, gestrichelte Pfeile repräsentieren die Vererbungsbeziehung und weisen von Unter- zu Oberklasse.

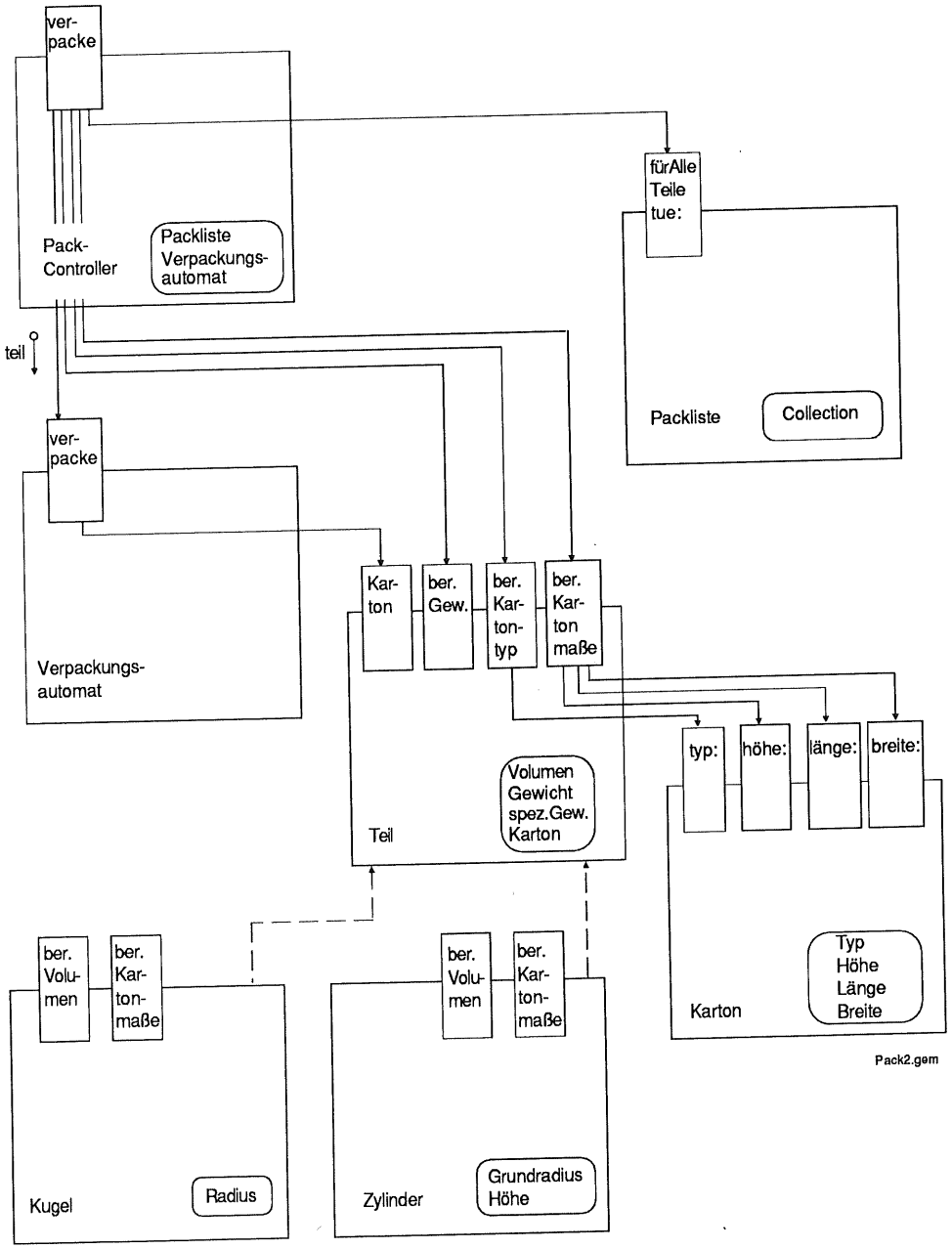


Abb. 8: Modellierung des Beispiels nach Object-Oriented Structured Design

4.1.2.4 Design Method for Object-Oriented Programming

Object Interaction Diagrams zeigen den Nachrichtenaustausch zwischen Objekten in zwei Schichten auf: Der User-Interface Level enthält alle Objekte, mit denen der Anwender direkt in Kontakt tritt (Menues, Forms), der User-Transparent Level die intern verborgenen Objekte; Pfeile symbolisieren den Austausch von Nachrichten. Class Structure Charts spezifizieren einzelne Methoden, wobei Sequenzen, Iterationen (als Ellipsen) und Fallunterscheidungen (als Rauten) im Ablauf der Methode ausgedrückt werden können.

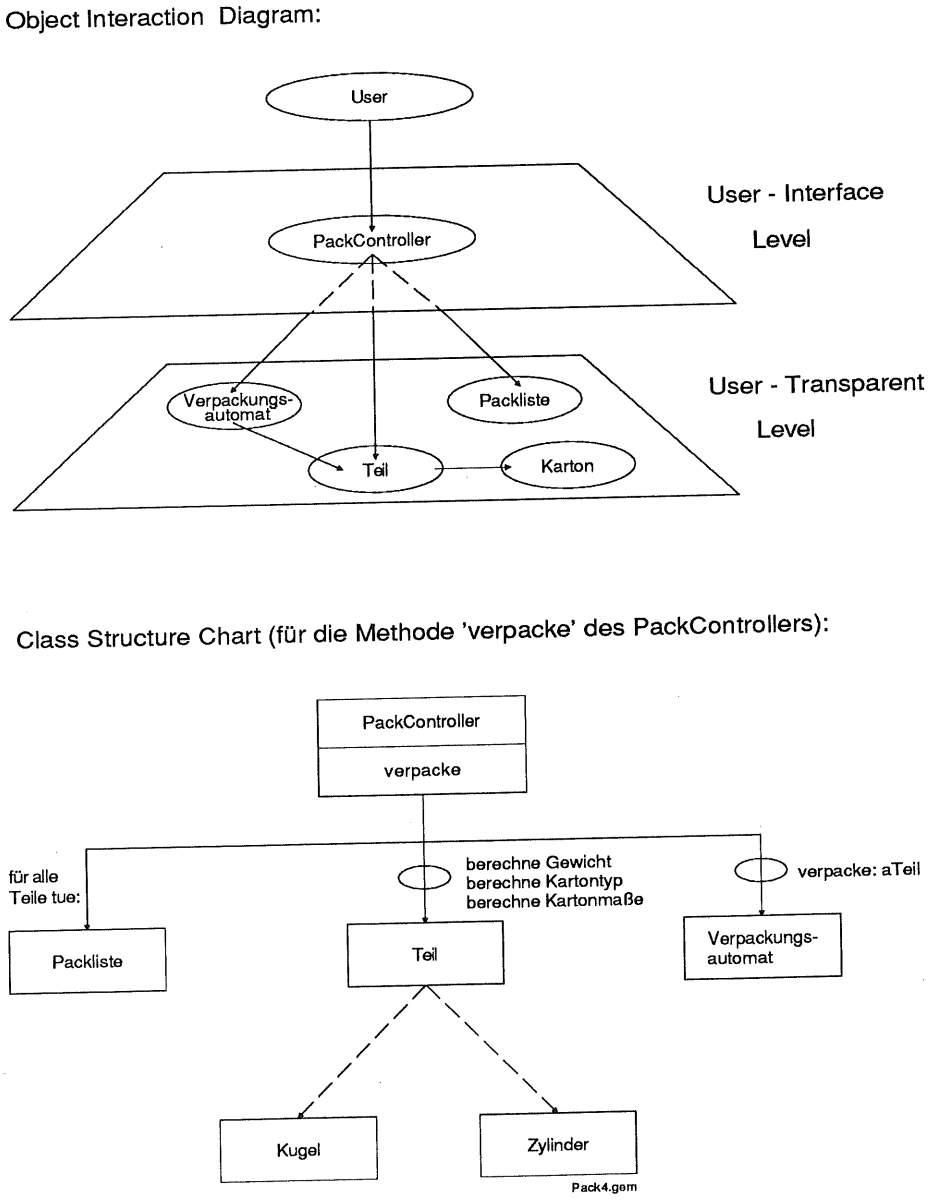
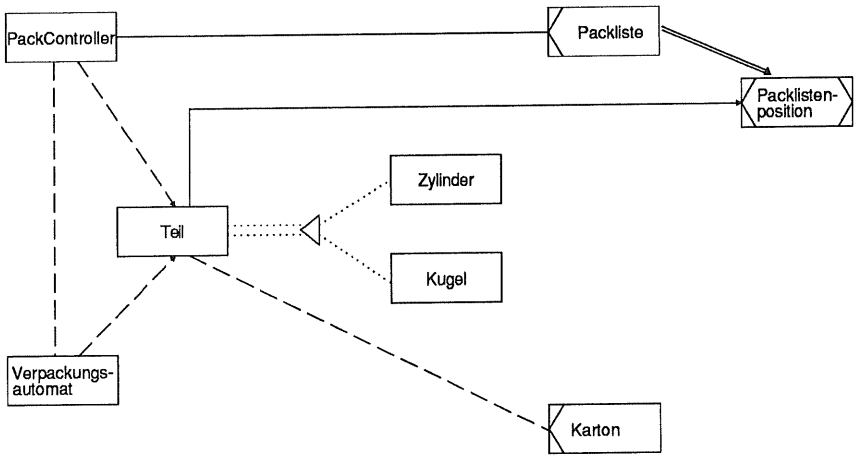


Abb. 9: Modellierung des Beispiels nach Design Method for Obj.-Oriented Programming

4.1.2.5 Semantisches Objektmodell

Die Notation des Sematischen Objektmodells stellt Beziehungen zwischen Klassen dar und kennt 'interacts_with'- (als gestrichelte Linien), 'is_part_of'- (als durchgezogene Linien) und 'is_a'-Beziehungen (als gepunktete Linien). Die Kardinalität der Beziehung wird durch die Art der Verbindung (ungerichtet (0,1), gerichtet (0,*), doppelt gerichtet (1,*)) zum Ausdruck gebracht. Die Visualisierung der Existenzabhängigkeiten geschieht durch die Anordnung des Diagramms als quasi-hierarchischer Graph, wobei die abhängigen Objekte rechts von den zugehörigen unabhängigen angeordnet werden.



entstanden aus folgendem SERM:

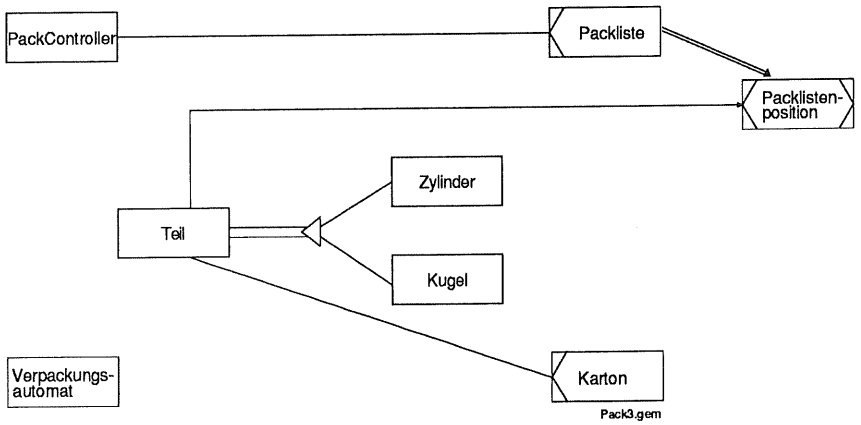


Abb. 10: Modellierung des Beispiels nach dem Semantischen Objektmodell

4.2 Designprozess

Neben den reinen Darstellungsmöglichkeiten ist es für die Akzeptanz und die Güte der Ergebnisse entscheidend, dem Entwickler eine Vorgehensweise zum Design an die Hand zu geben, die natürlich konsistent in den globalen Lifecycle eingepaßt werden muß. Der folgende Abschnitt skizziert die Beschreibung des Designprozesses im Rahmen der fünf genannten Methoden und versucht, anschließend eventuelle Gemeinsamkeiten zu identifizieren.

4.2.1 Object-Oriented Design

Booch betont die Wichtigkeit eines evolutionären Vorgehens unter möglicherweise ständigem Wechsel zwischen Top-down und Bottom-up Betrachtungsweise und der entsprechenden Detailierungstiefe.[22] Dies wird unterstützt durch die Vielfalt der Sichten, die die verschiedenen Arten von Diagrammen erlauben, so daß die Struktur des Designprozesses sehr stark vom speziellen Problem abhängt, generell jedoch die folgenden Schritte identifiziert werden können, die iterativ durchlaufen werden:

- Identifizierung der Klassen und Objekte,
- Identifizierung der Semantik der Klassen und Objekte,
- Identifizierung der Beziehungen zwischen Klassen und Objekten,
- Implementierung der Klassen und Objekte.

Der Designprozess beginnt also mit der Anwendung von Techniken der Domain Analysis auf den zu modellierenden Fachbereich und endet mit der Entscheidung über die interne Repräsentation der Klassen und Objekten, woran nochmal der fließende Übergang zwischen Design und Implementierung sowie der allmähliche Wechsel des Beobachtungsstandpunktes des Designers von der Außen- zur Innensicht der Objekte sichtbar werden.

4.2.2 Object-Oriented Analysis

Coad, Yourdon betrachten den Designprozess als kontinuierliche Erweiterung und Detaillierung der noch völlig programmiersprachenunabhängigen Analyseergebnisse unter Berücksichtigung der zugrundezulegenden Hard- und Softwarearchitektur. Object-

---

[22] Vgl. Booch, G.: a.a.O., S.188ff.

Oriented Analysis besteht aus fünf Hauptschritten:[23]

- Identifizierung der Klassen,
- Identifizierung der Strukturen zwischen Klassen,
- Identifizierung von Clustern (subjects),
- Definition von Attributen,
- Definition von Methoden,

denen jeweils aufeinander aufbauende Layers zuzuordnen sind, die als Projektion des Gesamtmodells auf die jeweilige Hauptansicht aufgefaßt werden können.

4.2.3. Design Method for Object-Oriented Programming

Die von Pun, Wilder vorgeschlagene Designmethode deckt die Analyse- und Designphase in einer dreistufigen Vorgehensweise ab:
In der Phase **Conceptual Level** wird das konzeptuelle Modell der Applikation aufgestellt, mit den Benutzeranforderungen abgeglichen und somit die relevanten Objekte und Operationen definiert, die als Object Interaction Diagrams dargestellt werden. Diese Phase wird weiter in zwei Schritte unterteilt, die sich mit der Gestaltung des User Interfaces und der systeminternen Architektur beschäftigen.
In der folgenden Phase **System Level** wird die erarbeitete Spezifikation weiter detailliert und der Kontrollfluß der Applikation festgelegt, so daß auf dieser Stufe auch die Entscheidung über Neuentwicklung von Teilkomponenten oder Modifikation existierender Klassen getroffen wird. In der Phase **Specification Level** wird aufbauend auf den vorgelagerten Ergebnissen ein implemetierungsgerechtes Design erarbeitet und mit Hilfe von Class Structure Charts dargestellt.

4.2.4 Object-Oriented Structured Design

Object-Oriented Structured Design liegt keine eigenständige Vorgehensweise zugrunde. Die Autoren betonen die Offenheit ihrer Notation zur Verwendung innerhalb verwandter Designmethoden (Booch, HOOD[24]) und verweisen lediglich auf einige sehr allgemeine Richtlinien (Kombination von Top-down und Bottom-up-Vorgehensweise, maximale Anzahl der Operationen einer Klasse).

---

[23] Vgl. Coad, P.; Yourdon, E.: a.a.O., S. 163ff.

[24] Vgl. Heitz, M.: HOOD: Hierarchical Object-Oriented Design for large technical and realtime software. CISI Ingenierie, Direction Midi Pyrenees 1987.

4.2.5 Semantisches Objektmodell

Das Semantische Objektmodell basiert auf den Ergebnissen der Datenstrukturierung mit Hilfe des Strukturierten Entity-Relationship-Modells (SERM)[25]. Der eigentlich objektorientierte Entwurf beginnt mit der konzeptionellen Objektmodellierung, bei der im Objektsystementwurf die Beziehungen der Objekttypen beschrieben und im Objektentwurf die Struktur und das Verhaltens eines jeden Objekttyps spezifiziert werden. Zur Abbildung durchgängiger Abläufe werden dann Vorgangsobjekttypen gebildet, die das Zusammenwirken mehrerer konzeptioneller Objekttypen bestimmen.

4.2.6 Bewertung

Mit Ausnahme des sehr eng an der zuvor durchgeführten Datenmodellierung angelehnten Semantischen Objektmodells liegt allen vorgestellten Designprozessen eine sehr ähnliche Struktur zugrunde, wobei die Methoden von Coad, Yourdon und Pun, Wilder durch die Integration der Analysephase zusätzliche Aktivitäten zur Erhebung der Benutzeranforderungen an das zu entwickelnde System beinhalten. Mit Ausnahme dieser Besonderheiten kann eine generell verwendbare Grobstruktur wie folgt identifiziert werden:

- Identifizierung der Klassen und Objekte,
- Identifizierung der Beziehungen zwischen Klassen und Objekten,
- Definition von Attributen,
- Definition von Methoden,
- Spezifizierung des Nachrichtenaustauschs,
- Identifizierung von Modulen.

Die großen Übereinstimmungen in der Vorgehensweise des Designs legen es nahe, eine weitgehende Unabhängigkeit des Designprozesses von der verwendeten Notation zu konstatieren, was bei Object-Oriented Structured Design auch ganz explizit von den Autoren vertreten wird.[26] Es scheint somit ein gangbarer Weg zu sein, um einer weiteren Diversifizierung der Methoden entgegenzuwirken und insbesondere eine Austauschbarkeit der Ergebnisse zur erhalten, sich in absehbarer Zeit auf eine graphische Notationsform als Quasi-Standard zu einigen, der die wesentlichen Vorteile der oben vorgestellten Ansätze

---

[25] Vgl. Sinz, E.J.: Das Strukturierte Entity-Relationship-Modell (SER-Modell), in: Angewandte Informatik, 30 (1988) 5, S. 191-202.

[26] Vgl. Wasserman, A.I.; Pircher, P.A.; Muller, R.J.: a.a.O., S. 57f.

vereinigt. Unabhängig davon kann die eigentliche Vorgehensweise zum Design offen bleiben.

5 Toolunterstützung

Ähnlich wie traditionelle Methoden um objektorientierte Konzepte erweitert werden, sind zur Zeit viele Anbieter von CASE-Systemen bemüht, ihre Tools um objektorientierte Methoden zu ergänzen. Hier ist, wie bei konventionellen Methoden auch, zu unterscheiden, ob der Rechner lediglich als (halbwegs intelligentes) Mittel zur Unterstützung der Diagrammerstellung genutzt werden kann, oder ob neben der Bereitstellung objektorientierter Notationen auch weitere Hilfestellungen (Konsistenzcheck, Diagrammtransformationen) angeboten werden.

Trotz der engen Bindung zwischen Design und Implementierung ist im Bereich der Toolunterstützung weiterhin eine deutliche Zweiteilung festzustellen: Programmiersprachenunabhängige CASE-Tools (siehe unten) ermöglichen die Anwendung oben beschriebener Designmethoden. Implementierungsnahe Werkzeuge dagegen, wie z. B. verschiedene Arten von Browsern zum "Durchstöbern" von Klassenhierarchien, sind bisher lediglich in Entwicklungsumgebungen einiger Programmiersprachen (z.B. Smalltalk, Actor) verfügbar. Hier stößt die theoretisch geforderte enge Verbindung dieser beiden Phasen bei der EDV-Unterstützung an die Grenzen ihrer praktischen Umsetzung. Gefordert werden muß also die Integration dieser beiden Abstraktionsebenen der Anwendungsentwicklung, was zum einen bedeutet, daß die Codierung auf der Basis von Designergebnissen unterstützt wird (mit zugehörigen Konsistenzchecks) und zum anderen die bisher nur zur Erforschung des Codes verwendeten Browser auch in gleicher Weise zur Verwaltung der Designergebnisse eingesetzt werden können.

Im folgenden wird eine (sicher unvollständige) Übersicht über CASE-Tools gegeben, die die oben erwähnten objektorientierten Designmethoden unterstützen:

Cadre Teamwork:

- Shlaer/Mellor (1988): Object-Oriented Systems Analysis[27]

---

[27] Vgl. Shlaer, S.; Mellor, S.J.: Object-Oriented Systems Analysis: Modeling the World in Data. Englewood Cliffs 1988.

- Booch (1986)[28], Buhr (1984, Structure Graphs)[29]

Coad/Yourdon Toolkit:

- Coad/Yourdon (1990): Object-Oriented Analysis

Object Maker (Mark V Systems):

- Booch (1986), Buhr (1984, Structure Graphs)

System Architect (Popkin Software):

- Booch (1986)
- Coad/Yourdon (1990) : Object-Oriented Analysis

6 Ausblick

Nur wenn es gelingt, sich in absehbarer Zeit auf eine einheitliche Darstellungsform objektorientierter Designergebnisse zu einigen, können die seit langem prophezeiten[30] und eng an die Objektorientierung geknüpften Produktivitätssprünge bei der Betonung der Wiederverwendung von Softwarekomponenten realisiert werden: Die Vorteile der Wiederverwendung kommen nur dann in vollem Maße zum Tragen, wenn ein überbetrieblicher Austausch von Softwarekomponenten (bzw. ein Vertrieb von domänspezifischen Bausteinbibliotheken) ermöglicht werden kann, der nicht bei innerbetrieblichen Standards an seine Grenzen stößt.[31]
Zum zweiten ist es für einen Erfolg dieses Vorhabens zwingend notwendig, die Wiederverwendung von Software nicht nur auf Code-Ebene zu betreiben, sondern Designergebnisse ganz eng an die implementierten Bausteine anzukoppeln, um die Suche des Anwendungsentwicklers nach geeigneten Komponenten auf dieser höheren abstrakteren Ebene zu ermöglichen, wozu dann natürlich auch entsprechende Retrievalwerkzeuge zur Verfügung stehen müssen.

---

[28] Vgl. Booch, G.: Object-Oriented Development, in: IEEE Transactions on Software Engineering, 12 (1986) 2, S. 211-221.

[29] Vgl. Buhr, R.J.A.: System Design with Ada. Englewood Cliffs 1984.

[30] Vgl. McIlroy, M.D.: Mass-produced Software Components, in: Buxton, J.M. et al. (Hrsg.): Software Engineering Concepts and Techniques, Proceedings of NATO Conference on Software Engineering 1968, S. 88-98.

[31] Vgl. Tsichritzis, D.; Gibbs, S.: Towards Integrated Software Communities. Centre Universitaire d'Informatique, Université de Genève 1990.

7 Literaturverzeichnis

Boehm, B.W.: Software Engineering Economics. Englewood Cliffs 1981.

Booch, G.: Object-Oriented Development, in: IEEE Transactions on Software Engineering, 12 (1986) 2, S. 211-221.

Booch, G.: Object-Oriented Design with Applications. Reading et al. 1991.

Buhr, R.J.A.: System Design with Ada. Englewood Cliffs 1984.

Coad, P.; Yourdon, E.: Object-Oriented Analysis. Englewood Cliffs 1990.

Ferstl, O.K.; Sinz, E.J.: Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM), in: Wirtschaftsinformatik, 32 (1990) 6, S. 566-581.

Fleischer, P. et al.: Der objektorientierte Software-Entwicklungsprozeß und seine Unterstützung durch Werkzeuge, in: Softwaretechnik-Trends, 11 (1991) 1, S. 23-53.

Heitz, M.: HOOD: Hierarchical Object-Oriented Design for large technical and realtime software. CISI Ingenierie, Direction Midi Pyrenees, 1987.

Henderson-Sellers, B.; Edwards, J.M.: The Object-Oriented Systems Life Cyle, in: Communications of the ACM, 33 (1990) 9, S. 142-159.

Lehner, F.: Wiederverwendbarkeit von Software aus der Sicht des Lebenszyklus-Modells, in: Softwaretechnik-Trends, 10 (1990) 1, S. 43-56.

Loomis, M.E.S.; Shah, A.V.; Rumbaugh, J.E.: An Object Modeling Technique for Conceptual Design, in: Proceedings of ECOOP '87, S. 192-202.

McIlroy, M.D.: Mass-produced Software Components, in: Buxton, J.M. et al. (Hrsg.): Software Engineering Concepts and Techniques, Proceedings of NATO Conference on Software Engineering 1968, S. 88-98.

Meyer, B.: Object-Oriented Software Construction. Hemel Hempstead 1988.

Meyer, B.: From Structured Programming to Object-Oriented Design: The Road to Eiffel, in: Structured programming, 10 (1989) 1, S. 19-39.

Pun, W.W.Y.; Winder, R.L.: A Design Method for Object-Oriented Programming, in: Proceedings of ECOOP '89, S. 225-240.

Scheer, A.-W.: Architektur integrierter Informationssysteme. Berlin et al. 1991.

Schlüter, P. et al.: Objektorientierte Software-Entwicklung: Konzepte und Terminologie, in: Softwaretechnik-Trends, 10 (1990) 2, S. 22-44.

Seidewitz, E.; Stark, M.: An Introduction to General Object-Oriented Software Develop-

ment. Rockville 1988.

Shlaer, S.; Mellor, S.J.: Object-Oriented Systems Analysis: Modeling the World in Data. Englewood Cliffs 1988.

Sinz, E.J.: Das Strukturierte Entity-Relationship-Modell (SER-Modell), in: Angewandte Informatik, 30 (1988) 5, S. 191-202.

Tsichritzis, D.; Gibbs, S.: Towards Integrated Software Communities. Centre Universitaire d'Informatique, Université de Genève 1990.

Wasserman, A.I.; Pircher, P.A.; Muller, R.J.: An Object-Oriented Structured Design Method for Code Generation, in: ACM SIGSOFT, Software Engineering Notes, 14 (1989) 1, S. 32-55.

Wasserman, A.I.; Pircher, P.A.; Muller, R.J.: The Object-Oriented Structured Design Notation for Software Representation, in: Computer, 23 (1990) 3, S. 50-63.

Wegner, P.: Learning the Language, in: BYTE, 14 (1989) 3, S. 245-253.

Wirfs-Brock, R.J.; Johnson, R.E.: Surveying current research in object-oriented Design, in: Communications of the ACM, 33 (1990) 9, S. 104-124.

Die Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi) im Institut für empirische Wirtschaftsforschung an der Universität des Saarlandes erscheinen in unregelmäßiger Folge.

* Die Hefte 1 - 31 werden nicht mehr verlegt.

Heft 32: A.-W. Scheer: Einfluß neuer Informationstechnologien auf Methoden und Konzepte der Unternehmensplanung, März 1982, Vortrag anläßlich des Anwendergespräches "Unternehmensplanung und Steuerung in den 80er Jahren in Hamburg vom 24. - 25.11.1981

Heft 33: A.-W. Scheer: Dispositon- und Bestellwesen als Baustein zu integrierten Warenwirtschaftssystemen, März 1982, Vortrag anläßlich des gdi-Seminars "Integrierte Warenwirtschafts-Systeme" in Zürich vom 10. - 12. Dezember 1981

Heft 34: J. Ahlers, W. Emmerich, H. Krcmar, A. Pocsay, A.-W. Scheer, D. Siebert: EPSOS - Ein Ansatz zur Entwicklung prüfungsgerechter Software-Systeme, Mai 1982

Heft 35: J. Ahlers, W. Emmerich, H. Krcmar, A. Pocsay, A.-W. Scheer, D. Siebert: EPSOS-D, Konzept einer computergestützten Prüfungsumgebung, Juli 1982

Heft 36: A.-W. Scheer: Rationalisierungserfolge durch Einsatz der EDV - Ziel und Wirklichkeit, August 1982, Vortrag anläßlich der 3. Saarbrücker Arbeitstagung "Rationalisierung" in Saarbrücken vom 04. - 06. 10.1982

Heft 37: A.-W. Scheer: DV-gestützte Planungs- und Informationssysteme im Produktionsbereich, September 1982

Heft 38: A.-W. Scheer: Interaktive Methodenbanken: Benutzerfreundliche Datenanalyse in der Marktforschung, Mai 1983

Heft 39: A.-W. Scheer: Personal Computing - EDV-Einsatz in Fachabteilungen, Juni 1983

Heft 40: A.-W. Scheer: Strategische Entscheidungen bei der Gestaltung EDV-gestützter Systeme des Rechnungswesens, August 1983, Vortrag anläßlich der 4. Saarbrücker Arbeitstagung "Rechnungswesen und EDV" in Saarbrücken vom 26. - 28.09.1983

Heft 41: H. Krcmar: Schnittstellenprobleme EDV-gestützter Systeme des Rechnungswesens, August 1983, Vortrag anläßlich der 4. Saarbrücker Arbeitstagung "Rechnungswesen und EDV" in Saarbrücken vom 26. - 28.09.1983

Heft 42: A.-W. Scheer: Factory of the Future, Vorträge im Fachausschuß "Informatik in Produktion und Materialwirtschaft" der Gesellschaft für Informatik e. V., Dezember 1983

Heft 43: A.-W. Scheer: Einführungsstrategie für ein betriebliches Personal-Computer-Konzept, März 1984

Heft 44: A.-W. Scheer: Schnittstellen zwischen betriebswirtschaftlicher und technische Datenverarbeitung in der Fabrik der Zukunft, Juli 1984

Heft 45: J. Ahlers, W. Emmerich, H. Krcmar, A. Pocsay, A.-W. Scheer, D. Siebert: EPSOS-D, Ein Werkzeug zur Messung der Qualität von Software-Systemen, August 1984

Heft 46: H. Krcmar: Die Gestaltung von Computer am-Arbeitsplatz-Systemen - ablauforientierte Planung durch Simulation, August 1984

Heft 47: A.-W. Scheer: Integration des Personal Computers in EDV-Systeme zur Kostenrechnung, August 1984

Heft 48: A.-W. Scheer: Kriterien für die Aufgabenverteilung in Mikro-Mainframe Anwendungssystemen, April 1985

Heft 49: A.-W. Scheer: Wirtschaftlichkeitsfaktoren EDV-orientierter betriebswirtschaftlicher Problemlösungen, Juni 1985

Heft 50: A.-W. Scheer: Konstruktionsbegleitende Kalkulation in CIM-Systemen, August 1985

Heft 51: A.-W. Scheer: Strategie zur Entwicklung eines CIM-Konzeptes - Organisatorische Entscheidungen bei der CIM-Implementierung, Mai 1986

Heft 52: P. Loos, T. Ruffing: Verteilte Produktionsplanung und -steuerung unter Einsatz von Mikrocomputern, Juni 1986

Heft 53: A.-W. Scheer: Neue Architektur für EDV-Systeme zur Produktionsplanung und -steuerung, Juli 1986

Heft 54: U. Leismann, E. Sick: Konzeption eines Bildschirmtext-gestützten Warenwirtschaftssystems zur Kommunikation in verzweigten Handelsunternehmungen, August 1986

Heft 55: D. Steinmann: Expertensysteme (ES) in der Produktionsplanung und -steuerung (PPS) unter CIM-Aspekten, November 1987, Vortrag anläßlich der Fachtagung "Expertensysteme in der Produktion" am 16. und 17.11.1987 in München

Heft 56: A.-W. Scheer: Enterprise wide Data Model (EDM) as a Basis for Integrated Information Systems, Juli 1988

Heft 57: A.-W. Scheer: Present Trends of the CIM Implementation (A qualitative Survey) Juli 1988

Heft 58: A.-W. Scheer: CIM in den USA - Stand der Forschung, Entwicklung und Anwendung, November 1988

Heft 59: R. Herterich, M. Zell: Interaktive Fertigungssteuerung teilautonomer Bereiche, November 1988

Heft 60: A.-W. Scheer, W. Kraemer: Konzeption und Realisierung eines Expertenunterstützungssystems im Controlling, Januar 1989

Heft 61: A.-W. Scheer, G. Keller, R. Bartels: Organisatorische Konsequenzen des Einsatzes von Computer Aided Design (CAD) im Rahmen von CIM, Januar

1989

Heft 62: M. Zell, A.-W. Scheer: Simulation als Entscheidungsunterstützungsinstrument in CIM, September 1989

Heft 63: A.-W. Scheer: Unternehmens-Datenbanken - Der Weg zu bereichsübergreifenden Datenstrukturen, September 1989

Heft 64: C. Berkau, W. Kraemer, A.-W. Scheer: Strategische CIM-Konzeption durch Eigenentwicklung von CIM-Modulen und Einsatz von Standardsoftware, Dezember 1989

Heft 65: A. Hars, A.-W. Scheer: Entwicklungsstand von Leitständen[1], Dezember 1989

Heft 66: W. Jost, G. Keller, A.-W. Scheer: CIMAN - Konzeption eines DV-Tools zur Gestaltung einer CIM-orientierten Unternehmensarchitektur, März 1990

Heft 67: A.-W. Scheer: Modellierung betriebswirtschaftlicher Informationssysteme (Teil 1: Logisches Informationsmodell), März 1990

Heft 68: W. Kraemer: Einsatzmöglichkeiten von Expertensystemen in betriebswirtschaftlichen Anwendungsgebieten, März 1990

Heft 69: A.-W. Scheer, R. Bartels, G. Keller: Konzeption zur personalorientierten CIM-Einführung, April 1990

Heft 70: St. Spang, K. Ibach: Zum Entwicklungsstand von Marketing-Informationssystemen in der Bundesrepublik Deutschland, September 1990

Heft 71: D. Aue, M. Baresch, G. Keller: **URMEL**, Ein **U**nte**R**nehmens**M**od**EL**lierungsansatz, Oktober 1990

Heft 72: M. Zell: Datenmanagement simulationsgestützter Entscheidungsprozesse am Beispiel der Fertigungssteuerung, November 1990

Heft 73: A.-W. Scheer, M. Bock, R. Bock: Expertensystem zur konstruktionsbegleitenden Kalkulation, November 1990

Heft 74: R. Bartels, A.-W. Scheer: Ein Gruppenkonzept zur CIM-Einführung, Januar 1991

Heft 75: M. Nüttgens, St. Eichacker, A.-W. Scheer: CIM-Qualifizierungskonzept für Klein- und Mittelunternehmen (KMU), Januar 1991

Heft 76: Ch. Houy, J. Klein: Die Vernetzungsstrategie des Instituts für Wirtschaftsinformatik - Migration vom PC-Netzwerk zum Wide Area Network (noch nicht veröffentlicht)

Heft 77: W. Kraemer: Ausgewählte Aspekte zum Stand der EDV-Unterstützung für das Kostenmanagement: Modellierung benutzerindividueller Auswertungssichten in einem wissensbasierten Controlling-Leitstand, Mai 1991

Heft 78: H. Heß: Vergleich von Methoden zum objektorientierten Design von Softwaresystemen, August 1991

Heft 79: A.-W. Scheer: Konsequenzen für die Betriebswirtschaftslehre aus der Entwicklung der Informations- und Kommunikationstechnologien, Mai 1991