Objektorientierte Systemanalyse und Prototyping

Object-oriented Systemanalyses and Prototyping

SUMMARY

The article has three goals: 1. To emphasize system construction as the decision with most consequences in software development and to show solutions; 2. to demonstrate the all over applicability of object oriented techniques; 3. to underline the necessity of a unique view of software and data engineering and to emphasize, that a change of paradigmes in software engineering is overdue. The most important statements: Systems have to be constructed, they don't 'fall from heaven'. A top down design of software systems is impossible: hierarchies of data (inheritance!) grow bottom up by abstraction. Transformation oriented methods (SA, SADT, RSL, ISAC etc.) are too expensive and cause breaks in structures: therefore they have to be 'garbaged' from teaching and application.

STICHWORTE

Objektorientierte Analyse (OOA), SE-Methoden, AD/Cycle, Systembildung, Software Engineering, Prototyping.

KEYWORDS

Object-oriented analysis (OOA), SE methods, AD/Cycle, system construction, software engineering, prototyping.

Der Beitrag hat drei Ziele: 1. Die Systembildung als folgenreichste konstitutive Entscheidung bei der Softwareentwicklung hervorzuheben und Lösungen aufzuzeigen; 2. die Durchgängigkeit objektorientierter Techniken zu demonstrieren; 3. die Notwendigkeit einer einheitlichen Sicht von Software und Data Engineering zu unterstreichen und damit deutlich zu machen, daß ein Paradigmenwechsel im Software Engineering überfällig ist. Die wichtigsten Aussagen: Systeme sind zu konstruieren, sie „fallen nicht vom Himmel". Ein Top-down-Entwurf von Softwaresystemen ist nicht möglich: Hierarchien von Daten (Vererbung!) entstehen bottom up durch Abstraktion. Transformationsorientierte Spezifikationsmethoden (SA, SADT, RSL, ISAC und andere) sind zu aufwendig und führen zu Strukturbrüchen: Sie sind daher aus Lehre und Anwendung zu „entsorgen".

1. DAS PROBLEM: SOFTWARE-ENGINEERING-METHODEN

Methoden sind das wichtigste Thema des Software Engineering (SE), weil Werkzeuge als die andere Seite des SE immer Methoden implizieren. Methoden sind gewissermaßen die Spezifikation der Werkzeuge. Der folgende Beitrag behandelt Analyse und Entwurf administrativer Systeme. Das hier skizzierte Vorgehensmodell ist entstanden und wurde angewendet in über 5 Jahren Entwicklung von über 12 großen Softwaresystemen, die alle in Betrieben eingesetzt sind oder sich in der Einführung befinden. Das Entwicklungsvolumen der letzten 4 Jahre von über 44 MJ zuzüglich 19 MJ Wartungstätigkeit ist in einer Datenbank auswertbar dokumentiert. Auf dieser Basis behauptet der Autor, eine empirisch fundierte Technologie zu empfehlen und weiß aus eigener Erfahrung, daß Entscheidungen für strategische Softwarewerkzeuge Fehlentscheidungen werden müssen, wenn man die von den Werkzeugen zugrundegelegten Methoden nicht vorher kritsch geprüft hat. Dies hat im Rahmen der IBM-Ankündigungen zu AD/Cycle ein großes Gewicht. Besonders hohe Methoden-Defizite bestehen bei der Bildung und Abgrenzung von Systemen.

Die Struktur eines Softwaresystems entsteht nicht erst beim Softwareentwurf, sondern vor Beginn der Spezifikation. Außer dem Verfahren JSD von Jackson [1] war bis Ende 1988 kein Vorgehensmodell bekannt, das diesen Schritt explizit vorsah. Anfang 1989 erschien das Modell OBAS des Autors [2] (Objektorientierter Entwurf Administrativer Systeme).

Mitte 1990 folgte OOA von Coad und Yourdon [3], danach [4] von Scheer. In Abbil-

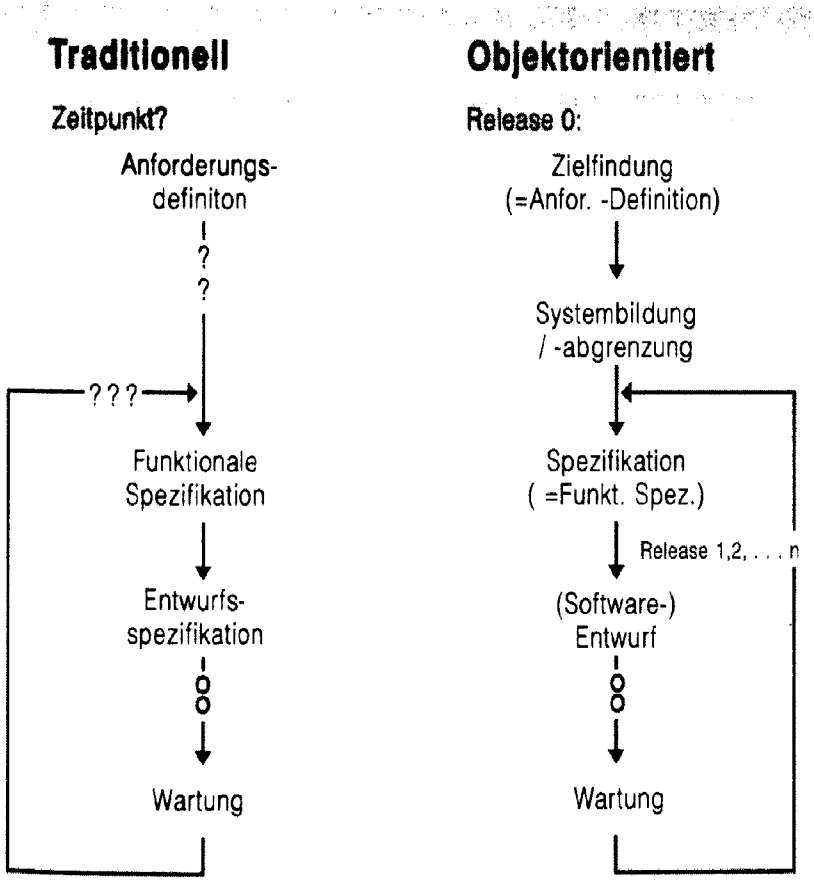

Abb.1: Traditionelles versus objektorientiertes Vorgehensmodell

dung 1 werden das traditionelle und ein objektorientiertes Vorgehensmodell bezüglich der frühen Phasen gegenübergestellt.

Warum ist das alternative, zyklische Vorgehensmodell objektorientiert und das „traditionell" genannte nicht? Beim traditionellen Modell werden Systeme intuitiv gebildet und in Teilsysteme zerlegt. Danach werden sie (günstigstenfalls!) vor der Implementierung in einer Phase „DV-Technik" völlig neu strukturiert und objektorientiert implementiert. In der Wartung werden diese Strukturen zwangsläufig zerstört, da die Spezifikation nicht mit der Softwarearchitektur zusammengebracht werden kann (Einzelheiten in [2], Kapitel 3).

2. DIE REALITÄT, ODER: VON SA ZU AD/CYCLE

Alle gängigen Methoden des Software Engineering schlagen im Prinzip folgende Vorgehensweise vor: Funktionen beziehungsweise Prozesse werden gesammelt und geordnet. Die Ordnung erfolgt meist top down im Zuge einer schrittweisen Verfeinerung. Entweder werden die Daten beim Darstellen der Funktionen mitbetrachtet oder die Funktionen werden als Funktionshierarchie gesehen und die Daten als Datenhierarchie (in SADT oder in JSD). Ein Rückgriff auf die Methoden zur Datenmodellierung (Relationenmodell, Entity-Relationship-Methode) findet nicht statt oder sie werden an „alte" Methoden angehängt wie zum Beispiel in AD/Cycle ([5] und [6]). Als Beispiel für eine Funktionshierarchie sei auf das „Bibliotheksbeispiel" von Schulz ([7], S. 37) verwiesen. Eine hierzu passende „Datenhierarchie" (siehe auch Vetter [6]) würde wie in Abbildung 2 dargestellt aussehen.

Die traditionellen Methoden modellieren allesamt einen Daten- oder Steuerfluß nach dem primitiven Schema einer erweiterten Wertzuweisung (E-V-A Prinzip). Bei SADT und bei ISAC ist ein Verfeinerungsmechanismus eingebaut, der die Darstellung bis etwa zur dritten Verfeinerungsebene besser handhabbar und übersichtlicher macht als bei SA oder RSL. Jedoch wehe dem, der zu tief verfeinert! 30 Diagramme auf Ebene 3 erzeugen 120 bis 200 Diagramme auf Ebene 4. Damit kein Mißverständnis entsteht: Als Darstellungstechnik ist eine Methode wie SADT hervorragend geeignet, nicht jedoch für einen Entwurf.

Die funktionsorientierte Entwurfstechnik findet sich noch überwiegend in industriell genutzten Systemen, da bei einem intuitiven, nicht explizit durchgeführten Softwareentwurf praktisch immer solche Strukturen entstehen. Explizit entstehen diese Strukturen bei Verwendung der Methoden *structured design* (SD) und *structured analysis* (SA), die in vielen Derivaten existieren. Auch AD/Cycle basiert auf SA. Diese Methoden sind aufwendig und bei ▶

Objektorientierte Systemanalyse und Prototyping

Anwendung auf reale Entwicklungen kaum überschaubar. Dem Verfasser sind Dokumente von weit mehr als 1000 Seiten zugänglich, die mit dem Werkzeug IEW erzeugt wurden.

3. DIE ZUKUNFT: OBJEKTORIENTIERTE METHODEN

Der Entwurf von Software folgt zwei prinzipiell möglichen Ansätzen:
- dem funktions- oder datenflußorientierten,
- dem datentyp- oder auch objektorientierten.

Die beiden grundlegenden Paradigmen Datenfluß oder Datentyp wurden schon 1972 von Parnas in ihren Auswirkungen auf die Wartbarkeit von Software diskutiert [8].

Danach besteht objektorientierte Software aus Moduln, die Operationen auf Datenstrukturen bereitstellen. Moduln interagieren durch Benutzung der Operationen anderer Moduln. Datenstrukturen repräsentieren Datenobjekte, die temporär im Hauptspeicher oder dauerhaft auf Datenträgern gehalten werden.

Die Klammerung von Datentyp und Operationen ist grundlegend für eine Objektorientierung, wenn auch nicht hinreichend (Näheres siehe unten). Sie liegt dem Ansatz von Parnas zugrunde und ist in der „Datenbankwelt" so selbstverständlich, daß reale Datenbanksysteme eigentlich nur nach diesem Prinzip konstruiert sind. Die Methoden der Datenanalyse (siehe auch [9]) sind hervorragend geeignet für eine objektorientierte Systemanalyse, man muß sie nur weitergehend zur reinen Datenmodellierung nutzen. Der Unterschied zwischen transformationsorientierten und objektorientierten Methoden ist noch einmal in Abbildung 3 dargestellt.

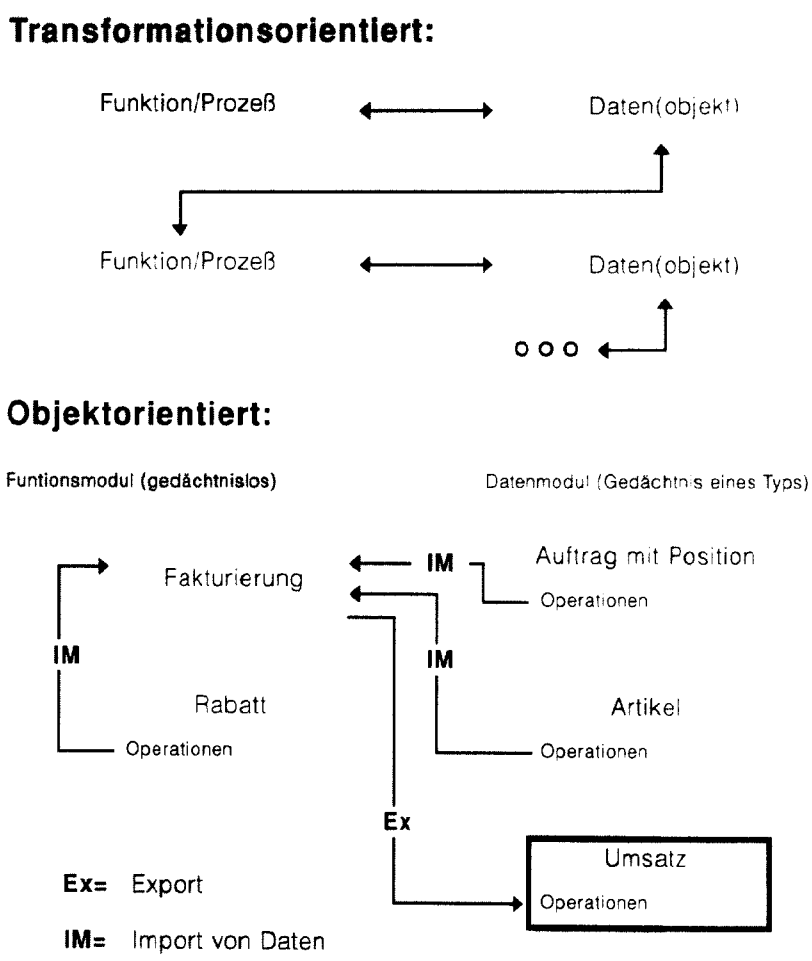

Abb. 3: Transformationsorientierte versus objektorientierte Systeme

Nach diesem Schema „erzeugt" ein Funktionsmodul niemals Daten, obwohl er Daten exportieren kann, wie in diesem Beispiel abgeleitete Daten. *Fakturierung* bedient sich dabei der Operation *Archivieren* des Moduls *Umsatz*.

Während die Bindung von Operation und Datentyp von objektorientierten Sprachen unterstützt wird, muß sie bei der Analyse durch Methoden geschaffen werden. Die Schlüsselfrage einer objektorientierten Analyse muß sein:

Wer (in der Organisation) erzeugt (= Operationen) welche (primären) Daten? Bei einer objektorientierten Analyse müssen die datenerzeugenden Operationen zu den Objekttypen (= Entities) gefunden werden, die primäre Daten beinhalten. Dies sind Stammdaten und Vorgangsdaten. Sekundäre (oder auch abgeleitete) Daten sind meist verdichtete Vorgänge,

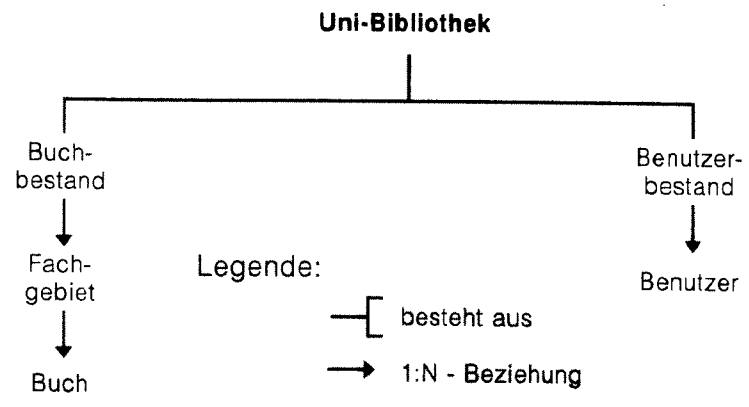

Abb. 2: Datenhierarchie zu einer gegebenen Funktionshierarchie

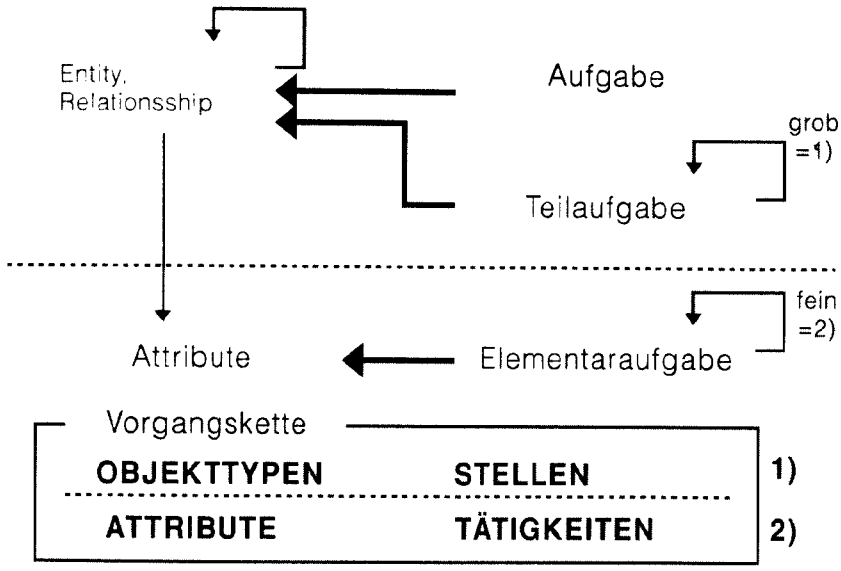

Abb.6: Schema der Systembildung mit Vorgangsketten

mit denen man eine Systembildung nicht belasten sollte.

Für eine echte Objektorientierung fehlt noch ein Vererbungsmechanismus. Das Vererbungsprinzip ist in der Datenbankwelt als Abstraktion, Generalisierung oder auch Aggregation bekannt. Datenanalysemethoden benutzen den Begriff Vererbung nicht oder wenig (so auch in [2]), da er den Kern des Problems in einer Analysephase nicht trifft: „Vererbung" suggeriert Hierarchie und Top-down-Entwurf (siehe Abb. 2). Der Entwurf von Datenmodellen geschieht jedoch fast immer bottom up durch Abstraktion. Dies ist das Gegenteil einer Verfeinerung.

Es genügt nicht, einer Funktionsanalyse ein Entity-Relationship-Modell anzuhängen. Man muß vielmehr in der Analyse die datenerzeugenden Operationen zu den Objekttypen finden. Das Vorgehensmodell hierzu wurde aus der Erfahrung vieler Softwareprojekte in der Schering AG und der Vatter Gruppe entwickelt.

4. OBJEKTORIENTIERTE ANALYSE ZUR SYSTEMBILDUNG

Die Konstruktion von Softwaresystemen ist als Phase Systembildung beziehungsweise -abgrenzung *vor* der Spezifikation einzuordnen (siehe Abb. 1). Es muß folgendes geleistet werden:

- Objekttypen für die relevanten primären Daten finden,
- verantwortliche Benutzer finden, die diese Daten erzeugen,
- je Stelle die erzeugenden Operationen auf Typen und Attributen feststellen.

Ein objektorientiertes System besteht aus Objekttypen einer zentralen Datenbasis, die zunächst nur aus primären Daten besteht. Das System ist eingebettet in diejenige Organisationseinheit, die die Objekte der Objekttypen erzeugt. Die objektorientierte Analyse von OBAS zerfällt in einen groben Teil und einen detaillierten:

1. *Grobentwurf:* Finden von Objekttypen nach dem Entity-Relationship-Modell (graphisch) mit Hilfe von Begriffsdefinitionen. Erstellen eines Aufgabenmodells (Tätigkeiten von Stellen bei der Erzeugung und Pflege der Objekte).
2. *Feinentwurf:* Ergänzen der Attribute nach dem Relationenmodell mit Normalisierung (verbal). Notieren der erzeugenden Operationen (= Elementaraufgaben).

Man beachte, daß man bei der Bildung von Objekttypen immer wichtige Attribute wenigstens implizit mitbetrachtet, da Objekttypen erst durch ihre Attribute definiert sind.

Ziel der Vorgehensweise ist eine Systembildung und -abgrenzung mit 1) wenig Aufwand und 2) so präzise wie nötig – zu sehen in Abbildung 6.

Diese Methode zur Konstruktion von Softwaresystemen bricht mit einem verbreiteten Paradigma des Software Engineering: Es wird nicht top down entworfen. Der Autor bezweifelt entschieden, daß es möglich ist, Datenmodelle top down zu entwerfen, wie dies auch in neuerer Zeit noch behauptet wird (vergleiche Unterlagen zu AD/Cycle; aber auch: Jackson [1], Vetter [6]). Zur Erläuterung sei ein Teil des Beispiels aus Abbildung 2 als Relationenmodell notiert:

BUCHBESTAND	(Wirtschaftsjahr#, Anzahl-Titel, Bestands-Wert)
FACHGEBIET	(Wirtschaftsjahr#, Gebiets#, Gebietsname, Anzahl-Titel, Bestands-Wert)
BUCH	(Inventar#, Gebiets#, Autor, Titel, Verlag, ..., Preis)

In der Hierarchie von Abbildung 2 fehlt die wichtigste Relation, weil man sie bei einem top-down-Entwurf nicht findet:

AUSLEIHE	(Inventar#, Benutzer#, Ausleih-Datum, Rückgabe-Datum,...)

Die in der Hierarchie höher gelegenen Relationen enthalten nur abgeleitete Daten. Sie vererben nichts auf die Elementarrelation BUCH, im Gegenteil: sie erben (= bottom up) primäre Daten in verdichteter Form. Die hierzu erforderlichen Operationen gehören zum hierarchisch höher gelegenen Objekt und nicht zum tiefer gelegenen.

Man sollte also bei Analogien zwischen objektorientierter Analyse und objektorientierten Sprachen vorsichtig sein. Viele Hierarchien entstehen nicht durch Top-down-Entwurf, sondern durch Abstraktion. Oft wird bei der Rede vom Top-down-Entwurf Produkt und Entwurf verwechselt.

5. PROTOTYPING UND SOFTWAREENTWURF

Prototypen sind nach der Systembildung sehr sinnvoll bei der Spezifikation einsetzbar. Sie dienen als Kommunikationshilfe bei Abstimmungen mit dem Benutzer, den geschriebene Spezifikationen meist nicht erreichen, schon gar nicht Papierberge von „Detailfunktionen" (vergleiche [5]). Zur Konstruktion von Systemen sind Prototypen jedoch gänzlich ungeeignet. Prototyping hat nichts mit „quick and dirty" zu tun (mehr in [2]).

Für einen Systementwurf braucht man ▶

Objektorientierte Systemanalyse und Prototyping

zunächst Skizzenblock und Bleistift, eine Tafel, Diskussionspartner und vor allem: Kopf. Eine gute, das heißt wartbare Softwarearchitektur, basierend auf einer objektorientierten Analyse, ist so ziemlich das Anspruchsvollste, was bei der Softwareentwicklung zu tun ist (siehe auch [10], [11]). Nur auf Basis einer wohl durchdachten Architektur kann man daran denken, aus Prototypen inkrementell Echtsysteme zu entwickeln, wie dies seit Jahren – allerdings nur bei einfachen Systemen – in der Vatter Unternehmensgruppe erfolgreich und kostengünstig geschieht.

Dr.-Ing. habil. Thorsten Spitta ist Leiter Organisation und Information der Vatter Unternehmensgruppe, Augsburg, Schongau und Rheine: Vatter GmbH, Postfach 2165, W-4440 Rheine.

Literatur:

[1] Jackson, M.A.: System Development. Prentice Hall, Hempsteadt 1983.

[2] Spitta, Th.: Software Engineering und Prototyping. Springer, Berlin – Heidelberg – New York et.al. 1989.

[3] Coad, P.; Yourdon, E.: Object Oriented Analysis. Prentice Hall, Englewood Cliffs 1990.

[4] Scheer, A.W.: Architektur integrierter Informationssysteme, Springer, Berlin – Heidelberg – New York et.al. 1991.

[5] Beetz, J.; Lambers, H.: Eine Anwendungs-Entwicklungsmethodik für AD/CYCLE. Addison-Wesley, Bonn – München – Paris et.al. 1991.

[6] Vetter, M.: Aufbau betrieblicher Informationssysteme, 6.Aufl. Teubner, Stuttgart 1990.

[7] Schulz, A.: Software-Entwurf, 2.Aufl. Oldenbourg, München – Wien 1990.

[8] Parnas, D.: On the Criteria to be Used in Decomposing Systems into Modules. Comm. ACM 15(1972) No.12, pp.1052-1058.

[9] Ortner, E.: Semantische Modellierung – Datenbankentwurf auf der Ebene der Benutzer, in: Informatik-Spektrum 8(1985) H.1, S.20-28.

[10] Denert, E.: Software-Engineering. Springer, Berlin – Heidelberg – New York et.al. 1991.

[11] Nagl, M.: Softwaretechnik: Methodisches Programmieren im Großen. Springer, Berlin – Heidelberg – New York et.al. 1990.