# Framework-basierte Anwendungsentwicklung oder Was ist das Problem?

Guido Gryczan, Stefan Roock, Henning Wolf, Heinz Züllighoven

Wenn sich in den ersten Wochen des kommenden Jahres der Rauch in den IT-Abteilungen etwas verzogen hat und die „Restrisiken“ der Y2K-Projekte beseitigt sind, werden viele IT-Manager erkennen, daß sich an der grundlegenden Misere der Softwareentwicklung wenig geändert hat. Die meisten Y2K-Projekte haben konzeptionell von der Hand in den Mund gelebt. Dringend anstehende Strukturprobleme wurden auf Eis gelegt. Diese Probleme können aber auf die Dauer nicht ausgesessen werden und es stellt sich die Frage, was tun?

Die Situation im Unternehmensumfeld hat sicher eher verschärft: Der wachsende Konkurrenzdruck offener Märkte und kritische Kunden verlangen nach einer offensiven kundenorientierten Geschäftstrategie. Wenn dabei die Kostenseite stimmen soll, dann müssen verbesserte Serviceleistungen durch eine passende IT-Infrastruktur unterstützt werden. Doch was heißt passend? Je nach Standpunkt finden wir unterschiedliche Antworten.

- *Anwender* wollen eine optimale Unterstützung bei der Erledigung ihrer täglichen Aufgaben. Ändern sich die Aufgaben oder kommen neue Kundenwünsche auf, dann soll rasch eine funktionierende und handhabbare Softwarelösung bereitstehen.
- *Manager einer Anwendungsorganisation* wollen schnell auf die Entwicklungen des Marktes und die Anforderungen ihrer Kunden reagieren: Time-to-market ist ein zentrales Anliegen. Die kostengünstige IT-Unterstützung der Mitarbeiter ist dabei eine Randbedingung.
- *Anwendungsentwickler* wollen die fachlichen Probleme ihrer Kunden durch Software lösen. Neue Technologien sind dabei Mittel zum Zweck: Software muß schnell und fehlerfrei funktionieren. Warum eine bewährte Technologie verlassen, wenn ein Kundenwunsch damit zufriedenstellend erfüllt wird?
- *IT-Manager* müssen mit begrenzten Mitteln haushalten und gleichzeitig den Technologiewandel im Auge behalten: Läßt sich der aktuelle Service besser mit bereits vorhandenen Technologien unterstützen? Erfordern die Anforderungen von morgen ganz andere Technologien?

Das ist der Hintergrund, vor dem wir unser Thema beleuchten: Welchen Beitrag können Frameworks (deutsch auch Rahmenwerke genannt) für Anwendungssoftwareentwicklung leisten? Wer sollte sie unter welchen Umständen einsetzen und was ist dabei zu beachten?

Unsere Einschätzungen beruhen dabei auf praktischen Erfahrungen aus industriellen und aus Forschungsprojekten, in denen objektorientierte Frameworks eingesetzt werden. Wir sind selbst Frameworkentwickler für interaktive Anwendungen (vgl. http://www.jwam.de).

***<<Kasten>>***

## *Frameworks und Bausteinbibliotheken*

In objektorientierten Softwaresystemen sind Klassen und Objekte das primäre Strukturierungsmittel. Sie erlauben die Kapselung von Daten und Funktionen in einer semantisch sinnvollen Einheit, der Klasse. Zusammengehörige Klassen können in Bausteinbibliotheken gesammelt und unmittelbar wiederverwendet werden. Beispiele sind Behälterklassen oder Klassen für den Anschluß relationaler Datenbanksysteme.
Allerdings stoßen wir mit Bausteinbibliotheken alleine schnell an Grenzen:

- Das Verhalten kann nur schwer modifiziert werden, ohne den Quellcode der Bausteinklassen zu ändern.
- Die Bausteinklassen erlauben im wesentlichen die Wiederverwendung von Code. Sie sagen nichts über die sinnvolle Architektur einer Anwendung. Der Kontrollfluß der Anwendung muß jedesmal neu entworfen werden.

Gerade bei Produktfamilien müssen so gleiche Problemstellungen bei verschiedenen Produkten zwar ähnlich, aber doch mit Variationen gelöst werden.
Der objektorientierte Vererbungsmechanismus erlaubt uns, die Grenzen reiner Bausteinklassen zu überwinden. Das Verhalten einer Klasse kann adaptiert werden, ohne den Quellcode der Klasse verändern. Wir können auf einer allgemeinen Ebene in Klassen das Zusammenspiel von Objekten definieren. Klassen, die den prinzipiellen Kontrollfluß für eine Anwendung vorgeben, können in Frameworks zusammengefaßt werden. Die konkreten Anwendungen werden so entwickelt werden, daß spezifische Klassen und ihre Operationen mit dem im Framework festgelegten Kontrollfluß kombiniert werden. Frameworks sind also eine Klassenstruktur, die ein

abstraktes Design für eine Familie von Problemen zur Verfügung stellen. Sie ermöglichen damit neben der Wiederverwendung von Code auch die Wiederverwendung von Konzepten auf einer höheren Ebene („Wiederverwendung im Großen“). Beispiele für Frameworks sind ET++, Taligent, das Smalltalk-System und das von uns entwickelte JWAM-Framework (siehe http://www.jwam.de).

## *Softwareentwicklung mit Frameworks*

Objektorientierte Programmiersprachen ermöglichen die Wiederverwendung von Code, von Design und von Architekturen. Trotzdem sind in vielen objektorientierten Projekten die Ziele der Wiederverwendung nicht erreicht worden. Woran liegt das? Ein häufig vernachlässigter Aspekt ist das Management von Wiederverwendung. Durch Wiederverwendung entsteht prinzipiell ein zweistufiger Entwicklungsprozeß (siehe Abbildung 1). Eine Entwicklerorganisation führt in der Regel mehrere Anwendungsentwicklungsprojekte parallel durch. Die daraus entstehenden Anwendungen werden beim Kunden eingesetzt. Die Anwendungsentwicklungsprojekte verwenden ein gemeinsames Framework.

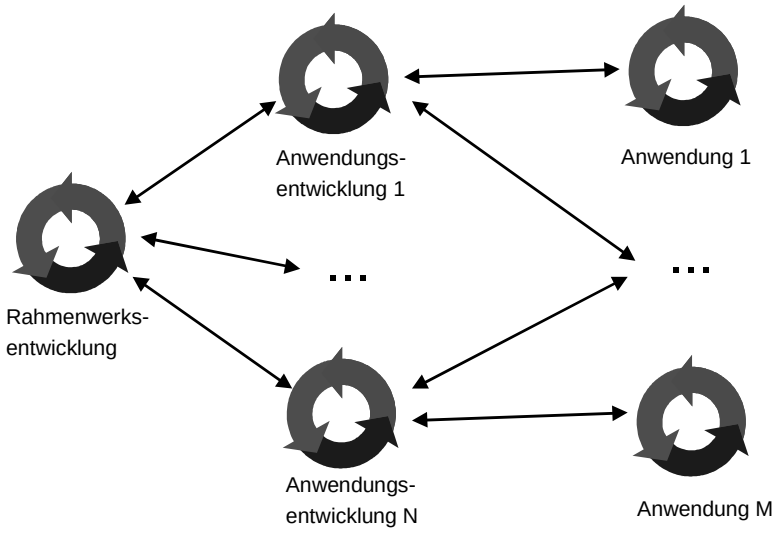


**Abbildung 1: Zweistufiger Entwicklungsprozeß bei Frameworksverwendung**

Aber wer entwickelt das Framework? Anwendungsentwickler können dies schlicht nicht „nebenbei“ mit erledigen. Schließlich stellen Frameworks generalisierte Lösungen zur Verfügung, die in möglichst vielen Projekten eingesetzt werden sollen. Anwendungsentwicklern fällt es aber oft schwer, vom konkreten Projekt zu abstrahieren. Zudem können sie sich oft nicht ausreichend mit neuen Technologien beschäftigen.
Eine zusätzliche Instanz sollte also für die Frameworkentwicklung zuständig sein – die „Architekturgruppe“. Sie gibt mit dem Framework die Architektur für die Anwendungsprojekte vor. Die Mitglieder der Architekturgruppe müssen hochqualifizierte Entwickler mit reichhaltiger Praxiserfahrung aus Projekten sein:
Hier ist allerdings Fingerspitzengefühl gefordert, um zu verhindern, daß sich zwischen Framework- und Anwendungsentwicklern feste Fronten bilden. Dann droht der Architekturgruppe ein ähnliches Schicksal, wie den meisten M&V-Abteilungen (M&V = Methoden und Verfahren): Technologie- und Methodenproduktion auf Halde.
Deshalb sollen Frameworkentwickler zumindest zeitweise in Anwendungsprojekten mitarbeiten. Sie können so die Probleme der Anwendungsentwickler am „eigenen Leib“ erfahren und gleichzeitig ihr Wissen über das Framework in die Projekte tragen. Damit diese Zusammenarbeit reibungslos funktionieren kann, müssen Frameworkentwickler mehr als technische Kompetenzen besitzen:

- Teamfähigkeit
- Anpassungsfähigkeit
- Kommunikationsfähigkeit
- Erfahrungen in objektorientierten Projekten
- Fachliches Wissen über die zu unterstützende Domäne (z.B. Banken)
- Fundierte Kenntnisse über Architekturen und Entwurfsmuster

Daß die „weichen“ Faktoren hier an erster Stelle stehen, bedeutet, daß sie mindestens so wichtigsten sind wie die anderen Faktoren. Allerdings werden sie viel häufiger vernachlässigt.

## *Aufbau von Frameworks*

Wenn klar ist, was ein Framework ist und wie es in Softwareprojekten eingesetzt werden kann, stellt sich die Frage nach den Bauprinzipien und der Strukturierung von Frameworks.

Aus der nachfolgenden Kritik einer rein technischen Architketur, entwickeln wir eine anwendungsfachliche Strukturierung.

### Eine technisch motivierte Struktur von Frameworks reicht nicht aus

Die wohl momentan am intensivsten diskutierte Softwarearchitektur wird in der Literatur als Three-Tier-Architektur bezeichnet. Grundlegend für diese Architektur ist eine Aufteilung des Gesamtsystems entlang der drei Schichten Datenschicht, Funktionalität und Präsentation.

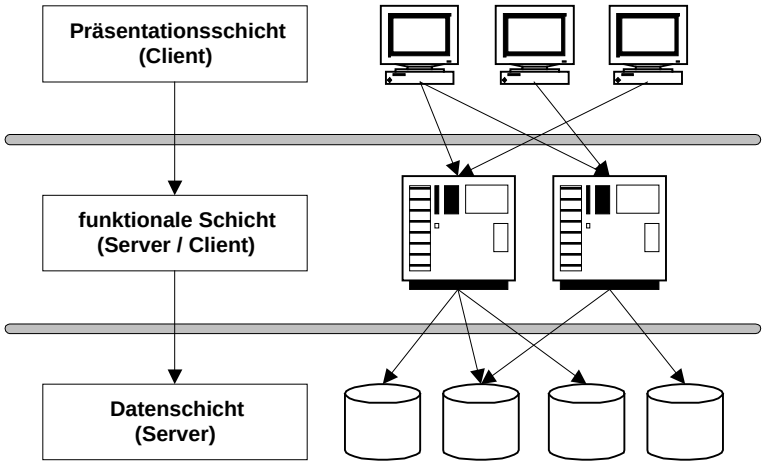


**Abbildung 2: Schema der Drei-Schichten-Architektur**

So sinnvoll diese Konfiguration auch für eine Aufteilung von Prozessen eines laufenden Systems auf unterschiedliche Rechner sein mag, sie sagt noch nichts darüber aus, wie die Teile eines Frameworks *geschnitten* werden und wie sie *interagieren* können. Aber auch die versprochenen technologischen Vorteile werden selten eingelöst. Nach allgemeinen Erfahrungen kann bei dieser Architektur leicht eine Schicht gegen eine andere technische Implementierung ausgetauscht werden, aber solche Änderungen treten während der gesamten Lebensdauer einer Anwendung relativ selten auf.

Die *getrennte Entwicklung der Schichten* ist selten möglich. Denn in der Regel führen fachliche Änderungen sowohl zu Erweiterungen an der Präsentations- als auch an der Datenschicht.

Auch die einfache Versionierung und Ausbreitung einer derartigen Architektur ist fraglich, da die dabei entstehenden Kosten i.d.R. nicht von der Größe des auszubreitenden Programmpakets abhängen, sondern durch die Ausbreitung auf die verschiedenen Rechner an sich enstehen.

### Anwendungsfachliche Konzepte bestimmen den Frameworkaufbau

Wenn Kundenorientierung und die adäquate Unterstützung von täglichen Aufgaben das vorrangige Motiv für den Einsatz von IT sind, dann muß sich dies auf den Aufbau von Frameworks auswirken. Die UML-Autoren haben recht, wenn schreiben, daß (Software-) Modelle fachlichen Konzepte widerspiegeln müssen. Wir nennen dieses Prinzip „Strukturähnlichkeit“:

- *Fachliche Anforderungen* sind der Dreh- und Angelpunkt jedes Anwendungsframework. Hier wird der fachliche Kern einer Domäne modelliert. Fragen, die hier gestellt werden, sind: Was sind die zentralen Arbeitsgegenstände der Anwender? Mit welchen Mitteln werden sie von wem bearbeitet? In welche Produktbereiche oder Sparten ist das Unternehmen aufgebaut? Wie „wandern“ diese Arbeitsgegenstände als Vorgänge durch die Organisation? Konstruktiv und konzeptionell sollten die entsprechenden Modelle den fachlichen Kern jedes Anwendungsframework bilden.

Doch der fachliche Kern reicht noch nicht aus, um alle Anforderungen an ein Framework zu erfüllen. Auf der Anwendungsebene soll ein Framework ja zwei gegenläufige Forderungen unterstützen. Unterschiedliche Arbeitsplätze (oder Arbeitsplatztypen) müssen die gleiche fachliche Funktionalität in verschiedenen Formen anbieten. Gleichzeitig sollen die einzelnen Anwendungskomponenten aber einheitlich und durchgängig präsentiert werden und bearbeitbar sein. Deshalb sollte ein Anwendungsframework die folgenden Aspekte kapseln und damit austauschbar und leicht veränderbar machen:

- Die *Handhabung & Präsentation* eines Systems ist abhängig von der jeweiligen Arbeitssituation der Anwender und ihrer Vertrautheit im Umgang mit Software. Die Erledigung von Aufgaben muß einfach, schnell und verständlich unterstützt werden. Fragen die hier gestellt werden sind: Gibt es unterschiedliche Benutzergruppen für die Anwendungssoftware? Müssen anwendungsfachliche Vorgänge sowohl hoch standardisiert als auch flexibel durchgeführt werden? Wie präsentieren sich die Arbeitsgegenstände und wie können sie miteinander kombiniert werden?

Schließlich müssen wir noch die Abhängigkeit der Anwendungssoftware von der jeweils eingesetzten Technologie betrachten. Einerseits muß ein Framework mit der bestehenden Systembasis zurechtkommen. Zum anderen müssen neuen Basistechnologien und Middleware ohne Bruch der Gesamtarchitektur eingebunden werden können. Das bedeutet:

- Die *eingesetzte Technologie* muß die unterschiedlichen Arten der Verwendung der Software unterstützen. Der fachliche Kern der Software und ihre Handhabung müssen so gekapselt sein, daß verschiedene Technologien eingesetzt und ohne Seiteneffekte ausgetauscht werden können. Fragen die hier gestellt werden sind: Welche Basissysteme müssen angebunden werden? Wird die Software nur intern am Arbeitsplatz verwendet oder müssen andere Verwendungsarten (WAN, mobiler Einsatz, WWW) in Betracht gezogen werden? Welche Skalierbarkeit für das Anwendungssystem muß eingeplant werden?

## Eine Schichtenmodell für Frameworks

Wir können die oben genannten Dimensionen berücksichtigen, wenn wir für Anwendungssoftware mit Frameworks eine *Schichtenarchitektur* vorgeben. Die Schichtenarchitektur übernimmt für große Softwaresysteme die Rolle des Organigramms für Unternehmen: Für die relevanten Anliegen werden Einheiten mit konkreten Zuständigkeiten und geregelter Kommunikation gebildet.

Eine Schicht organisiert die softwaretechnischen Komponenten zu einer fachlich und technisch motivierten Entwurfs- und Konstruktionseinheit. Die verschiedenen Schichten sind hierarchisch aufgebaut. Als Komponenten innerhalb einer Schicht verwenden wir Klassenbibliotheken und *Frameworks*. In diesem Sinne können Frameworks andere Frameworks enthalten. Die Komponenten sind durch Konstruktionsmuster, Benutzt-Beziehung oder Vererbung realisiert (nach [Zül98]).

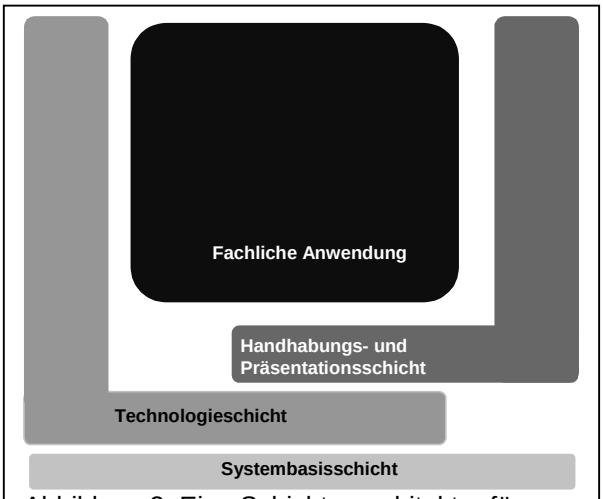

Abbildung 3: Eine Schichtenarchitektur für Anwendungssoftware

Abbildung 3 zeigt beispielhaft die Schichtenarchitektur des JWAM-Frameworks. In jeder Schicht sind Rahmenwerke für spezifische Zwecke enthalten:

- Die Schicht *Fachliche Anwendung* umfaßt Frameworks und Klassenbibliotheken des jeweils modellierten Anwendungsbereichs.
- Die Schicht *Handhabung und Präsentation* umfaßt Frameworks, die den *wiederverwendbaren Anteil* der Konstruktion interaktiver Anwendungssoftware implementieren. So stellen wir etwa die Implementation des Kontrollflusses innerhalb eines Werkzeugs nach dem Beobachter-Muster (vgl. [Gam96]) zur Verfügung.
- Die Systembasisschicht umfaßt Frameworks, die Entwurfsentscheidungen über die verwendete Basistechnologie kapseln. Dazu gehören z.B. Frameworks zur Abstraktion von einer konkreten Middleware.
- Die Technologieschicht umfaßt Frameworks, welche die verwendeten Technologien (siehe Systembasisschicht) so gekapselt zur Verfügung stellen, wie wir sie gerne in den weiter oben liegenden Schichten verwenden möchten.

Eine derartige Schichtenarchitektur ist beispielhaft, aber nicht zwingend. Andere Anwendungsframeworks, etwa IBMs San Francisco, verwenden eine andere. Unabhängig davon, welche konkreten Schichten mit einem

Anwendungsframework verbunden sind, ist jedoch das Ziel: die verschiedenen Dimensionen der Entwicklung von Anwendungssoftware sollen weitgehend unabhängig voneinander bleiben.

Wir meinen, daß die kundenorientierten und anwendungsfachlichen Anliegen auch bei der Entwicklung und dem Einsatz von Anwendungsframeworks im Vordergrund stehen sollten:

- Die fachlichen Forderungen der Kunden und Anwender sollen durch Spezialisierung oder durch zusätzliche Komponenten eines Framework realisiert werden.
- Die Forderungen des Anwendermanagements nach schnellen Entwicklungszyklen können durch die entsprechend vorfabrizierten Frameworkteile bei der Anwendungsentwicklung umgesetzt werden.
- Die Forderungen der Entwickler und des IT-Managments werden durch die Schichtenarchitektur realisierbar.

## Statt eines Resumés

Der Einsatz und die Entwicklung von Frameworks wirft viele Fragen auf. Einige zentrale Probleme haben wir in diesem Artikel angesprochen. Zum Abschluß fassen wir unsere Antworten auf weitere Fragen thesenartig zusammen:

- **Frameworkentwicklung ist ein Profigeschäft**

  Nur wer ausreichendes Fachwissen und eine große Portion softwaretechnische Erfahrung und Kenntnisse im Entwicklerteam hat, sollte sich mit der Neuentwicklung eines Framework befassen. Auch dann ist es ein längerfristiger und kostspieliger Prozeß, der sich erst mit der Zeit auszahlt. Die versprochenen Vorteile (Kosten- und Zeitersparnis) werden nur bei hoher fachlicher und technischer Qualität eingelöst.

- **Frameworks vergegenständlichen Anwendungswissen**

  Ein Framework kann niemals unabhängig von einem bestimmten Anwendungskontext entwickelt werden. (Auch) deswegen kann man heute keine anwendungsfachlichen Frameworks kaufen. Dort liegt aber gerade die Chance für Eigenentwicklungen in Anwenderorganisationen. Sie können ihr fachliches Wissen in zugeschnittene Frameworks hineinkonstruieren. Das Wissen über die technische Konstruktion von Anwendungen ist vergleichsweise groß. Deshalb kann man Behälterklassen oder GUI-Rahmenwerke kaufen. Generische fachliche Frameworks (wie San Francisco) versuchen den Brückenschlag. Dadurch werden sie aber zu unspezifisch und unhandlich. Der Aufwand zu ihrer Spezialisierung ist hoch; Änderungen an der Frameworkstruktur schlagen dann massiv durch.

- **Frameworks bilden die Basis für Produkt- und Projektfamilien**

  Der Einsatz von Frameworks erfordert qualifizierte Entwickler, die sich in die teils sehr komplexen Anwendungsframeworks einarbeiten müssen. Erst mit der Entwicklung einer Reihe von ähnlichen Produkten oder für vergleichbare Projektfamilien wird sich dieser Aufwand spürbar auf der Kostenseite bezahlt machen. Trotzdem sollte nicht übersehen werden, daß gute Frameworks schon qualitativ hochwertige Entwürfe und Softwarekomponenten bereitstellen.

- **Frameworks setzen einen Rahmen für die Anwendungsentwicklung**

  Jedes Framework hat seinen Anwendungsschwerpunkt. Nur Anwendungen die in diesen Rahmen passen können leicht entwickelt werden. Für alle anderen Anwendungen gilt: Der Aufwand für die Änderungen kann schnell größer als bei einer Neuentwicklung werden.

- **Frameworks verändern den Entwicklungsprozeß**

  Wer mit Frameworks Anwendungssoftware entwickelt, muß die Entwicklung des Frameworks zusätzlich im Auge behalten. Denn Frameworks müssen sich weiterentwickeln. Damit stellt sich die doppelte Herausforderung, nicht nur die Anwendung mit den wechselnden Anforderungen im Einsatzkontext, sondern auch mit den Frameworkversionen zu synchronisieren.

- **Frameworkentwicklung und Anwendungsentwicklung sind eng miteinander verzahnt**

  Bei der InHouse-Entwicklung muß ständig überprüft werden, welche Anteile der Anwendungssoftware zum Zweck der Wiederverwendung durch andere Projekte in das Framework wandern sollen. Umgekehrt muß (durch die Architekturgruppe) sichergestellt werden, daß die Dienste des Framework auch in den Anwendungen genutzt werden und daß nicht am Framework „vorbeientwickelt“ wird.

Literatur: [Zül98], [Gam96]