

Ein Nachdruck aus OBJEKTSpektrum 6/97

# Ein Leitfaden für die objektorientierte Anwendungsentwicklung in der Sparkassenorganisation

Die Einführung der objektorientierten Anwendungsentwicklung darf nicht auf die Wahl einer objektorientierten Programmiersprache reduziert werden. Am Beispiel eines aktuell entwickelten Leitfadens für die objektorientierte Anwendungsentwicklung in der Sparkassenorganisation werden die vielfältigen Themenbereiche für die vollständige Abdeckung des objektorientierten Entwicklungszyklus verdeutlicht. Der Beitrag zeigt, welchen Herausforderungen und Fragestellungen sich eine große Organisation bei der Migration zur Objekttechnologie aus Sicht der Anwendungsentwicklung stellen muß.

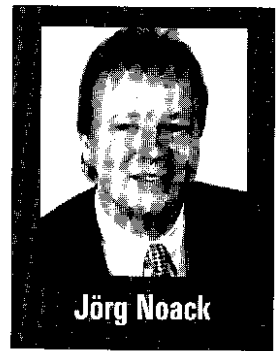
In den Landesbanken und in den Verbandsrechenzentren der Sparkassenorganisation werden objektorientierte Anwendungsentwicklungsprojekte seit etwa vier Jahren durchgeführt. Diese Entwicklungsprojekte zeigten einerseits den Nutzen der Objekttechnologie, machten andererseits aber auch deutlich, daß für eine systematische objektorientierte Anwendungsentwicklung in einer großen Organisation ein umfassendes methodisches Rahmenkonzept notwendig ist.

Auf der Grundlage der in den OO-Projekten der Sparkassenorganisation gemachten Erfahrungen (vgl. zusammenfassend [Kar97]) wurde ein gemeinsamer Leitfaden für die objektorientierte Anwendungsentwicklung im Rahmen eines Projekts des *Informatikzentrums der Spar-*

*kassenorganisation (SIZ)* erstellt. Der Leitfaden dient den DV-Entwicklungseinheiten der Sparkassenorganisation als Orientierungshilfe für OO-Projekte. Er fördert Kooperationsprojekte und erleichtert den Austausch von Anwendungen und Softwarekomponenten.

Der Leitfaden deckt den objektorientierten Entwicklungszyklus vollständig und durchgängig ab. Seine Kernkonzepte (vgl. Abb. 1) sind:

- Die *Modellierung* definiert einen sogenannten Modellierungskern aus Modell- und Diagrammtypen, angelehnt an die „Unified Modeling Language“ (UML), sowie Richtlinien für die Übergänge von der Geschäftsprozeß- zur Objektmodellierung und zur Programmierung.
- Im Kernkonzept *Architektur* wird die Entwicklung einer Softwarearchitektur für verteilte Anwendungen betrachtet. Dabei werden Schichtenbildung, Verteilungsmodell und Persistenz (Anbindung relationaler Datenbanksysteme) behandelt.
- Die *Programmierung* enthält Richtlinien für die Implementierung einer Anwendung in Smalltalk und C++ sowie Empfehlungen für die Fehlerbehandlung und die Gestaltung der Benutzungsschnittstelle. Eine Ergänzung für Java ist in Arbeit.



Jörg Noack



Bruno Schienmann



Hans-Bernd Kittlaus

- Im Kernkonzept *Werkzeuge* wird die Architektur einer objektorientierten Entwicklungsumgebung beschrieben. Ferner werden Kriterien für die Auswahl einzelner Komponenten definiert.
- Die vier genannten Kernkonzepte werden durch das *Vorgehensmodell* miteinander verknüpft. Das Vorgehensmodell ist als Referenzmodell definiert, das an verschiedene Projekttypen angepaßt werden kann.

Diese inhaltliche Gliederung des Leitfadens orientiert sich an den unterschiedlichen Aufgaben und Rollen in der objektorientierten Anwendungsentwicklung. Der Leitfaden ist so aufgebaut, daß die einzelnen Bestandteile auch isoliert voneinander betrachtet werden können. So sind für einen Entwickler, der vor einer konkreten Kodierungsaufgabe in C++ steht, zunächst nur die C++-Programmerrichtlinien interessant. Die Vorteile des OO-Ansatzes – wie Durchgängigkeit und Nahtlosigkeit – le-

Dr. Jörg Noack (E-Mail: joerg.noack@siz.de) ist Mitarbeiter im Modellierungskompetenzzentrum des Informatikzentrums der Sparkassenorganisation (SIZ). Er ist verantwortlich für den Aufbau und die Weiterentwicklung des Anwendungsentwicklungsmodells.

Dr. Bruno Schienmann (E-Mail: bruno.schienmann@siz.de) ist ebenfalls Mitarbeiter im Modellierungskompetenzzentrum des SIZ. Er ist verantwortlich für das Thema Objektmodelle.

Hans-Bernd Kittlaus (E-Mail: kittlaus@acm.org) ist Bereichsleiter im Modellierungskompetenzzentrum im SIZ Bonn.

gen jedoch auch eine durchgängige und in den einzelnen Themen aufeinander abgestimmte Beschreibung des Leitfadens nahe.

Der vorliegende Beitrag erläutert ausführlich die Strukturierung sowie die Inhalte des Leitfadens und orientiert sich dabei an den oben genannten fünf Kernkonzepten. Sicherlich muß jedes Unternehmen, das vor der Aufgabe steht, den objektorientierten Ansatz in der Anwendungsentwicklung einzuführen, abhängig von unternehmensspezifischen Gegebenheiten wie vorhandenen Know-how oder Infrastruktur die hier vorgeschlagenen Lösungen weiter detaillieren. Als Referenz im Sinne von Empfehlung, Bezugnahme und Musterbeispiel – hilft dieser Beitrag, den Umfang, die Struktur und die Inhalte eines eigenen, unternehmensspezifischen Rahmenkonzepts festzulegen.

## Modellierung

Das Kernkonzept „Modellierung“ behandelt die Analyse und das Design innerhalb der Entwicklungsmethodik. Dabei wird ein sogenannter Modellierungskern mit Modell- und Diagrammtypen definiert. Außerdem werden die Übergänge vom Geschäftsprozeß zum Objektmodell und vom Objektmodell zum objektorientierten Programm betrachtet.

### Modellierungskern

Auf der Basis eines Vergleichs weit verbreiteter objektorientierter Analyse- und Designmethoden und Entwurfssprachen – wie der „Object Modeling Technique 2“ (vgl. [Rum91] und [Rum95]), „Booch-93“

(vgl. [Boo93]), „Object-Oriented Software Engineering“ (vgl. [Jac92]) und insbesondere der „Unified Modeling Language (UML)“ (vgl. [UM197]) – wurde für den Leitfaden ein Kern praxisbewährter Modell- und Diagrammtypen für die Anwendungsspezifikation bestimmt.

Diese Auswahl hatte das Ziel, die Anzahl der Modell- und Diagrammtypen möglichst gering zu halten. Die ausgewählten Ausdrucksmittel sollten jedoch den Bedarf der verschiedenen am Modellierungsprozess beteiligten Personen in ihren unterschiedlichen Rollen und Sichten möglichst redundanzfrei abdecken. Motiviert durch den Standardisierungsvorschlag bei der OMG und die Tatsache, daß alle größeren Werkzeughersteller die UML unterstützen (werden), wird für den Modellierungskern grundsätzlich die UML-Notation empfohlen. In der Projektarbeit wurde aber auch deutlich, daß die Version 1.0 der UML in einigen Bereichen – beispielsweise für Architekturbeschreibungen oder hinsichtlich der Rollenmodellierung – Schwachpunkte aufweist. Der Modellierungskern umfaßt folgende Modelle (in Klammern ist der zur Modellbeschreibung empfohlene Diagrammtyp der UML angegeben):

- **Klassenstrukturmodell (Class Diagram):** Das Klassenstrukturmodell dient zur Beschreibung der statischen Struktur der Klassen und ihrer Beziehungen untereinander.
- **Interaktionsmodell (Sequence Diagram):** Interaktionsmodelle definieren die Reihenfolge der Objektinteraktionen zur Erbringung einer bestimmten Funktionalität.

- **Use-Case-Modell (Use Case Diagram):** Use-Case-Modelle stellen die einzelnen Anwendungsfälle mit ihren Beziehungen untereinander und zu den Akteuren dar.
- **Zustandsmodell (State Transition Diagram):** Zustandsmodelle dienen der Spezifikation der typspezifischen Lebenszyklen von Objekten.
- **Komponentenmodell (Component Diagram):** Das Komponentenmodell definiert die Abhängigkeiten zwischen verschiedenen Modulen.
- **Verteilungsmodell (Deployment Diagram):** Verteilungsmodelle beschreiben die physischen Strukturen der Anwendung mit Verarbeitungsknoten und ihren Beziehungen.

Die wesentlichen Zusammenhänge zwischen den unterschiedlichen Modellen des Modellierungskerns sind – nach Modellsichten geordnet – in Abb. 2 dargestellt.

- Interaktionsmodelle beschreiben die verschiedenen Szenarien der in den Use-Case-Modellen festgelegten Anwendungsfälle.
- Die Objekte in Interaktionsmodellen sind Exemplare der Klassen im Klassenstrukturmodell. Relationen/Botschaften sind durch Beziehungen/Operationen im Klassenstrukturmodell abzubilden.
- Zustandsmodelle beschreiben die Lebenszyklen der Objekte von Klassen mit besonders komplexem Verhalten.
- Akteure in einem Use-Case-Modell sind Klassenkandidaten im Klassenstrukturmodell.
- Das Klassenstrukturmodell ist Grundlage für die Bestimmung der logischen und physischen Anwendungsarchitektur. Es beeinflusst die Verteilung und Allokation der Komponenten.
- Die Hardware-, Prozeß- und Modulstruktur in der Implementierungssicht werden durch Komponenten- bzw. Verteilungsmodelle beschrieben.

Für die Anwendung der UML in der Modellierung wird auf die einschlägige Fachliteratur (wie [Bur97]) verwiesen. Um den Austausch von Ergebnissen zwischen Projekten zu ermöglichen – aber auch um die Zusammenarbeit in größeren Projekten zu erleichtern –, sind zusätzliche Gestaltungsregeln und Konventionen für eine einheitliche Begriffsbildung und Namensgebung notwendig. Für die einzelnen Modellelemente des Modellierungskerns wurden deshalb Dokumentationsrichtlinien mit verpflichtenden und optionalen

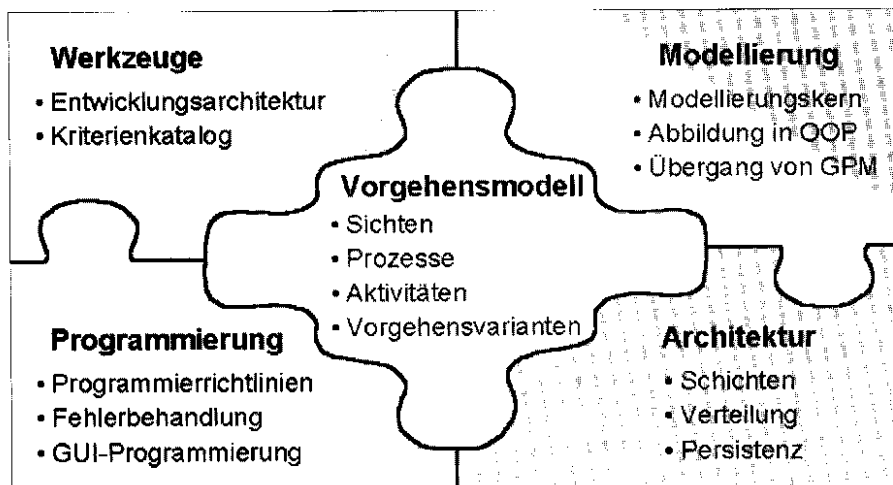


Abb. 1: Kernkonzepte des OO-Leitfadens

Beschreibungsattributen sowie Namenskonventionen definiert. So sind beispielsweise Elemente des Typs „Operation“ klein zu schreiben und durch Beschreibungsattribute – wie *Name*, *Definition*, *Sichtbarkeit*, *Parameter*, *Rückgabtyp*, *Vorbedingung*, *Nachbedingung*, *Virtualität* und *Zeitdauer* – zu dokumentieren.

## Übergang von der Geschäftsprozessmodellierung in die objektorientierte Modellierung

Vor allem durch die wachsende Anzahl an *Business-Process-Reengineering*-Projekten stellt die Geschäftsprozessmodellierung ein wichtiges Instrument dar, um DV-Anforderungen zu formulieren. Obwohl inzwischen auch Ansätze für eine objektorientierte Geschäftsprozessmodellierung entwickelt wurden (vgl. etwa [Jac94] oder [Kue96]), wird der überwiegende Teil nach strukturierten Ansätzen entwickelt. Weit verbreitet ist die Verwendung von *erweiterten Ereignisgesteuerten Prozessketten (eEPK)* (vgl. [Sch96]). Um solche Prozessmodelle mit ihrer Trennung von Daten und Funktionen für die objektorientierte Modellierung zu nutzen, wurde ein mehrstufiges Konzept für die Überführung in ein initiales Objektmodell auf der Basis einer *Use-Case-Analyse* entwickelt.

Ein mehrstufiges Vorgehen ist empfehlenswert, da viele Prozessmodelle aus Sicht der Anwendungsentwicklung erfahrungsgemäß einige Schwächen aufweisen und daher keinen direkten Übergang zu einem Objektmodell erlauben (vgl. [Bun95]). In solchen Prozessmodellen wird beispielsweise häufig nicht systematisch zwischen bankfachlichen und dv-technischen

Aspekten unterschieden: Z. B. müssen Funktionen in einer Prozesskette wie „Text mit Word bearbeiten“ zunächst bankfachlich rekonstruiert werden, bevor sie in die Nachricht „erstellen“ eines Aktors „Produktbetreuer“ an ein Objekt „Angebot“ im *Use-Case*- oder Interaktionsmodell überführt werden können. Auch findet in Prozessmodellen in vielen Fällen keine vollständige und durchgängige Modellierung statt, die vom Kunden ausgeht und zum Kunden hin führt. Häufig müssen bereits dokumentierte Prozesse vor der Überführung in *Use-Cases* bzw. in Szenarien zunächst fachlich vervollständigt und weiter detailliert werden.

## Übergang von der Modellierung zur objektorientierten Programmierung

Der Übergang vom Objektmodell zum objektorientierten Programm ist recht anspruchsvoll und wird, wie unsere Untersuchungen zeigen, von den Generatoren der marktgängigen Werkzeuge keineswegs optimal unterstützt. Um den – nach wie vor größtenteils vom Entwickler zu leistenden – Transformationsprozeß transparenter zu gestalten, ist eine *wechselseitige* Sicht auf den Übergang Modellierung/Programmierung sinnvoll:

■ Aus *Sicht der Programmierung* beschreiben Richtlinien die Abbildung von Modellelementen in programmiersprachliche Konstrukte. Diese geben etwa an, wie eine modellierte Mehrfachvererbung in Smalltalk auf einfache Vererbung und Delegation abgebildet wird, oder wie Zusicherungen durch Hüllenmethoden, welche die eigentliche Funktionalität durch Vor- und

Nachbedingungen kapseln, implementiert werden.

■ Aus *Sicht der Modellierung* werden Designempfehlungen gegeben, damit Besonderheiten der Zielsprache im physischen Entwurf berücksichtigt werden können. So sollten beispielsweise im Entwurf geschachtelte Klassen vermieden werden, da diese in Smalltalk nicht abgebildet werden können und in C++ dazu führen, daß die Schnittstelle der einbettenden Klasse unübersichtlich wird. Zusammengefaßt dienen diese Richtlinien und Empfehlungen als Nachschlagewerk für den Entwickler und den Modellierungsexperten. Weiterhin können sie dabei helfen, die Generierungsfähigkeiten von Werkzeugen realistischer einzuschätzen.

## Architektur

Das Kernkonzept *Modellierung* definiert insbesondere die sprachlichen Ausdrucksmittel für die Analyse und das Design einer Anwendung. Betrachten wir als nächstes die Frage, wie – auf der Grundlage dieser Ausdrucksmittel – eine geeignete Systemarchitektur zu entwerfen ist. Dabei gilt: „Good architectural design has always been a major factor in determining the success of a software system“ ([Sha96], S. 15). Der Leitfaden unterscheidet hinsichtlich der Architektur drei Hauptaspekte: Schichtenbildung, Verteilungsmodell und Persistenz (vgl. Abb. 3).

### Schichten

Die Schichtenbildung ist ein wesentliches Architekturmuster zur Reduzierung der Komplexität einer Anwendung (vgl. [Bus96]). Die Anordnung von Anwendungssystemen in Schichten geht dabei von einer hierarchischen Strukturierung aus. Die Funktionalität einer höherliegenden Schicht ist abhängig von der Funktionalität einer oder mehrerer tiefer gelegener Schichten (*geschlossene* oder *offene* Schichtenarchitektur). Die Schichtenbildung erlaubt es, Anwendungsteile weitgehend voneinander unabhängig zu halten, wodurch Änderbarkeit, Wartbarkeit und Austauschbarkeit verbessert werden.

Der Leitfaden zeigt auf, wie Schichten, Komponenten und deren Beziehungen in der UML dargestellt werden, legt in einzelnen Schritten die Vorgehensweise zur Entwicklung eines Schichtenmodells dar und definiert eine offene (Referenz-)Schich-

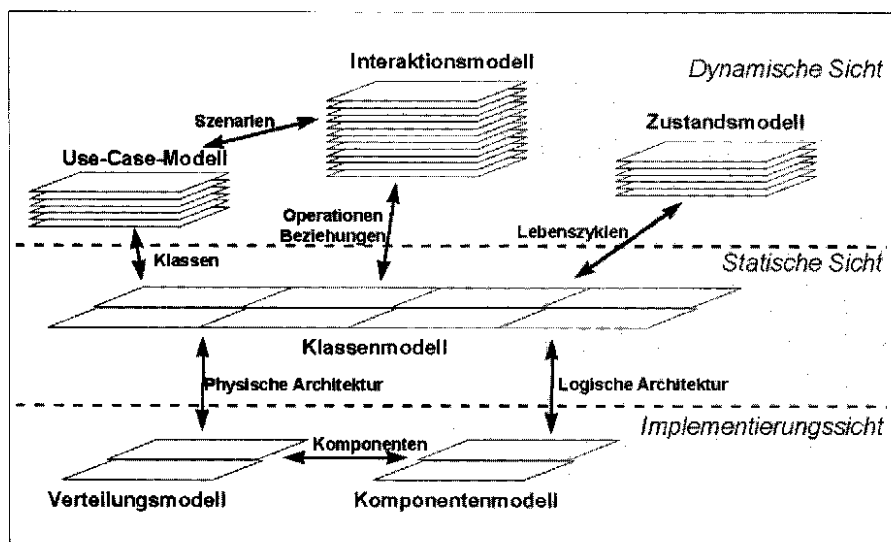


Abb. 2: Modelle und Modellzusammenhänge des Modellierungskerns

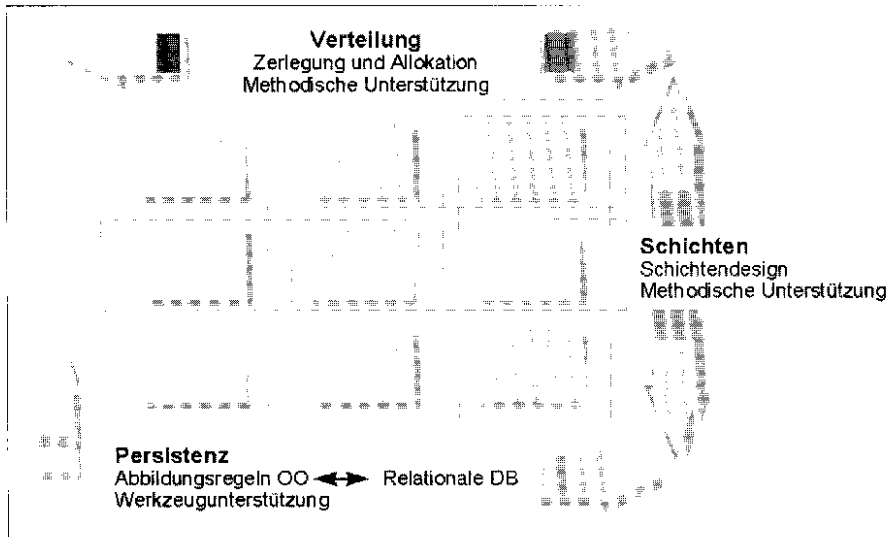


Abb. 3: Themenbereiche der Architektur

tenarchitektur mit den folgenden sechs Schichten:

- Benutzungsschnittstelle,
- Dialogsteuerung,
- Geschäftsprozesssteuerung,
- Anwendungsbereichsobjekte sowie
- logischer und
- physischer Datenzugriff.

Diese Referenzarchitektur ist mit der klassischen Drei-Schichten-Architektur und mit dem in der Smalltalk-Entwicklung häufig verwendeten *Model-View-Controller*-Paradigma verträglich.

Als Erweiterung der bisher von der UML zur Verfügung gestellten Konzepte (insbesondere des *Package*-Konzepts und der *Dependency*-Beziehung) werden weitere semantische Beziehungen – wie etwa *Client-Supplier* oder *Peer-to-Peer* – als benutzerdefinierte Stereotypen zur Schichtenspezifikation eingeführt.

### Verteilung

Zu den anspruchsvollsten Entscheidungen bei der Entwicklung von Client/Server-Anwendungen gehört das Design der Verteilung, von dem das Nutzungsverhalten, die Sicherheit und der Betrieb der Anwendung stark abhängen. Im Leitfaden werden verschiedene logische Verteilungsmodelle dargestellt und anhand von Bewertungskriterien – wie z. B. Netzlast, Implementierungs- und Administrationsaufwand – so bewertet, daß der Leser Hinweise zu konkreten Einsatzfeldern erhält. Der Übergang vom Schichtenmodell zum Verteilungsmodell wird erläutert, und es werden Hilfestellungen für die Zerlegung und Allokation von Komponenten gegeben. Insbesondere wird erläutert, wie die

se Verteilungsaspekte in dem auf der UML basierenden Modellierungskern dargestellt werden können. Der physikalischen Verteilung liegt die in der Sparkassenorganisation übliche dreistufige Hardwarearchitektur (Mainserver, Server-Stationen, Client-Stationen) zugrunde. Als verbindende Middleware-Technologien für die verteilten Komponenten werden

- MOM (Message Oriented Middleware),
- DCE (Distributed Computing Environment),
- CICS (Customer Information Control System),
- CORBA (Common Object Request Broker Architecture) und
- DCOM (Distributed Component Object Model)

aus dem Blickwinkel der Anwendungsentwicklung betrachtet. (Eine gute Beschreibung dieser Technologien gibt [Orf96]; weitere Hinweise zur Nutzung der Technologien finden sich in [Heu97]).

### Persistenz

Die große Verbreitung von relationalen Datenbanksystemen sowie die beträchtliche Anzahl der Datenbestände, die dort verwaltet werden, erfordert, daß die Anbindung von relationalen Datenbanken an objektorientierte Anwendungen systematisch angegangen wird. Zum einen müssen Hinweise gegeben werden, wie bestehende relationale Datenbanken in objektorientierte Anwendungen eingebunden werden können, zum anderen helfen Abbildungsregeln, persistente Teile des Objektmodells in relationale Datenbanken zu übertragen (vgl. etwa [Rum91], [Hah95], [Tka96]). Da häufig verschiedene Varianten der Abbil-

dung der Modellelemente des Klassenstrukturmodells bzw. für Objektidentitäten und Transaktionen existieren, werden im Leitfaden die jeweiligen Vor- und Nachteile herausgestellt und so Hinweise gegeben, welche Abbildung in einem konkreten Szenario zu wählen ist. Die Abbildungsregeln dienen dem Entwickler einerseits als Nachschlagewerk zur dauerhaften Speicherung von Objekten in relationalen Datenbanken. Andererseits stellen sie auch eine Meßlatte zur Beurteilung der Leistungsfähigkeit von Persistenzwerkzeugen dar, die den Abbildungsvorgang automatisieren.

## Programmierung

Programmierrichtlinien helfen, die Lesbarkeit, Verständlichkeit und Qualität des Quellcodes zu erhöhen. Sie erleichtern die Wartung, erhöhen die Wiederverwendbarkeit und verbessern die Austauschbarkeit von Anwendungen.

Um den Entwickler konkret bei der Implementierung zu unterstützen und die Akzeptanz des Leitfadens zu erhöhen, sind neben Empfehlungen zu Namenskonventionen oder zur Code-Formatierung und -Dokumentation auch Implementierungsmuster (Idiome) bzw. „Best Practices“ beschrieben (vgl. [Cop92], [Bec97]). Ein einfaches, bewährtes Smalltalk-Lösungsmuster ist etwa die Attributinitialisierung beim Lesezugriff („Lazy Initializing“) anstelle einer Konstruktoreninitialisierung, wenn diese nicht performant ist oder die Attributbelegungen beim Anlegen des Objekts unbekannt sind.

Ein wesentliches Ziel des Leitfadens besteht darin, die Einheitlichkeit und Austauschbarkeit von Anwendungen innerhalb der Sparkassenorganisation zu fördern. Aus diesem Grund beinhaltet der Aspekt Programmierung neben den Programmierrichtlinien auch Empfehlungen zur einheitlichen Behandlung von Fehlern bzw. Ausnahmen sowie zum Design der Benutzungsschnittstelle. Letztere berücksichtigen die Anforderungen der Ergonomie und der „Corporate Identity“, wie sie in den Gestaltungsrichtlinien für Benutzungsoberflächen (vgl. [SIZ96]) festgehalten sind. Als Programmierungsumgebungen werden zur Zeit VisualAge für Smalltalk und C++ berücksichtigt, wobei der Leitfaden größtenteils produktunabhängig ist. Entsprechende Erweiterungen für Java werden gegenwärtig erarbeitet.

## Werkzeuge

Eine objektorientierte Anwendungsentwicklung ist ohne umfassende Werkzeugunterstützung nicht denkbar. Auch im objektorientierten Umfeld gilt, daß es keine integrierte Softwareentwicklungsumgebung aus einer Hand gibt, die sämtliche Aktivitäten der Softwareerstellung und -wartung sowie das Management der Softwareprozesse zufriedenstellend unterstützt (vgl. [Lat97]).

In Anlehnung an das NIST/ECMA-Referenzmodell (vgl. [ECMA94]) wird im Leitfaden eine Softwareentwicklungsarchitektur aufgezeigt, die als Orientierungsrahmen für eine Werkzeugauswahl und deren Integration dient. Dieses Referenzmodell schlägt ein standardisiertes Framework zur Integration von Entwicklungsumgebungen dar. Dessen Komponenten bestehen – bezogen auf den Softwarelebenszyklus – aus horizontalen und vertikalen Werkzeugen sowie aus einer Entwicklungsdatenbank. Vertikale Werkzeuge, etwa für das Projektmanagement, die Konfiguration, die Qualitätssicherung oder die Wiederverwendung, umspannen den gesamten Lebenszyklus. Horizontale Werkzeuge, beispielsweise Entwurfs-, Wartungs- und GUI-Werkzeuge, decken statt dessen nur einzelne Aspekte ab.

Neben allgemeinen Kriterien für die Auswahl und Integration von Werkzeugen wurden detaillierte Anforderungen für die Beurteilung von Entwurfs- und Persistenzwerkzeugen sowie von Programmierumgebungen aufgelistet, da die Auswahl von Werkzeugen dieser drei Kategorien besonders hohe Priorität besitzt. Während der Entwicklung des Leitfadens wurden für Werkzeuge der unterschiedlichen Kategorien Marktübersichten erstellt, indem ausgesuchte Tools anhand dieser Kriterien beurteilt werden. Als Entwurfswerkzeuge wurden beispielsweise „Rational Rose“, „Paradigm Plus Enterprise“ und „Select Enterprise“ detailliert betrachtet. Da sich der OO-Werkzeugmarkt aber sehr dynamisch entwickelt, haben die Ergebnisse einer solchen Beurteilung natürlich nur temporäre Aussagekraft.

## Vorgehensmodell

Objektorientierte Vorgehensmodelle unterscheiden sich von klassischen Vorgehensmodellen für die strukturierte Anwendungsentwicklung insbesondere aufgrund der Durchgängigkeit der Entwicklungsergebnisse und der Umkehrbarkeit des Entwurfsprozesses – *Reversibility* und *Seamlessness* werden insbesondere von Waldén und Nerson hervorgehoben (vgl. [Wal95]). Phasenabschlüsse als

Revisionspunkte werden durch inkrementelle Erweiterungen eines bereits per Review geprüften Softwareprodukts abgelöst. Ein Vorgehensmodell sollte den folgenden Merkmalen der objektorientierten Anwendungsentwicklung Rechnung tragen:

- **Iterativ:** Die Entwicklung verläuft in zahlreichen Zyklen, d. h. eine Folge von Aktivitäten wird wiederholt durchlaufen.
- **Inkrementell:** Durch jeden Zyklus wird das zuvor erstellte Produkt bzw. das Gesamtsystem erweitert.
- **Ereignisorientiert:** Auf neue Anforderungen muß auch während der Projeklaufzeit reagiert werden können.
- **Architekturzentriert:** Die Entwicklung folgt einer Softwarearchitektur, die in den einzelnen Iterationen verfeinert wird.
- **Anwendungsgetrieben:** Die Entwicklung basiert auf – beispielsweise durch Anwendungsfälle – definierten Anforderungen.

Um unterschiedliche Projekttypen in der objektorientierten Entwicklung – wie z. B. geschäftsprozeß- und datenstrukturgetriebene Entwicklung oder Prototyping – unterstützen zu können, wurde das objektorientierte Vorgehensmodell als generisches Referenzmodell (vgl. Abb. 4) ausgelegt.

Die Aktivitäten der Anforderungsanalyse, Modellierung (mit Analyse und Design), Umsetzung und Einführung werden während der gesamten Projektlaufzeit immer wieder angestoßen. Als Metapher zur Darstellung des Vorgehensmodells dient eine Wendeltreppe mit acht Stufen in der Draufsicht. Das Vorgehensmodell besteht aus acht Kernaktivitäten, die jeweils die Wurzel einer Aktivitätenhierarchie bilden. Es umfaßt insgesamt ca. 200 Aktivitäten. Diese werden durch Vor- und Nachbedingungen verknüpft. Solange die Vorbedingungen einer Aktivität nicht erfüllt sind, kann diese nicht gestartet werden, und solange die Nachbedingungen nicht erfüllt sind, darf sie nicht beendet werden. Durch die Ausführung einer Aktivität werden bestimmte Dokumente, z. B. Modelle bzw. Diagramme, bearbeitet.

Das generische Vorgehensmodell dient als Rahmenwerk, um projekttypspezifische Vorgehensmodelle auf der Grundlage von Prozeßmustern (vgl. dazu [Cop95]) – wie etwa für die geschäftsprozeßgetriebene Entwicklung oder das Prototyping – abzuleiten. Durch die schärfere Formulierung von Randbedingungen und die Streichung oder Ergänzung von Aktivitäten können weitere

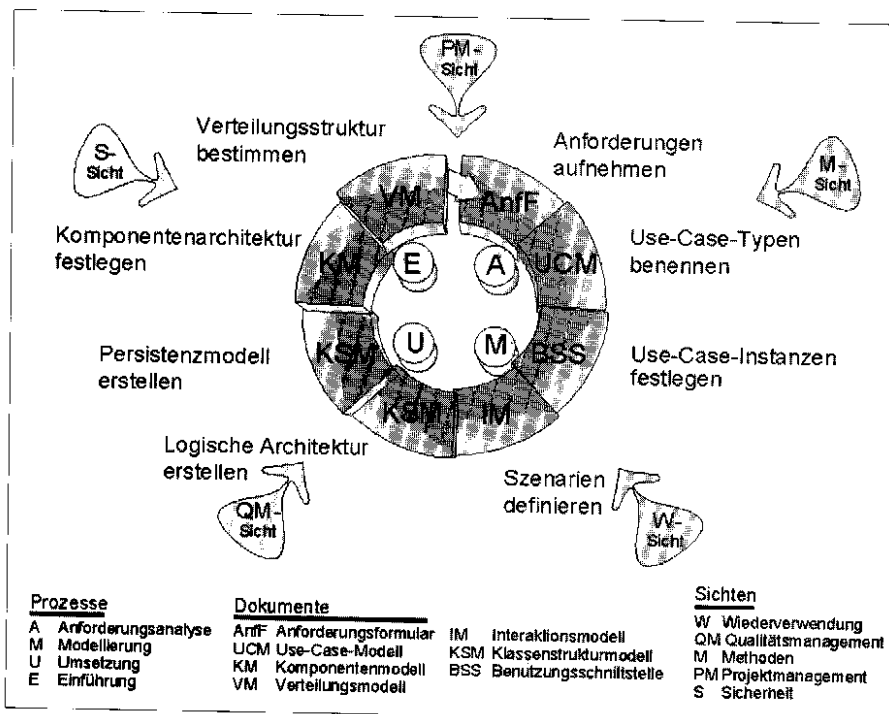


Abb. 4: Vorgehensmodell

projektspezifische Anpassungen (sogenanntes *Tailoring*) vorgenommen werden.

Neben der Methodensicht, welche die Aktivitäten der objektorientierten Entwicklung im engeren Sinne repräsentiert, existieren weitere Sichten für die Aspekte Sicherheit, Qualitätsmanagement, Wiederverwendung und Projektmanagement. Allen Sichten sind entsprechende Rollen zugeordnet, die in der Projektarbeit durch Mitarbeiter, die über definierte Kenntnisse und Fähigkeiten verfügen, zu besetzen sind.

## Wiederverwendung

Die Objektorientierung bietet die Möglichkeit der breitflächigen Wiederverwendung von Modellen, Mustern, Komponenten, Klassenbibliotheken oder Frameworks. Um mögliche Nutzpotentiale – wie die Senkung der Wartungsaufwände, die Steigerung der Entwicklungsproduktivität und eine Qualitätsverbesserung durch den Einsatz ausgereifter Komponenten – aber auch tatsächlich auszus schöpfen, bedarf es einer systematischen Herangehensweise.

Der Leitfaden enthält Konzepte und konkrete Handlungsanweisungen zur Nutzung und Erstellung von wiederverwendbaren Elementen und gibt Hinweise auf Organisationsstrukturen, die eine systematische Wiederverwendung unterstützen (siehe Abb. 5). Einen guten Überblick über die Thematik geben [Kar95] und [Zen95]. In [Jac97] werden mit *manage*, *create*, *reuse* und *support* vier zentrale Wiederverwendungsprozesse unterschieden. Dies entspricht in etwa der Dreiteilung *Nutzung*, *Erstellung* und *Organisation* des Leitfadens, wobei der Prozeß *Organisation* weiter in *manage* und *support* gegliedert werden kann.

## Sicherheit

Das nötige Vertrauen in die Sicherheit eines Anwendungssystems kann nur dann entstehen, wenn während des Softwareentwicklungsprozesses alle erforderlichen Sicherheitsmaßnahmen berücksichtigt wurden. Dies bedeutet, daß

- die Umsetzung der inhaltlichen Sicherheitsanforderungen (Vertrauenswürdigkeit, Integrität und Verfügbarkeit) sichergestellt ist,
- die Integrität aller Zwischen- und Endergebnisse gewährleistet ist und
- diese bei einer späteren Abnahme und Überprüfung nachvollziehbar bleiben.

Im Bereich *Homebanking* müssen Sicherheitsanforderungen etwa verhindern, daß sich ein Angreifer für den Anwender als Kreditinstitut ausgibt. Daher wird gefordert, daß sich die Kommunikationspartner gegenseitig identifizieren und authentifizieren. Zur sicheren Identifikation sind symmetrische oder asymmetrische Verschlüsselungsverfahren mit genügend großer Schlüssellänge einzusetzen. Weitere Gefahren bestehen im Abhören sensibler Informationen, wie z. B. Paßwörtern, PINs, TANs und Kreditkartennummern, oder im Abfangen und späteren Wiedereinspielen von Nachrichten. Um hier Manipulationen zu verhindern bzw. aufzudecken, muß sichergestellt sein, daß Nachrichten verschlüsselt und die Daten mittels digitaler Unterschriften signiert werden.

Solche inhaltlichen Sicherheitsanforderungen sind während des gesamten Entwicklungsprozesses zu berücksichtigen und projektbegleitend zu dokumentieren, um deren Umsetzung transparent zu machen. Dabei ergeben sich im Vergleich zur strukturierten Anwendungsentwicklung durch die iterative und inkrementelle Vorgehensweise, die Verteilung von Komponenten, die Wiederverwendung und schließlich auch durch die inhärenten Eigenschaften objektorientierter Programmiersprachen (Vererbung, Polymorphie) eine Reihe zusätzlicher sicherheitsspezifischer Fragen, die im Entwicklungsprozeß beachtet werden müssen.

## Qualitätsmanagement

Das Qualitätsmanagement stellt eine kontinuierliche Aufgabe in der Anwendungsentwicklung dar. Dabei unterscheiden sich die angestrebten Qualitätsmerkmale und die grundsätzliche Herangehensweise nicht wesentlich vom strukturierten Umfeld. Die Umsetzung der einzelnen *konstruktiven*, *analytischen* und *organisatorischen* Maßnahmen erfordert jedoch teilweise gänzlich andere Verfahren und Techniken.

Zur Prüfung von UML-Modellen wurden spezielle Checklisten definiert. Weitere Maßnahmen sind die Code-Inspektion oder der Einsatz von objektorientierten Softwaremetriken (vgl. dazu etwa [Lor94]). Das Testen – als immer noch wichtigste analytische Qualitätssicherungsmaßnahme – nimmt aufgrund des dafür einzuplanenden Aufwands eine Sonderstellung ein. Das Testkapitel des Leitfadens enthält daher zahlreiche Durchführungshinweise zu speziellen Testverfahren (vgl. [McG94], [Amb96]), wie

- Use-Case-Test,
- Zustandstest,
- Klassentest und
- Regressionstest für Vererbung etc.

Diese Tests erfolgen auf unterschiedlichen Ebenen, wie

- Modultest,
- Integrationstest,
- Systemtest und
- Abnahmetest.

## Projektmanagement

Ein wichtiger Erfolgsfaktor für die Einführung der Objektorientierung in der Anwendungsentwicklung ist das Projektmanagement. Das SIZ hat 1996 ein allgemeines Projektmanagementkonzept für die Sparkassenorganisation entwickelt (vgl. [Noa96]), das auf IT- und Nicht-IT-Projekte anwendbar ist und sehr leicht für die konventionelle Anwendungsentwicklung adaptiert werden kann. Obwohl die Rahmenorganisation eines Projekts weitestgehend unabhängig vom Entwicklungsparadigma ist, besteht für die objektorientierte Entwicklung weiterer Anpassungsbedarf in den Rollen der Projektmitarbeiter, den Qualifizierungskonzepten und in der Projektablauforganisation. Planung, Steuerung, Review und Abschluß dürfen nicht mehr phasenorientiert erfolgen, sondern müssen an die iterative und inkrementelle Vorgehensweise angepaßt werden (vgl. [Hes94]).

## Weitere Aspekte

Im Leitfaden werden die Kernkonzepte der objektorientierten Anwendungsentwicklung *Modellierung*, *Architektur*, *Programmierung*, *Werkzeuge* und *Vorgehensmodell* behandelt, ohne daß Aussagen zu unternehmensspezifischen Migrationsstrategien gemacht werden (gute Hinweise dazu finden sich in [Gol95]). Der Leitfaden ist neutral formuliert gegenüber orthogonalen Ansätzen zur Identifikation und Klassifizierung von Objekten in der Anforderungsanalyse und Modellierung. Solche Ansätze sind beispielsweise die in [Kil94] beschriebene Werkzeug-Material-Metapher, der auf der natürlichen Sprache basierende Ansatz von Cockborn (vgl. [Coc92]) oder der in [Sch97] beschriebene Ansatz zur Rekonstruktion der Fachbegriffe eines Anwendungsbereichs. Durch die Einführung entsprechender Stereotypen wie „Werkzeug“ oder „Material“ für Klassen ist beispielsweise eine Abbildung der Werk-

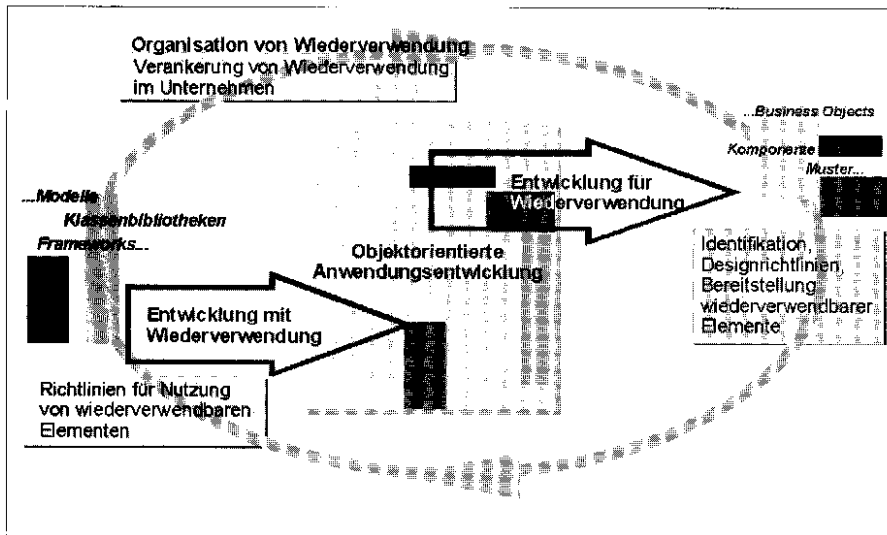


Abb. 5: Wiederverwendung

zeug-Material-Metapher auf den Modellierungskern in UML-Notation möglich.

Als Einstiegshilfe in die objektorientierte Entwicklung auf Basis des Leitfadens wurde ein Qualifizierungskonzept erarbeitet. Das Konzept geht davon aus, daß die Know-how-Profile der im objektorientierten Vorgehensmodell definierten Rollen durch marktgängige Schulungsmaßnahmen erreicht werden können. Bei der Definition der einzelnen Maßnahmen wurde insbesondere auch die Weiterbildung von Mitarbeitern berücksichtigt, die bisher im strukturierten Umfeld tätig waren.

## Fazit und Ausblick

Die Einführung eines Leitfadens für die objektorientierte Entwicklung verspricht durch die Standardisierung der Vorgehensweise in puncto Qualität, Produktivität, Termin- und Budgettreue viele Vorteile. Die für Methoden und Verfahren zuständigen Abteilungen leiden jedoch häufig unter dem Ruf mangelnder Akzeptanz bei ihren Kunden im Management und in der Anwendungsentwicklung. Die Ursache liegt häufig darin, daß die entsprechenden Standards und Richtlinien als „Schrankware“ verschwinden. Um solchen Schwierigkeiten zu begegnen, wurde der Leitfaden für die strukturierte Anwendungsentwicklung in der Sparkassenorganisation mit Hilfe eines Prozeßmodellierungswerkzeugs aufbereitet (vgl. [Noa97]), wodurch ein entscheidender Beitrag zur Akzeptanzverbesserung erzielt wurde. Aufgrund dieser positiven Erfahrungen ist geplant, den hier

beschriebenen Leitfaden für die objektorientierte Anwendungsentwicklung in einem nächsten Schritt für die Verbreitung im Intranet aufzubereiten.

## Danksagung

Die Autoren danken allen Mitarbeitern und Review-Partnern des SIZ-Projekts „AE-Modell OO-Ergänzung“, die maßgeblich an der Erarbeitung des Leitfadens mitgewirkt haben.

## Literatur

[Amb96] S. Ambler, Testing Objects, in: Software Development, 8/1996, S. 55-62  
 [Bec97] K. Beck, Smalltalk Best Practices, Prentice Hall, 1997  
 [Boo93] G. Booch, Object-Oriented Analysis and Design, With Applications, Benjamin Cummings, 1993  
 [Bun95] W. Bungert, H. Heß, Objektorientierte Geschäftsprozeßmodellierung, in: Informationsmanagement 1/95, S. 52-63  
 [Bur97] R. Burkhardt, UML – Unified Modeling Language, Objektorientierte Modellierung für die Praxis, Addison-Wesley, 1997  
 [Bus96] F. Buschmann, R. Meunier, H. Rohnert, M. Stal, P. Sommerlad, A System of Patterns, Pattern-Oriented Software Architecture, Wiley, 1996  
 [Coc92] A.A.R. Cockburn, Using Natural Language as a Metaphoric Base for Object-Oriented Modeling and Programming,

Technical Report TR-36.0002, IBM, 1992  
 [Cop92] J. Coplien, Advanced C++ Programming, Styles and Idioms, Addison-Wesley, 1992

[Cop95] J. Coplien, Organizational and Process Patterns, Bell Laboratories, 1995 (siehe <http://www.bell-labs.com/people/cope/Patterns/Process/>)

[ECMA94] ECMA (Hrsg.), Mapping of PCTE to the ECMA/NIST Frameworks Reference Model, Technical Report TR/66, ECMA, 1994

[Gol95] A. Goldberg, K.S. Rubin, Succeeding with Objects, Decision Frameworks for Project Management, Addison-Wesley, 1995

[Hah95] W. Iahn, F. Toeniessen, A. Wittkowski, Eine objektorientierte Zugriffsschicht zu relationalen Datenbanken, in: Informatik Spektrum 18 (1995), S. 143-151

[Hes94] W. Hesse, F. Wcltz, Projektmanagement für evolutionäre Software-Entwicklung, in: Information Management 9 (1994) 3, S. 20-32

[Heu97] H. Heuer-Hasenpatt, B. Hollunder, H.-B. Kittlaus, N. Schumacher, Bausteinorientierte Anwendungsentwicklung: Voraussetzungen, Anforderungen und Auswirkungen, in: OBJEKTspektrum 3/97, S. 40-51

[Jac92] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard, Object-Oriented Software Engineering, A Use Case Driven Approach, Addison-Wesley, 1992

[Jac94] I. Jacobson, M. Ericsson, A. Jacobson, The Object Advantage, Business Process Reengineering with Object Technology, Addison-Wesley, 1994

[Jac97] I. Jacobson, M. Griss, P. Jonsson, Software Reuse, Architecture, Process and Organization for Business Success, Addison-Wesley, 1997

[Kar95] E.-A. Karlsson, Software Reuse, A Holistic Approach, Wiley, 1995

[Kar97] W. Karbach, J. Noack, H.-B. Kittlaus, Leveraging a Large Banking Organization to Object Technology, Proc. 19th. Int. Conf. on Software Engineering, Boston, ACM Sigsoft 1997, S. 554-555

[Kil94] K. Kilberth, G. Gryczan, H. Züllichoven, Objektorientierte Anwendungsentwicklung, Konzepte, Strategien, Erfahrungen, Vieweg, 1994

[Kue96] P. Kueng, P. Bichler, M. Schrefl, Geschäftsprozeßmodellierung – ein zielbasierter Ansatz, in: Information Management 2/96, S. 40-50

[Lat97] T. Latza, Programmierentwicklungsumgebungen für Smalltalk – Wunsch

- und Wirklichkeit, in: OBJEKTspektrum 4/97, S. 65-71
- [Lor94] M. Lorenz, J. Kidd, Object-Oriented Software Metrics, Prentice Hall, 1994
- [McG94] J.D. McGregor, T.D. Korson, Object-Oriented Software Testing and Development, in: CACM 9/94, S. 59-77
- [Noa96] J. Noack, T. Balsler, Umfassendes PM-Konzept für die Praxis, in: Betriebswirtschaftliche Blätter 11/96, S. 507-510
- [Noa97] J. Noack, AE-Modell: Prozeßorientiertes Rahmenwerk für die Anwendungsentwicklung in der Sparkassenorganisation, (erscheint in: Information Management 4/97)
- [Orf96] R. Orfali, D. Harkey, J. Edwards, The Essential Distributed Objects Survival Guide, Wiley, 1996
- [Rum91] J. Rumbaugh, M. Blaha, W. Premerlani et al., Object-Oriented Modeling and Design, Prentice Hall, 1991
- [Rum95] J. Rumbaugh, OMT Method Summary, Rational Software Corporation, 1995
- [Sch96] A. Scheer, Wirtschaftsinformatik – Referenzmodelle für industrielle Geschäftsprozesse, 6. Auflage, Springer Verlag, 1996
- [Sch97] B. Schienmann, Objektorientierter Fachentwurf, Teubner, 1997
- [Sha96] M. Shaw, D. Garlan, Software Architecture, Perspectives on an Emerging Discipline, Prentice Hall, 1996
- [Sie96] S. Siegel, Object-Oriented Software Testing, Wiley, 1996
- [SIZ96] SIZ GmbH (Hrsg.), Gestaltungsrichtlinien für grafische Oberflächen (Style-Guide), Deutscher Sparkassenverlag, 1996
- [Tka96] D. Tkach, F. Walter, A. So, Visual Modeling Technique, Object Technology Using Visual Programming, Addison-Wesley, 1996
- [UML97] Unified Modeling Language (UML), Rational Software Corporation, 1997 (siehe <http://www.rational.com/uml/1.1>)
- [Wal95] K. Waldén, J.-M. Nerson, Seamlessness Object-Oriented Software Architecture, Analysis and Design of Reliable Systems, Prentice Hall, 1995
- [Zen95] A. Zandler, Konzepte, Erfahrungen und Werkzeuge zur Wiederverwendung, Tectum, 1995

**OBJEKTspektrum** ist eine Fachpublikation des Verlags:



SIGS Conferences GmbH  
Hauptstr. 293 - 297 · D-51465 Bergisch Gladbach  
Tel.: 02202/9372-0 · Fax: 02202/9372-2  
E-mail: 100634.2070@compuserve.com  
URL: //www.sigs.com/publications/obsp