Objektorientiertes Prototyping
- Konzepte, Werkzeuge und Erfahrungen -

Dirk Bäumer
RWG
Räpplenstr. 17
D-70191 Stuttgart
dirk_baeumer@rwg.e-mail.com

Walter R. Bischofberger
Schweizerische Bankgesellschaft
Bahnhofstr. 45
CH-8021 Zurich
bischi@ubilab.ubs.ch

Horst Lichter
ABB Corporate Research
Postfach 101332
D-69003 Heidelberg
lichter@decrc.abb.com

Heinz Züllighoven
Universität Hamburg
Vogt-Köllnstr. 30
D-22527 Hamburg
zuellighoven@informatik.uni-hamburg.de

Zusammenfassung

Die Entwicklung hoch interaktiver graphischer Software gewinnt für Arbeitspatzsysteme zunehmend an Bedeutung. Dazu werden häufig objektorientierte Sprachen und Systeme eingesetzt. Ebenso oft ist Prototyping ein Bestandteil der Entwicklungsstrategie. Dieser Beitrag wertet industrielle Erfahrungen und Tendenzen aus und gibt Empfehlungen für das objektorientierte Prototyping. Fazit:
Wird Prototyping als durchgängiges Konzept und nicht als Aufsatz auf ein herkömmliches Phasenmodell eingesetzt, so sind die erzielten Ergebnisse durchweg positiv. Dies gilt besonders, wenn Prototyping zusammen mit Client-Server Computing oder Objektorientierung eingesetzt wird. Objektorientierte Entwicklungsumgebungen und objektorientiert konstruierte Application Frameworks sind besonders dann für das Prototyping geeignet, wenn ausgehend von den Prototypen evolutionär das Anwendungssystem mit einer verständlichen und änderbaren Architektur entwickelt werden soll. Dies unterscheidet diese Entwicklungsplattformen deutlich von 4. Generationssystemen oder von klassischen Interface-Buildern.[1]
Schlüsselwörter: Prototyping, Objektorientierung, graphische Benutzungsoberflächen, Prototyping-Werkzeuge, industrielle Erfahrungen, Entwicklungsprozeß

1. Einleitung und Motivation

Betrachten wir die Tendenzen, die sich heute in der Entwicklung von Anwendungssoftware abzeichnen, so fällt auf, daß hier neben Client-Server Computing vor allem objektorientierte Entwicklungssysteme und Prototyping zu nennen sind. Dieser Zusammenhang wird noch deutlicher, wenn der Anteil der hoch interaktiven graphischen Anwendungen betrachtet wird. Warum ist dies so?

Zwei Gründe sind vor allem zu nennen: Interaktive graphische Anwendungen werden heute meist als sog. „reaktive Systeme“ gebaut. Dies bedeutet, daß alle Systemaktivitäten initial durch Benutzungsereignisse wie Mausklick oder Tastenanschläge ausgelöst werden. Alle gängigen Fenstersysteme sind auf solche reaktive Benutzung ausgelegt und die dahinter stehende Softwarearchitektur legt einen objektorientierten Entwurf der einzelnen Komponenten nahe. Das führt aber zu der Frage, wie denn die potentiellen Benutzeraktionen aussehen sollen. Eng damit verbunden ist die Beobachtung, daß die Akzeptanz solcher interaktiven graphischen Anwendungen stark von ihrer Benutzungsoberfläche und Handhabbarkeit abhängt. Prototyping ist in beiden Fällen ein ausgezeichnetes Mittel, um Ideen über die zukünftige Oberfläche und die Benutzung zu entwickeln. Durch Prototyping können diese Ideen vergegenständlicht werden, um sie mit den Anwendern zu diskutieren und so die Qualität der Systeme zu verbessern.
Sollen diese ersten Prototypen allerdings evolutionär zu einem großen Anwendungssystem ausgebaut werden, so ist die Wahl einer stabilen Softwarearchitektur entscheidend. Objektorientierte Techniken, insbesondere objektorientierte Frameworks haben sich dabei bewährt.
Dieser Beitrag wertet industrielle Erfahrungen aus, die auf einer Untersuchung von neun industriellen Projekten basieren, bei denen der Einsatz von Prototyping im Mittelpunkt stand. Ein weiterer Erfahrungshintergrund für diesen Beitrag bilden unsere eigenen Arbeiten im Bereich objektorientierter Frameworks und industrieller Projektbegleitung. Die Kürze des Beitrags verbietet, alle im Hauptteil formulieren Empfehlungen und Tendenzen beim objektorientierten Prototyping in Form ausgearbeiteter Folgerungen aus vorgelegten Fakten darzustellen.

2. Grundlagen

Im folgenden Abschnitt führen wir die von uns benutzte Terminologie ein. Weiterhin diskutieren wir grundlegende Framework-Konzepte.

---

[1] publiziert in *Procs. Softwaretechnik '96*, Koblenz, 12.-13. September 1996

2.1 Prototyping

In der Diskussion über Prototyping hat sich die Einteilung von Floyd [5] in die drei „E“ – explorativ, experimentell, evolutionär – durchgesetzt. Interpretationen dieser Begriffe ergeben sich in zwei Richtungen: Der Prozeß und die dabei angestrebten Ziele werden betrachtet (vgl. [4]) oder der Prototyp wird als Produkt in den Mittelpunkt gestellt (vgl. [3]). Nachfolgend ordnen wir den drei „E“ der Prozeßsicht die dabei verwendeten Prototyparten zu.

- *Exploratives Prototyping* soll die Problemstellung klären. Diskutiert werden veränderte Arbeitsinhalte und die Möglichkeiten der DV-Unterstützung.

  Prototyparten: Demonstrationsprototypen, funktionale Prototypen.

- *Experimentelles Prototyping* klärt die technische Umsetzung der Anforderungen an ein System.

  Prototyparten: funktionale Prototypen, Labormuster.

- *Evolutionäres Prototyping* ist Teil eines kontinuierlichen Verfahrens, in dem ein Anwendungssystem innerhalb eines Entwicklungsprozesses an die sich ändernden Anforderungen angepaßt wird.

  Prototyparten: alle. Große Bedeutung haben Pilotsysteme, da selten Zielsysteme im traditionellen Sinne entwickelt werden.

Während sich die Prozeßsicht des Prototyping auf die untersuchten Fragestellungen bezieht, werden die Prototyparten entsprechend ihrer Gestaltung im Sinne eines Produktes klassifiziert:

- *Demonstrationsprototypen* zeigen die prinzipiellen Einsatzmöglichkeiten, meist nur die möglichen Handhabungsformen des künftigen Systems. Sie sind somit stark auf die zukünftige Benutzungsoberfläche ausgerichtet.
- *Funktionale Prototypen* modellieren sowohl Ausschnitte der Benutzungsoberfläche als auch Teile der fachlichen Funktionalität.
- *Labormuster* modellieren einen technischen Aspekt des späteren Anwendungssystem. Dieser Aspekt kann sich je nach Fragestellung auf die Architektur oder auf die Funktionalität beziehen.
- *Pilotsysteme* sind Prototypen von solcher Ausbaustufe und „Reife“, daß sie im Anwendungsbereich und nicht nur unter Laborbedingungen eingesetzt werden.

In diesem Beitrag sprechen wir oft von Oberflächenprototypen. Damit ist keine weitere Prototypart gemeint, sondern wir bezeichnen damit solche Prototypen, bei denen die Modellierung der Benutzungsoberfläche im Vordergrund steht.

2.2 Frameworks und Prototyping

Objektorientierte Sprachen stellen dank Vererbung und Polymorphismus mächtige Grundmechanismen zur Verfügung, um adaptierbare und erweiterbare Softwaresysteme zu implementieren. Basierend auf diesen Grundmechanismen lassen sich Bibliotheken mit wiederverwendbaren Bausteinen aufbauen, die sich ausgezeichnet für das Prototyping eignen.
Ein Framework besteht aus einer Menge von Klassen, die eine generische Lösung für eine Reihe verwandter Probleme implementieren. Es legt dabei die Rollen der einzelnen Klassen bzw. Objekte und ihr Zusammenspiel untereinander fest. Das Framework definiert auch jene Stellen, an denen die Funktionalität erweitert und angepaßt werden kann.
Die atomare Einheit für die Wiederverwendung, die durch ein Framework definiert wird, ist somit nicht eine einzelne Klasse sondern ein ganzer Klassenverband. Dadurch wird der Fokus weg von der Wiederverwendung einzelner Klassen hin zur Wiederverwendung ganzer Architekturen verschoben.
Wieviel ein Framework von einer Anwendung eines bestimmten Gebiets abdeckt, hängt davon ab, wie weit man die benötigte Funktionalität standardisieren kann. Dies beeinflußt die Benutzung des Frameworks. Man unterscheidet zwei Arten: White-Box- und Black-Box-Wiederverwendung [9].
Erweitert man ein Framework durch Unterklassenbildung und dem Überschreiben von Methoden, spricht man von *White-Box-Wiederverwendung*. Das Framework legt dabei wichtige Teile und Beziehungen fest. Diese müssen durch anwendungsspezifische Teile ergänzt werden.
Will man ein White-Box-Framework erweitern, so muß man in dieses hineinschauen und das Zusammenspiel der verschiedenen Teile verstanden haben. White-Box-Wiederverwendung ist eine mächtige Art der Wiederverwendung, da man in den Unterklassen sehr flexibel Anpassungen und Erweiterungen vornehmen kann. Die Erweiterungen am Framework werden durch Programmierung neuer Klassen vorgenommen, was relativ aufwendig ist und ein gutes Verständnis für die im Framework implementierten Mechanismen erfordert. Trotzdem eignen sich White-Box-Frameworks gut für das Prototyping, da bereits umfangreiche Funktionalität und ein wiederverwendbares Design bereitstehen.
Ein *Black-Box-Framework* stellt verschiedene direkt benutzbare Klassen zur Verfügung. Es kann verwendet werden, ohne daß man versteht, wie die Objekte intern zusammenspielen. Typische Vertreter von Black-Box Frameworks sind Kapselungen von Grafiksystemen, objektorientierte Schnittstellen zu relationalen Datenbanken oder die Microsoft Foundation Class Library.
Die Black-Box-Wiederverwendung erlaubt nur beschränkte Anpassungen, dafür kann man sie mit gerin-

gem Aufwand erlernen und anwenden. Der Einsatz von Black-Box-Frameworks bedeutet, daß neue Anwendungen vorwiegend dadurch entstehen, daß bestehende Bausteine komponiert und konfiguriert werden. Dieses führt zu sehr kurzen Entwicklungszeiten und ist damit für das Prototyping speziell attraktiv.
Zunehmend gewinnen Rahmenwerke an Bedeutung, die ganze Anwendungsgebiete abdecken. Solche Application Frameworks bestehen aus einer generischen Architektur und einer Vielzahl von wiederverwendbaren Bausteinen (Black-Box- und White-Box-Frameworks), um entsprechende Anwendungen zu realisieren. Beispiele für Application Frameworks sind ET++ [12], fACTs++ (Objective Edge, www.objectiveEdge.com) und die Klassenbibliotheken verschiedener Smalltalk-Systeme. Application Frameworks sind ideal, um Anwendungen für ihr spezifisches Gebiet mit Prototyping zu entwickeln.

3. Erfahrungen mit objektorientiertem Prototyping

3.1 Die analysierten Projekte

Im Rahmen des GI-AK Prototyping haben wir neun industrielle Projekte untersucht, die explizit mit Prototyping gearbeitet haben. Vier davon setzten eine objektorientierte Entwicklungsplattform ein. Die Informationen sind in zwei Tabellen zusammengefaßt. Abbildung 1 charakterisiert jedes Projekt durch ein Kürzel und eine Kurzbeschreibung. Abbildung 2 zeigt für die einzelnen Projekte, welche Prototypen gebaut, welche Prototypingansätze angewandt und welche Werkzeugarten dazu eingesetzt wurden. Eine detaillierte Beschreibung dieser Projekte erfolgt in [1].

3.2 Konzepte und Werkzeugeinsatz

Bei der Analyse der untersuchten Prototyping-Projekte hat sich deutlich gezeigt, daß die Hoffnung trügerisch ist, mit Hilfe von Werkzeugen grundlegende konzeptionelle Probleme der Anwendungsentwicklung zu lösen. Grundlegend für die Lösung solcher Probleme sind dagegen etwa die Sichtweise und ein dazu passendes Leitbild, die Wahl einer geeigneten Methode und Entwicklungsstrategie sowie die Analyse der Anwendungssituation. Werkzeuge können immer nur die konstruktive Umsetzung einer vorhandenen konzeptionellen und methodischen Basis unterstützen. Wenn diese Basis fehlt, wird der Einsatz von Werkzeugen oft zum Selbstzweck und kann mehr schaden als nützen. Symptome eines verfehlten Werkzeugeinsatzes zeigen sich an mehreren Stellen:

- Die rasche Erstellung von Oberflächenprototypen und die Betonung der Programmierung (auch der objektorientierten) können leicht darüber hinwegtäuschen, daß die fachliche Grundlage fehlt. In diesem Sinne warnen wir vor einem "Rapid Prototyping". Erfahrungen zeigen, daß Prototypen ohne ausreichende fachliche und oft auch kommunikative Grundlage nicht "konvergieren", d.h. daß in den auswertenden Diskussionen mit den Benutzern immer neue und andere Anforderungen aufkommen.

Projektname	**Projektübersicht**
Kundenberatungssystem (KBS)	Eine auf Banksoftware spezialisierte Firma entwickelt ein neues Kundenberatungssystem. Die Hauptziele sind die Entwicklung einer homogenen Benutzungsschnittstelle und einer tragfähigen, erweiterbaren Systemarchitektur.
Editor für SWIFT-Nachrichten (ESN)	Ein Banksoftwarelieferant untersucht, ob eine neue Art SWIFT-Nachrichten abzuwickeln von ihren Kunden akzeptiert würde. Es soll herausgefunden werden, ob die Kunden bereit sind, ein interaktives Werkzeug zur Definition ihrer Nachrichtenströme zu verwenden.
Funktionseditor für mechatronische Systeme (FEM)	Eine Forschungsabteilung entwickelt ein System zur Verbesserung der Qualität mechatronischer Systeme (Systeme, die aus mechanischen und elektronischen Teilen bestehen). Eine Komponente dieses Systems ist eine Anwendung zum interaktiven Spezifizieren, Simulieren und Analysieren mechatronischer Systeme.
SWAPS-Manager (SM)	Die Forschungsabteilung einer Großbank entwickelt einen Prototyp einer Anwendung, die es SWAPS-Händlern ermöglicht, komplexe Szenarien zu definieren, zu simulieren und zu analysieren während sie am Telefon handeln.
Konzernbetreuerarbeitsplatz (KBA)	Eine Großbank will die Qualität der Betreuung ihrer Kunden mit einer neuen Generation von Softwaresystemen unterstützen. Dazu wurde eine erste Anwendung zur Unterstützung von Konzernbetreuern realisiert.
Fahrkartenautomat (FKA)	Im Rahmen der Einführung neuer Fahrkartenautomaten im Bereich des öffentlichen Verkehrs wird eine Ausschreibung durchgeführt. Aufgrund der Wichtigkeit der Qualität der Benutzungsschnittstelle wird als Teil der Angebote ein Benutzungsschnittstellenprototyp verlangt.
Graphische Debugger Benutzungsschnittstelle (GDB)	Ein Softwarehaus plant, die unter UNIX verfügbaren Entwicklungswerkzeuge auf ihre Plattform zu portieren. Dabei soll eine graphische Benutzungsschnittstelle für den Debugger entwickelt werden.
Multimediasystem zur Verkaufsberatung (MVB)	Ein großer Automobilhersteller will herausfinden, ob es sinnvoll ist, ein Multimediasystem zur Verkaufsberatung zu entwickeln. Das System soll dem Kunden Produktinformationen in Form von gesprochenem Text, zwei- und dreidimensionalen Bildern sowie Filmen anbieten.
Vorkalkulations- und Projektabwicklungssystem (VPS)	Ein auf den Bau von großen Stahlverarbeitungsanlagen spezialisiertes Unternehmen will seine Entwicklungsprozesse mit einem neuen Vorkalkulations- und Projektabwicklungssystem besser in den Griff bekommen. Die Softwaretechnikabteilung einer Universität bekommt den Auftrag, ein solches zu entwickeln.

Abb. 1: Die analysierten Projekte

Dies ist ein deutliches Zeichen, daß keine gemeinsame Basis für ein Softwareprojekt vorhanden ist.

- Ähnliches gilt beim Einsatz von Frameworks. Frameworks sind teilweise technisch extrem komplex und jenseits von kleinen Beispielanwendungen nur mit fundierten softwaretechnischen Kenntnissen einzusetzen. Dazu müssen die dahinterstehenden fachlichen Konzepte verstanden sein, da sich sonst kaum sinnvoll die Komponenten konstruieren lassen, die ein Framework erst zu einer einsatzfähigen Anwendung machen.

Projekte		KBS	ESN	FEM	SM	KBA	FKA	GDB	MVB	VPS
Prototyparten	Demonstrations PT	x	x			x		x	x	x
	Funktionaler PT		x		x		x		x	x
	Labormuster	x		x			x	x	x	
	Pilotsystem	x	x	x	x	x				x
Angewandte Prototypingansätze	Explorativ	x				x	x		x	
	Experimentell	x				x	x	x	x	
	Evolutionär	x	x	x	x	x				x
Eingesetzte Werkzeuge	HyperCard	x				x	x			
	Interface Builder	x						x	x	
	4GS					x				x
	Framework	x	x	x	x	x				

Abb. 2: Prototyparten, Prototypingansätze und eingesetzte Werkzeuge

- Generell können technikzentrierte Ansätze als Argument für das DV-Management gelten, sich nicht mit grundsätzlichen Problemen der Anwendungsentwicklung auseinanderzusetzen. Hier sehen wir eine "Mitschuld" von Werkzeuganbietern und einigen Beratungsunternehmen, die den Eindruck erwecken, durch Werkzeugeinsatz oder eine objektorientierte Programmiersprache ließen sich unter Beibehaltung der traditionellen Vorgehens- und Sichtweise grundlegende Probleme meistern. Kennzeichen dieser Problemlage sind völlig illusionäre Terminplanungen unter besonderer Vernachlässigung einer ausreichenden Analyse des Anwendungsbereichs. Dazu kommt oft eine personelle Unterdeckung des Projektes. Dies alles geschieht mit der falschen Hoffnung auf die zu erwartende ungeheure Effizienz der neuen Umgebung. Dabei wird sogar gelegentlich auf eine Einarbeitungsphase für die verwendete Entwicklungsumgebung verzichtet, da diese für das Prototyping ja ohne Aufwand einsetzbar sei.

3.3 Neue Sichtweise auf den Entwicklungsprozeß

Die Sichtweise auf den Entwicklungsprozeß spielt bei Werkzeugauswahl und -verwendung eine wichtige Rolle. Vielfach dominiert die sog. Produktsicht, d.h. Softwareentwicklung wird als ein Produktionsprozeß verstanden, der ähnlich der industriellen Herstellung von Konsumgütern in einem festgelegten und möglichst arbeitsteiligen Produktionsprozeß zu einem vorab definierten Produkt führt. Entsprechend sind viele Entwicklerorganisationen in spartenorientierte Abteilungen wie Kundenbetreuung, Entwicklung, Organisation, Vertrieb, Infrastruktur etc. aufgeteilt. In den letzten Jahren ist deutlich geworden, daß Software so kaum anwendungsorientiert entwickelt werden kann. An die Stelle der Produktsicht tritt vermehrt eine Projekt- oder Prozeßsicht, in der Softwareentwicklung vor allem als Lern- und Kommunikationsprozeß aller beteiligten Gruppen gesehen wird [6]. Die Folgen für das Prototyping sind offensichtlich. Während die Produktsicht Prototyping vorrangig als Instrument der Anforderungsanalyse einsetzt, versteht die Prozeßsicht Prototyping als grundlegendes Mittel zur Förderung von Lernprozessen und als Grundlage der Kommunikation und Kooperation im gesamten Entwicklungsprozeß. Dies hat seinerseits Auswirkungen auf die Organisation, da sich die spartenorientierte Abteilungsstruktur als sperrig für solche anwendungs- und prozeßorientierten Projekte mit integrierten Teams erweist.

3.3 Leitbilder

Das Leitbild bei der Softwareentwicklung prägt den Prototyping-Prozeß entscheidend. Daher ist in jedem Projekt zum frühest möglichen Zeitpunkt zu klären, welche (Wert-) Vorstellungen und damit leitenden Ideen mit einem Projekt verbunden sind. Passend zur

Produktsicht finden wir auf der einen Seite die Idee der möglichst vollständigen Ablaufsteuerung und Automatisierung, d.h. das Bild der Fließbandarbeit. Dagegen finden wir gerade bei der Projekt- oder Prozeßsicht die Idee von Software als unterstützendem Hilfsmittel für qualifizierte menschliche Arbeitstätigkeiten, d.h. das Bild vom gut ausgestatteten Arbeitsplatz [8]. Derartige Systeme werden als teilselbständige Komponenten realisiert, die situativ vom Benutzer gehandhabt werden. Es sollte klar geworden sein, daß Systeme nach solchen Leitbildern gut mit den beschriebenen objektorientierten Prototypingtechniken realisierbar sind.
Jedes Softwareprojekt wird sich seinen Platz im Spektrum zwischen völliger Ablaufsteuerung und reiner Unterstützung suchen müssen. Dieser Platz bestimmt aber die Möglichkeiten, objektorientiertes Prototyping einzusetzen: Eine rigide Ablaufsteuerung (auch in der modernen Variante des Workflow) und Prozeßautomatisierung als Leitbild wird bei den Beteiligten selten auf die Akzeptanz stoßen, die Voraussetzung für eine kooperative und offene Arbeit mit Prototypen zwischen Anwendern, Benutzern und Entwicklern ist. Nur wenige Anwender und Benutzer haben bei Software die Wunschvorstellung einer fließbandartigen Tätigkeit. Dagegen sehen viele Anwender in einer besseren Unterstützung von weitgehend selbstbestimmter Arbeit durch geeignete Hilfsmittel ein durchaus anstrebenswertes Entwicklungsziel.

3.4 Werkzeuge bestimmen den Entwicklungsprozeß

Wenn Werkzeuge und Entwicklungsumgebungen auch kaum die konzeptionellen Probleme der Softwareentwicklung lösen können, so hat ihr Einsatz doch Auswirkungen auf den Entwicklungsprozeß weit jenseits der Konstruktion eines Prototyps. Besonders Entwicklungsumgebungen mit ihren Mitteln der graphischen Programmierung legen explizit oder implizit eine bestimmte Vorgehensweise und eigene Modellierungskonzepte fest.
Da ist zunächst die zeitlich-logische Abfolge beim Bau von Prototypen: Auch im Bereich objektorientierter Entwicklungsumgebungen stellen wir hierbei eine negative Tendenz fest: Durch die Verwendung von Interface-Buildern und graphischen Programmierkomponenten werden die Anwendungssysteme meist von der Benutzungsoberfläche „nach unten“ programmiert.
Aus mehreren Gründen macht es u.E. wenig Sinn, zuerst die Oberfläche zu entwerfen und dann daraus die fachliche Komponente der Anwendung abzuleiten. Ein offensichtliches Problem kommt durch die enge Verknüpfung von Form und Inhalt zum tragen. So verlockend es für Entwickler sein mag, zunächst ein Oberflächenlayout mit den Benutzern abzustimmen und dieses dann schrittweise mit fachlicher Funktionalität anzureichern, so wenig tragfähig ist diese Vorgehensweise, wenn die fachlichen Inhalte der zukünftigen Anwendung den Beteiligten prinzipiell noch nicht klar sind.
Sind diese Voraussetzungen gegeben, dann wollen wir auch bei Verwendung moderner objektorientierter Entwicklungsumgebungen vor der Illusion warnen, daß die Veränderung und Weiterentwicklung eines Prototyps in der Diskussion von Benutzern und Entwicklern vor Ort interaktiv am Bildschirm erfolgen kann. Dies mag für einzelne Aspekte der Oberflächengestaltung der Fall sein. Aber bei fachlichen oder grundsätzlich konzeptionellen Änderungen ist dies nicht in „Realzeit“ möglich und im Regelfall nach unserer Erfahrung auch nicht erforderlich. Sehr viel wichtiger ist, daß die Entwickler verstehen, welche Änderungen erwünscht sind, um diese dann in absehbarer Zeit zu realisieren. Im übrigen sind nur wenige Prototyping-Werkzeuge so gebaut, daß sie extrem kurze Änderungszyklen unter Beibehalt einer softwaretechnisch tragfähigen Architektur überhaupt zulassen. Die eigentlichen Prototyping-Sitzungen sollten daher der Diskussion und dem Ausprobieren vorbehalten sein. Dabei hat es sich sehr bewährt, wenn ein „Drehbuch“ die Anwendungssituationen skizziert, die dabei durchgespielt werden.

3.5 Frameworks verändern den Entwicklungsprozeß

Drastisch sind auch die Auswirkungen des Einsatzes von Frameworks in der Softwareentwicklung. Hier spielen zwei Dinge eine Rolle:

- Die Entwicklung von Anwendungssystemen bedeutet, daß in einen vorgegebenen Rahmen mit eigenen strukturellen und dynamischen Anforderungen neue Komponenten konstruiert und eingepaßt werden müssen.
- Die Weiterentwicklung des Frameworks selbst muß über das einzelne Softwareprojekt hinaus geplant und organisatorisch unterstützt werden. Wir stellen als Tendenz fest, daß die Frameworkentwicklung eigenständigen Projektcharakter gewinnt.

Dies bedeutet für viele Entwicklerorganisationen eine erhebliche organisatorische und strategische Umstellung [2]. Neben den hohen Kosten, die dies zur Folge hat, darf nicht übersehen werden, daß dadurch teilweise eine neue Form von Herstellerabhängigkeit eingegangen wird. Wir haben erlebt, daß Entwicklerorganisationen sich auf der einen Seite mühsam von der Bindung an einen Hardwarehersteller gelöst haben, um andererseits sich nicht minder intensiv einem Werkzeughersteller oder dem Lieferanten eines Frameworks zu verpflichten.

3.6 Entwicklungswerkzeuge und Anwendungsarchitekturen

Die von uns festgestellte Tendenz zu einer oberflächenorientierten Vorgehensweise führt häufig zu Softwarearchitekturen, die keine saubere Trennung von Darstellung und Handhabung an der Oberfläche sowie fachlicher Funktionalität haben. Die dabei oft verwendeten Generatoren legen nahe, Oberflächenkomponenten (z.B. ein Eingabefeld) unmittelbar mit fachlichen Werten (z.B. einer Vertragslaufzeit) zu verknüpfen, wobei dann große Teile der fachlichen Dynamik implizit in der Oberflächenkomponente realisiert werden. Damit ist aber die erwünschte Trennung zwischen Funktionalität und Handhabung weitgehend aufgehoben.
Durch den direkten Zugriff von der Oberfläche auf fachliche Werte läßt sich zudem die Konsistenz eines fachlichen Objektes wie "Vertrag" nur mit Mühe wahren. Typischerweise wird bei diesem Vorgehen die Diskussion über die fachliche Substanz solcher Objekte vollkommen vernachlässigt. Dies führt zu fachlichen Komponenten, die softwaretechnisch explizit als reine "Datenobjekte" mit generischen „Setze“ und „Gib“ Operationen abgebildet sind. Objektorientierte Architekturen dieser Art sind schwer verständlich und kaum weiterentwickelbar. Die zunehmend wichtiger werdende Diskussion um moderne Softwarearchitekturen hat auch Auswirkungen auf das objektorientierte Prototyping. Grundsätzlich stimmen wir der häufig zitierten Aussage zu, daß zur Bewältigung der Komplexität umfangreicher interaktiver Anwendungen nur das objektorientierte Paradigma geeignet erscheint. Deshalb ist an Werkzeuge zum Prototyping die Frage zu richten, in welchem Umfang sie in ein solches objektorientiertes Entwicklungsumfeld passen. Bis auf die Application Frameworks ist bei den meisten Entwicklungsumgebungen wie VisualWorks, VisualAge oder NextStep aus unserer Sicht noch keine wirklich positive Antwort zu geben. Denn die Meinung, daß durch einfaches interaktives Verbinden von Oberflächenkomponenten komplexe Anwendungen „zusammengezogen“ und „verdrahtet“ werden können, ist irrig. Solche Anwendungen lassen sich schon ab mittlerer Komplexität kaum noch konsistent ändern oder weiterentwickeln.
Grundsätzlich bestehen also Probleme, wenn Prototypen neben der fachlichen Funktionalität und Handhabung auch die Architektur und die softwaretechnischen Grundprinzipien des zukünftigen Anwendungssystems modellieren sollen. Dies gilt ganz besonders für Werkzeuge und Umgebungen, bei denen werkzeugbedingt die fachliche Komponente in der Form von interpretierbaren Programmfragmenten eng in die Oberflächenkomponente integriert ist.

4. Empfehlungen für das objektorientierte Prototyping

4.1 Vorgehensmodell

Da die Entwicklung interaktiver Systeme vielfach anwendungs- und benutzerorientiert gesehen wird, muß Prototyping als Bestandteil einer evolutionären Vorgehensweise Ausdruck einer neuen Projektkultur werden. Diese neue Projektkultur ist dann entstanden, wenn diese Vorgehensweise der Normalfall ist und nicht speziell betont werden muß. Das bedeutet, in der Entwicklerorganisation traditionelle Vorgehensmodelle zu ersetzen. Aber das neue Konzept muß auch den Kunden gegenüber verdeutlicht werden, denn schließlich hat eine evolutionäre Vorgehensweise Auswirkungen bis in die Vertragsgestaltung.
Nur innerhalb eines veränderten Gesamtkonzepts können Werkzeuge beim Prototyping ihren vollen Nutzen erbringen. Wir weisen nochmals darauf hin, daß im Rahmen des Prototyping, so wie wir es verstehen, der fachlichen Klärung im Projekt vorrangige Bedeutung zukommt.

4.2 Objektorientierte Prototypen und ihre Gestaltung

Fachliche Stimmigkeit eines Prototyps und des späteren Systems ist eng verbunden mit einer benutzergerechten Handhabung.
Denn fachliche Stimmigkeit umfaßt sowohl die fachliche Logik, also die „interne“ Funktionalität, als auch die an der Oberfläche sichtbare Begrifflichkeit und Darstellung. Dazu kommt die Handhabung als die Art und Weise, wie Benutzer mit dem System umgehen, mit ihm „hantieren“ können. Mit Blick auf die unterschiedlichen Ein- und Ausgabemedien stellt sich die Frage, wie gut ein System für die Erledigung der entsprechenden fachlichen Aufgaben „in der Hand liegt“. Dies bedeutet, daß Oberflächengestaltung heute nicht mehr ein nachrangiger „Aufsatz“ auf ein „an sich“ funktionstüchtiges System sein kann. Vielmehr müssen Handhabung, Darstellung und fachlich-logische Aspekte als eng verzahnt und in ihrer gegenseitigen Bedingtheit erkannt werden. Da hier langjähriges Erfahrungswissen und gereifte Konzepte fehlen, kann u.E. nur das experimentell empirische Vorgehen des objektorientierten Prototyping weiterführen.

4.3 Grenzen der Formalisierung reaktiver Systeme

Die formalen Verfahren der Spezifikation greifen für interaktive Anwendungssoftware besonders schlecht. Wie Peter Wegner [11] aufzeigt, ist kein Formalismus denkbar, um interaktive objektorientierte Systeme mit ihren Oberflächenkomponenten in Gänze

widerspruchsfrei zu spezifizieren. Damit ist ein systematischer formaler Weg von der fachlichen Spezifikation zum korrekten lauffähigen Softwaresystem nicht gangbar. In der Konsequenz heißt dies, daß große reaktive Systeme mit Benutzerinteraktion nicht von einem "allwissenden" Automaten kontrolliert und gesteuert werden sollten. Kleine, überschaubare Komponenten müssen die dezentrale Regelung übernehmen und im Konfliktfalle durch den Eingriff eines Menschen auf einer anwendungsbezogenen Ebene "justierbar" sein. Durch die Objektorientierung lassen sich diese überschaubaren Komponenten in wesentlichen Aspekten auch formal, etwa nach dem Vertragsmodell [10] beschreiben. Die Komplexität des Zusammenspiels aller Komponenten eines interaktiven System läßt aber Prototyping zur unumgänglichen Projektaktivität werden. Dabei kommt funktionalen Prototypen eine besondere Rolle zu. Hier ist der Weg weg von reinen Oberflächenprototypen vorgezeichnet.

5. Ausblick und Zusammenfassung

Was bleibt mit Blick auf die Ergebnisse unserer Studie und die inzwischen sichtbar gewordenen Entwicklungen zu sagen? Zunächst haben sich bei den Entwicklungswerkzeugen einige Tendenzen verdeutlicht.

Tendenziell scheinen die Unterschiede zwischen den einzelnen Werkzeugarten, die beim Prototyping sinnvoll eingesetzt werden, zu verschwinden. Interface-Builder, Application Frameworks und 4. Generationssysteme mit (partieller) interpretativer Ausführung verschmelzen. Beispiele dafür sind die kommerziellen Smalltalk-Systeme (VisualWorks, Visual Smalltalk, VisualAge etc.), die selbst zur Zeit in einer Phase der Fusionierung sind.

Diese Kombination von aufeinander abgestimmten Werkzeugen führt dank der integrierten Interface-Builder zu einfach konstruierbaren Prototypen, zur flexiblen Erweiterbarkeit dank der Application Frameworks und, aufgrund der entsprechenden Abfragesprachen zur standardisierten Datenabfrage, -Anzeige und -Manipulationsfunktionalität.

Die Analyse der Interface-Builder zeigt einen interessanten Aspekt: Offenbar gibt es mittlerweile ein konsolidiertes Erfahrungswissen darüber, wie die Bausteine einer interaktiven Oberfläche aussehen sollten. Dies zeigen auch die Oberflächenkomponenten der neueren Entwicklungsumgebungen. Ein starker Trend zur Vereinheitlichung zeichnet sich bei den Mechanismen zum Datenaustausch und zur einfacheren Wiederverwendung von Basisfunktionalität ab (z.B. OLE und OpenDoc). Problematisch kann hier werden, daß der einfachere Zugang und Austausch zwischen den Komponenten zu einer weiteren Datenflut führt (wie im WWW, das selbst zum Trend in diesem Bereich geworden ist).

Arbeiten im Forschungs- und Entwicklungsbereich deuten auf eine klarere Trennung von offenen Entwicklungsumgebungen und Anwendungssoftware.

Generell kommt der Architektur großer interaktiver Systeme mehr Bedeutung zu. Für das Prototyping bietet dies die Lösung zweier derzeit recht gravierender Probleme: Die mangelnde Möglichkeit, generierte Oberflächen als verständliche Bausteine für die Softwareentwicklung auch jenseits von Prototypen zu verwenden, und die Schwierigkeiten, die die Änderung von Oberflächen bereitet, wenn sie um fachlichen Komponenten ergänzt worden sind.

Die heutigen Application Frameworks werden sich unter Prototypinggesichtspunkten ebenso verändern. Neben White-Box-Frameworks, die durch Reimplementierung spezialisiert werden müssen, werden in Zukunft auch Black-Box-Frameworks treten, die durch geeignete Werkzeuge oder Umgebungen zu spezifischen Anwendungen konfiguriert und komponiert werden können. Die Konstruktion eines passenden Oberflächenprototypen wird dann eine Teilaufgabe bei der Konfigurierung einer solchen Anwendung sein. Systeme dieser Art werden derzeit in Forschungseinrichtungen erprobt.

Was die Rolle des Prototyping in der industriellen Praxis anbetrifft, so hat sich im Laufe der letzten Jahre auf breiter Front wenig verändert. Noch immer ist für uns erstaunlich, in welch großem Umfang die industrielle Softwareentwicklung die Erkenntnisse und Erfahrungen aus mehr als einem Jahrzehnt Praxis mit Prototyping als generellem Konzept ignoriert. Noch immer dominieren konventionell und teilweise konzeptionslos durchgeführte Projekte mit oft bedenklichen Ergebnissen. Dies scheint ein auf den ersten Blick unverständlicher Umstand, da die Ergebnisse besonders von objektorientierten Prototyping-Projekten durchweg als ermutigend zu bezeichnen sind.

Schauen wir allerdings in das organisatorische Umfeld, wird dieses Mißverhältnis verständlicher. Dann stellen wir nämlich fest, daß die westlichen industriellen Großunternehmen generell in einer Strukturkrise stecken. Vielfach wird versucht, mit neuen Werkzeugen oder technischen Maßnahmen unter Beibehaltung der konventionellen Herangehensweise die erwünschte Veränderungen hin zu qualitativ hochwertiger und anwendungstauglicher Software herbeizuführen. Zu befürchten bleibt, daß sich eine grundlegend anwendungsorientierte Denkweise erst in den Unternehmensleitungen durchsetzen muß, ehe sich an der Softwareentwicklung wirklich Entscheidendes ändert.

Ergänzend zu dieser, eher pessimistischen Einschätzung sehen wir die Situation im Forschungs- und Wissenschaftsbereich. Zumindest für den deutschsprachigen Raum gilt, daß Prototyping und damit Ansätze zur experimentellen und partizipativen Softwareentwick-

lung von der vorherrschenden Informatik als unwissenschaftlich und "soft" abqualifiziert wird. Auch die systematische und fundierte Auseinandersetzung mit der Objektorientierung ist eher die Ausnahme. Immerhin gibt es einige Ansätze, die sich um eine Komplementarität von formalen Methoden und experimentell evolutionärer Vorgehensweise unter dem objektorientierten Paradigma bemühen (vgl. [11]). Nicht zu unterschätzen sind auch die Bemühungen, konsolidiertes Erfahrungswissen bei der Lösung wiederkehrender Konstruktionsprobleme in Form von Entwurfsmustern zu dokumentieren (vgl. [7]).

Abschließend ein Wort zur erfreulichen Rolle einzelner industrieller Unternehmen. Einige, besonders kleine und mittelständige Unternehmen haben die sich auf aktuelle Trends wie Client-Server Computing, Objektorientierung und Software für kooperative, verteilte Arbeit spezialisiert. In diesen Bereichen sind Prototyping und die damit verbundenen Entwicklungsstrategien zur Alltagspraxis geworden. Auch DV-Abteilungen einiger großer Organisationen, die verstärkt auf kundenzentrierte Dienstleistungen umstellen, haben diesen Trend erkannt. Von diesen Bereichen werden neue Impulse für das Prototyping ausgehen. Wir erwarten, daß sich hier die wechselseitige Beziehung von Entwicklungsstrategie und Unternehmensorganisation am stärksten zeigen wird.

Literatur

[1] Bäumer D., et. al. (1994): Prototyping von Benutzeroberflächen, UBILAB Technical Report 94.9.2, Union Bank of Switzerland, UBILAB, 8021 Zürich.

[2] Birrer A., Bischofberger W.R., Eggenschwiler T. (1995): Wiederverwendung durch Frameworktechnik - vom Mythos zur Realität, in OBJEKTspektrum, September/Oktober.

[3] Bischofberger W.R., Pomberger G. (1992): Prototyping-Oriented Software Development – Concepts and Tools, Springer-Verlag.

[4] Budde R., Kautz K., Kuhlenkamp K., Züllighoven H. (1992): Prototyping– An Approach to Evolutionary System Development., Springer-Verlag.

[5] Floyd C. (1984): A Systematic Look at Prototyping, in Budde R., et. Al. (Eds.), Approaches to Prototyping, Springer Verlag.

[6] Floyd C. (1987): Outline of a Paradigm Change in Software Engineering. In: G. Bjerknes, P. Ehn, M. Kyng (Eds.), Computer and Democracy, Avebury, Gower Publishing Company Limited, Aldershot.

[7] Gamma E., Helm R., Johnson R., Vlissides J. (1994): Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley.

[8] Gryczan G., and Züllighoven H. (1992): Objektorientierte Systementwicklung - Leitbild und Entwicklungsdokumente. In: Informatik-Spektrum 15 (5), 264-272.

[9] Johnson R.E. and Russo V.F. (1991): Reusing Object-Oriented Designs, Department of Computer Science, University of Illinios at Urbana-Champaign.

[10] Meyer C. (1988): Object-oriented Software Construction. Prentice Hall, Hemel Hempstead.

[11] Wegner, P. (1995): Models and paradigms of Interaction, Department od Computer Sience, Brown University, Technical Report, No. CS-95-21.

[12] Weinand A., and Gamma E. (1994): ET++ - a Portable, Homogenous Class Library and Application Framework. Computer Science Research at UBILAB, Strategy and Projects; Proceedings of the UBILAB '94 Conference, Zürich, September 1994. Universitätsverlag Konstanz.