Unterstützung anwendungsorientierter Softwareentwicklung durch das IBM SanFrancisco Rahmenwerk

STUDIENARBEIT

Anthony Norman Schmude
Matrikel-Nummer: 467 63 99

Fachbereich Informatik
UNIVERSITÄT HAMBURG

Betreut von

Dr. Ralf Klischewski

Arbeitsbereich Softwaretechnik
Fachbereich Informatik
UNIVERSITÄT HAMBURG

6. Dezember 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Herstellung komplexer Softwarelösungen ist teuer und aufwendig. Insbesondere trifft dies für die Entwicklung anwendungsorientierter Software zu, sollen qualitativ hochwertige Anwendungen entstehen, die gleichzeitig im höchsten Maße anpassbar und integrationsfähig sind. Im Gegensatz zu Individualsoftware, verspricht die Einführung von Standardsoftware kalkulierbarer und günstiger zu sein. Zudem profitiert der Einkäufer vom *Know-how* des Softwareherstellers. Die Einführung von Standardsoftware bedeutet aber die Anpassung der Geschäftsabläufe an die Rahmenbedingungen der Software, Stichwort: *Business Process Reengineering*. Nur eine Individualsoftware kann bestehende oder neue Geschäftsabläufe eines Betriebes optimal umsetzen und unterstützen. Der Einsatz der „passenden" Geschäfts-anwendungen in großen Firmen ist eine Strategieentscheidung und bietet die Chance, Marktvorteile gegenüber Konkurrenten zu generieren. Eine Fehlentscheidung führt unter Umständen zu dem Gegenteil. Wofür sollen sich IT Verantwortliche also entscheiden ? „*Buy or Build*" ?

IBM möchte mit SanFrancisco den gerade beschriebenen klassischen Entscheidungspolen eine dritte Option hinzufügen. IBM SanFrancisco ist eine Software zur Entwicklung von Geschäftsanwendungen, die bereits einen Teil der Geschäftslogik enthält. Der Aufwand zur Erstellung einer Geschäftsanwendung soll deutlich geringer sein als eine Neuentwicklung, verspricht IBM. Dabei soll eine SanFrancisco basierte Anwendung aber ebenso flexibel wie eine Individualsoftware sein. Diese Arbeit wird diesen beiden Aussagen nachgehen.

1.1 Ziel der Arbeit

Diese Arbeit ordnet IBM SanFrancisco in das Spannungsfeld zwischen Standardsoftware und Individualsoftware ein. Es wird untersucht, ob SanFrancisco die Vorteile beider Ansätze in sich vereinen kann, ohne die jeweiligen Nachteile zu stark aufkommen zu lassen. Was aber bedeutet „Vorteile vereinen" ? Die Entwicklung einer Geschäftsanwendung basierend auf SanFrancisco muss weniger komplex als eine Neuentwicklung sein. Dabei muss die Anpassbarkeit der SanFrancisco Bausteine die gängiger Standardsoftware deutlich übersteigen. Daraus ergibt sich die Fragestellung dieser Arbeit.

> **Fragestellung**: In welchem Maße wird die Entwicklung komplexer Geschäftsanwendungen durch IBM SanFrancisco vereinfacht und inwieweit gleicht die Flexibilität einer IBM SanFrancisco basierten Anwendung der einer Individualsoftware ?

1.2 Vorgehen

Zunächst werden Kriterien erarbeitet, anhand derer IBM SanFrancisco im Sinne der Fragestellung bewertet werden kann. Die Begriffe Softwareentwicklung und Geschäftsanwendung werden im ersten Teil untersucht und eingeschränkt. Dazu gehört die Betrachtung des Umfelds dieser Begriffe. Es wird beschrieben, was einen qualitativ hochwertigen Softwareprozess ausmacht und wie eine Vereinfachung der Softwareentwicklung gemessen werden kann. Ebenso wird der zentrale Begriff „komplexe Geschäftsanwendung" untersucht und daraus abgeleitet, was in diesem Zusammenhang Flexibilität bedeutet.

IBM SanFrancisco wird anschließend anhand folgender vier Aspekte vorgestellt:

- **Ausführungsumgebung**: Die Ausführungsumgebung einer SanFrancisco basierten Anwendung wird analysiert. Es wird der Frage nachgegangen, ob SanFrancisco die Anforderungen einer komplexen Geschäftsanwendung erfüllt.
- **Softwarearchitektur**: Die Softwarearchitektur von SanFrancisco wird vorgestellt und untersucht. Anhand der Softwarearchitektur lassen sich Aussagen zur Flexibilität und Anpassbarkeit ableiten. Außerdem ergibt sich dadurch ein Eindruck, wie kompliziert die Softwareentwicklung mit SanFrancisco ist.
- **Geschäftslogik**: Die mitgelieferte Geschäftslogik von SanFrancisco wird beispielhaft vorgestellt. Von Interesse ist dabei, ob die Geschäftslogik verwendet oder durch einfache Anpassungen genutzt werden kann.
- **Softwareentwicklungsprozess**: Der von IBM vorgeschlagene Softwareprozess zur Entwicklung SanFrancisco basierter Anwendungen wird beschrieben. Hierdurch ergeben sich Abschätzungen über die Einfachheit der Softwareentwicklung, aber ebenso zur Flexibilität der Software.

Diese Arbeit trägt Informationen zu SanFrancisco aus allen frei verfügbaren Quellen zusammen. Alle Angaben und Aussagen beziehen sich auf die Version 1.3 von IBM SanFrancisco. Da IBM nicht in alle Details Einblick gewährt, bleiben einige interessante Fragen unbeantwortet, zum Beispiel zur Objektrationalen Abbildung der Objekte. Aufgrund der Komplexität von SanFrancisco wird die Benutzungsschnittstelle und die Performanz nicht untersucht.

Abschließend werden die betrachteten Aspekte von SanFrancisco im Kontext der eingangs erarbeiteten Kriterien diskutiert und bewertet. Dadurch kann die Fragestellung fundiert beantwortet werden.

1.3 Aufbau der Arbeit

Das anschließende Kapitel 2 erläutert die zentralen Begriffe und Sichtweisen dieser Arbeit. Drei Bereiche sind von Interesse: Anforderungen an komplexe Geschäftsanwendungen, Softwarequalität und Möglichkeiten zur Vereinfachung der Softwareentwicklung. Die Anforderungen und Kriterien für hohe Softwarequalität werden als Bewertungskatalog zusammengeführt.

Das Kapitel 3 stellt die Hintergründe der IBM zur Entwicklung von SanFrancisco vor. Großen Raum nehmen die Ziele ein, die sich IBM gesetzt hat. Davon ausgehend, dass IBM die gesetzten Ziele umgesetzt hat, ergibt sich eine gute Vorstellung der Leistungsfähigkeit von SanFrancisco.

Die Ausführungsumgebung von SanFrancisco ist Thema des Kapitel 4. Es werden die Komponenten der Ausführungsumgebung und deren Zusammenwirken beschrieben. Außerdem wird detailliert vorgestellt, wie SanFrancisco Daten persistent speichert. Das Kapitel schließt mit einer kurzen Übersicht der Administrationswerkzeuge.

Das Kapitel 5 widmet sich der Softwarearchitektur von SanFrancisco. Die Schichtung der Software und die jeweilige Aufgabe dieser Schichten wird erläutert. Das Hauptaugenmerk richtet sich dabei auf die Basiskonzepte von SanFrancisco.

Die mitgelieferte Geschäftslogik von SanFrancisco ist Thema des Kapitel 6. Es werden die Funktionalitäten der einzelnen Anwendungsdomänen beschrieben, die SanFrancisco unterstützt.

Das Kapitel 7 beschreibt den Softwareentwicklungsprozess, wie ihn IBM vorschlägt, um SanFrancisco basierte Applikationen zu entwickeln. Es werden die einzelnen Aufgaben des Prozesses erläutert. Dieses Kapitel stellt zusätzlich die Werkzeugstrategie von IBM am Beispiel des *SFBuilders* vor.

Das Kapitel 8 beantwortet die Fragestellung dieser Arbeit indem SanFrancisco anhand der Kriterien des Bewertungskatalogs aus Kapitel 2 bewertet wird. Die erarbeiteten Fakten aus den Kapiteln 3 bis 7 werden dazu herangezogen. Das Kapitel geht weiterhin auf die Praxisrelevanz von SanFrancisco ein.

Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 9.

1.4 Formalien

Diese Arbeit versucht aus Gründen der Lesbarkeit die vielen englischen Fachworte und Begriffe zu vermeiden. Englische Begriffe werden durch adäquate deutsche ersetzt. Sollte das nicht möglich sein, werden *englische Begriffe kursiv* dargestellt. Alle englischen IBM Termini werden erwähnt, dann aber durch entsprechende deutsche ersetzt.

Programmkode wie auch Begriffe aus Programmierzusammenhängen werden durch eine der `Schreibmaschine` ähnlichen Schrift kenntlich gemacht.

> **Begriffsbestimmungen** werden eingerückt und durch einen Querbalken am linken Rand gekennzeichnet. Der zu erläuternde Begriff ist am Anfang fett ausgewiesen.

2 Anwendungsorientierte Softwareentwicklung

Ziel dieser Arbeit ist, IBM SanFrancisco zwischen Standard- und Individualsoftware einzuordnen. SanFrancisco ist keine fertige Geschäftsanwendung sondern eine Art Fundament zu ihrer Entwicklung. Ohne Softwareentwicklung ist SanFrancisco also nicht einsetzbar. Dieses Kapitel erarbeitet dazu den Hintergrund. Es wird der Begriff Geschäftsanwendung erläutert und Qualitätskriterien von Software und deren Entwicklung beschrieben. Konkret erarbeitet dieses Kapitel Kriterien, die der Beantwortung der Fragestellung dienen, also solche zur Bewertungen der Flexibilität einer Geschäftsanwendung und solche zur Vereinfachung der Softwareentwicklung.

Dieses Kapitel führt zunächst in das Umfeld des Gegenstandsbereichs ein. Im zweiten Teil werden wichtige Anforderungen an eine Geschäftsanwendung dargestellt, beginnend damit, was Flexibilität in diesem Zusammenhang bedeutet. Der dritte Teil geht auf die Entwicklung von Software ein. Dort wird der Entwurf von Software mit dem Ziel betrachtet, Merkmale hoher Softwarequalität zu erarbeiten. Nicht nur die Entwicklung sondern vielmehr der Entwicklungsprozess ist entscheidend für die Qualität einer Software. Der vierte Teil dieses Kapitels beschäftigt sich daher mit Merkmalen eines guten Softwareprozesses. Der fünfte Teil benennt Kriterien, die die Entwicklung von Geschäftsanwendungen vereinfachen. Am Ende des Kapitels, im Teil sechs, werden die Ergebnisse in Form eines Bewertungskatalogs zusammengefasst.

2.1 Klärung zentraler Begriffe

Software wird in verschiedene Klassen eingeteilt. Es wird Systemsoftware und Anwendungssoftware unterschieden. Systemsoftware bildet mit der Hardware die Technik- und Kommunikationsinfrastruktur späterer Anwendungssysteme. Anwendungssoftware dagegen orientiert sich an betrieblichen Aufgaben (vgl. [StHa99]). Die Orientierung an betrieblichen Aufgaben bedeutet zum einen die Anlehnung an organisatorische Gegebenheiten, aber natürlich auch die Einbeziehung der Benutzer. Anwendungssoftware ist kein Selbstzweck und führt Aufgaben nicht selbstständig durch. Automatisierte Funktionen oder Prozesse gehen immer von Benutzerinteraktionen aus. Zusammenfassend ergibt sich:

> **Anwendungssoftware**: „Anwendungssoftware ist ein Mittel zum Zweck, um fachliche Aufgaben in einem oder mehreren Anwendungsbereichen zu erledigen.(...) Anwendungssoftware dient der Steuerung von technischen Prozessen oder der Unterstützung von Arbeitsprozessen. Bei ihrer Entwicklung ist auch die Gestaltung der Interaktion zwischen den Anwendungsfachleuten und dem eingesetzten Anwendungssystem zu beachten.“ [FlZü97, S. 642]

Anwendungssoftware wiederum wird in Standard- und Individualsoftware unterteilt. Die Unterscheidung bezieht sich nicht auf den Funktionsumfang oder mögliche Anwendungsbereiche. Standardsoftware wird für einen anonymen Markt, Individual-

software für genau einen Auftraggeber entwickelt (siehe [Ro98]). Individualsoftware kann dadurch stärker an betriebliche und organisatorische Gegebenheiten des Auftraggebers angepasst werden. Heute werden immer noch viele Softwareprojekte durch Entwicklung von Individualsoftware umgesetzt. Das zeigt, wie wichtig und unterschiedlich die Unternehmensorganisationen und betriebliche Abläufe sind.

Der Fokus dieser Arbeit richtet sich nicht auf Anwendungssoftware im allgemeinen, sondern auf Software, die Geschäftsvorfälle eines Unternehmens direkt unterstützt. Es geht also nicht um Textverarbeitungssoftware sondern um Buchhaltungssoftware, Lagerhaltungssoftware usw.

Damit eine Software diese Unterstützung leisten kann, muss sie Unternehmensdaten bereitstellen, anlegen und verändern können. In objektorientierten Softwaresystemen werden solche Unternehmensdaten durch Geschäftsobjekte repräsentiert.

> **Geschäftsobjekt:** „Ein Geschäftsobjekt repräsentiert einen Gegenstand, ein Konzept, einen Ort oder eine Person aus dem realen Geschäftsleben innerhalb eines Softwaresystems (zum Beispiel: Vertrag, Rechnung)." [Oest97]

Der für diese Arbeit zentrale Begriff Geschäftsanwendung findet sich in der deutschsprachigen Wissenschaftsliteratur kaum, eher in Marketingunterlagen. Die Fachliteratur spricht von betrieblicher Anwendungssoftware. Man unterscheidet diese je nach unterstützter Anwendungsdomäne. Betriebliche Anwendungssoftware kann unter anderem die Materialwirtschaft, die Buchhaltung, die Produktion, das Management oder beliebige Kombinationen unterstützen. Die Gesamtheit betrieblicher Anwendungssoftware in einen Unternehmen ergibt das betriebliche Informationssystem. Der Hauptaugenmerk richtet sich auf die Verarbeitung und Speicherung von Informationen, weniger auf die direkte Steuerung von Anlagen (vgl. [StHa99]). Diese Arbeit verwendet den Begriff Geschäftsanwendung im obigen Sinne und meint damit konkret:

> **Geschäftsanwendung:** Eine Geschäftsanwendung ist Anwendungssoftware zur Unterstützung von Geschäftsvorfällen in Unternehmen. Funktionen und Verrichtungen werden an Geschäftsobjekten ausgeführt.

Offen bleibt absichtlich, wie diese Unterstützung konkret aussehen soll. Die Geschäftsanwendung könnte sich an ein Management Informationssystem, ein sozio-technisches System, eher an einer funktionalen oder prozessorientierten Unternehmensorganisation anlehnen. Für die Umsetzung bedeutet das, entweder das ablaufgesteuerte Unternehmen oder den qualifizierten Mitarbeiter bzw. Prozesse oder Objekte als Hauptgegenstand der Gestaltung zu betrachten. Welche übergreifenden Anforderungen große Unternehmen an eine Geschäftsanwendung stellen, ist Thema des folgenden Abschnitts.

2.2 Anforderungen an eine Geschäftsanwendung

Für den erfolgreichen Einsatz von Geschäftsanwendungen in großen Unternehmen, müssen viele Anforderungen erfüllt werden. Natürlich muss eine Geschäftsanwendung zu aller erst die jeweiligen Aufgaben, Prozesse und Unternehmensorganisation optimal unterstützen. Darüber hinaus gibt es aber viele weitere Kriterien, die den Einsatz zu

einem Erflog oder Misserfolg werden lassen. Große Unternehmen sind oft weltweit tätig und verfügen über mehrere Firmenniederlassungen mit verschiedenen Aufgabengebieten. Die Unternehmensleitung möchte zeitnah und vollständig über alle Daten aller Firmenteile informiert sein. Die einzelnen Niederlassungen wiederum müssen optimal miteinander kooperieren, was insbesondere für die eingesetzte Software gilt. Zusätzlich zu diesen Forderungen muss sich das Informationssystem ändernden Firmenstrukturen und sich ständig ändernden Geschäftsprozessen anpassen können. Nicht zuletzt hängt der erfolgreiche Einsatz einer Geschäftsanwendung entscheidend von der Akzeptanz der Benutzer ab.

Anhand dieser allgemeinen Überlegungen lassen sich konkrete Forderungen an eine Geschäftsanwendung ableiten, die nachfolgend in Form von Kriterien zusammengestellt sind. Ein Hauptaspekt dieser Arbeit ist die Flexibilität einer Geschäftsanwendung, die zuerst erläutert wird. Die nachfolgenden Kriterien tauchen in unterschiedlicher Literatur ([Han96], [StHa99], [Ro98] u.a.) immer wieder als wichtige Punkte auf, wobei meist jede Auflistung nicht vollständig ist. Vielmehr geht es darum, die für die Beantwortung der Fragestellung relevanten Anforderungen zu beschreiben, aber auch diejenigen, die schwer umzusetzen sind.

2.2.1 Flexibilität

Flexibilität einer Software bedeutet, dass sie in unterschiedliche Unternehmens-organisationen eingepasst werden kann. Unternehmen werden eher funktional oder prozessorientiert organisiert. Eine Geschäftsanwendung ist entweder an der Ablaufsteuerung von Aufgaben und Prozessen orientiert oder stellt Materialien bzw. Objekte, an denen Aufgaben verrichtet werden, in den Mittelpunkt. Die Mitarbeiter sind entweder beplante Ausführungseinheiten oder situativ selbst verantwortlich für Unternehmensprozesse (vgl. [Ro98]).

Eine Geschäftsanwendung kann Teil eines Management oder eines sozio-technischen Informationssystems eines Unternehmens sein. Je nachdem, für welche Sichtweise sich entschieden wird, sind Mitarbeiter Teil des Systems oder nur „Datenlieferant". Ein sozio-technisches System beteiligt die Mitarbeiter an ihrer Arbeitsorganisation (vgl. [Ro98]).

Diese zwei Anforderungsgebiete, Unternehmensorganisation und Art des Informationssystems, sind oft für den Misserfolg des Einsatzes einer Geschäftsanwendung verantwortlich (vgl. [Ro98]). Bei der Entwicklung werden implizit Annahmen über die Organisation aber auch über die Einbettung in das Informationssystem getroffen, die im Nachhinein schwierig zu ändern sind.

Flexibilität einer Software bedeutet aber auch, die Software einfach an neue Anforderungen anpassen zu können. Einfach heißt in Relation zu anderer Software. Ist ein komplettes *Redesign* der Software notwendig oder genügt es ein paar Einstellungen zu verändern, um neue Anforderungen umzusetzen ? Zum einen hängt das von den Anforderungen ab, aber natürlich auch von der Geschäftsanwendung.

Um ein flexibles System zu erhalten, ist eine gutes, lose gekoppeltes und modulares Softwaredesign notwendig, die mögliche zukünftige Veränderung berücksichtigt. Werden Daten, Benutzungsschnittstelle und Prozessabläufe gekapselt, kann jede dieser

Komponenten verändert werden, ohne das dies unmittelbar Auswirkungen auf die anderen Teile hat.

2.2.2 Offenheit

„Offene" Software basiert auf herstellerunabhängigen Standards. Ziel ist es, dem Unternehmen, das eine Anwendungssoftware einsetzt, möglichst viele Freiheitsgrade für zukünftige Entscheidungen zu lassen (siehe [Han96]).

Offene Softwaresysteme lassen sich entscheidend einfacher austauschen oder in bestehende Informationssysteme integrieren, die auf den gleichen Standards basieren. Eine Kommunikation solcher Systeme kann entscheidend vereinfacht werden. Diese Offenheit gilt insbesondere für die Datenhaltung von Anwendungssoftware. Die Daten müssen ohne großen Aufwand entnommen und weiterverarbeitet werden können, insbesondere dann, wenn die Anwendung nicht mehr eingesetzt werden soll. Daneben muss ein offenes Softwaresystem auch auf zukünftige Rechnerarchitekturen übertragbar sein.

2.2.3 Integrationsfähigkeit

Ein Softwaresystem ist integriert, wenn

- die Prozesse des Systems umfassend aufeinander abgestimmt sind,
- die Verbindungen zwischen verschiedener Anwendungssoftware weitgehend automatisiert ist, also ohne menschliche Eingriffe stattfindet,
- die Daten frühzeitig erfasst werden und für jede Software zentral im Zugriff sind (siehe [Han96]).

Durch die Umsetzung dieser Forderungen sind Zeit- und dadurch Kosteneinsparungen durch den Wegfall von Reibungsverlusten zu erwartet. Heute ist neben der Anwendungsintegration innerhalb eines Unternehmens die Integration der Geschäftsprozesse mit externen Partnern von Interesse.

Prozesse innerhalb einer Anwendung sind meist einfach zu integrieren. Ein „gutes", modulares Softwaredesign ist dabei hilfreich. Eine Trennung in Datenoperationen und Prozesssteuerung ist sinnvoll. So lassen sich die Daten in verschiedenen Prozessen unterschiedlich nutzen.

Eine Software muss „offen" sein, damit sie mit anderer Software einfach integriert werden kann. Setzen verschiedene Anwendungen auf gleiche Standards, ist eine Integration einfacher. Müssen komplizierte Filter- und Konvertierungsmaßnahmen durchgeführt werden, die womöglich noch fehleranfällig sind, ist eine reibungslose Integration schwierig.

2.2.4 Skalierbarkeit

Eine skalierbare Anwendung ist so konzipiert, dass sie bei steigenden Anforderungen gleichmäßig mitwachsen kann. Wenn eine Geschäftsanwendung auf einer bestimmten Hardware eine bestimmte Anzahl Transaktionen pro Zeiteinheit verarbeiten kann, sollte die Anwendung auf „doppelter" Hardware ungefähr die doppelte Menge an Transaktionen verarbeiten können.

Eine Geschäftsanwendung muss skalierbar sein, da Unternehmensgrößen sich verändern. Die Anwendung muss analog mitwachsen können. Die Kosten für das Informationssystem dürfen nicht explodieren. Das heißt, die Hardwareanforderungen pro Transaktion müssen annähernd linear mit der Anzahl der Transaktionen wachsen.

Eine verteilte Systemarchitektur fördert die Skalierbarkeit, da Transaktionsschwerpunkte auf verschiedene Rechner verteilt werden können. Reicht die Rechenleistung nicht mehr aus, kann ein weiterer Knoten hinzugefügt werden.

2.2.5 Ausfallsicherheit

Ein Softwaresystem ist ausfallsicher, wenn die erwartete Erreichbarkeit des Systems annähernd 100% ist.

In großen Unternehmen ist der Ausfall des Informationssystems oft eine Katastrophe, da es alle wichtigen Geschäftsprozesse unterstützt und teilweise erst möglich macht. Ein Ausfall bedeutet großen finanziellen Verlust.

Um Ausfälle zu vermeiden, sind hauptsächlich Hardwaremaßnahmen erforderlich. Diese Arbeit untersucht aber softwaretechnischen Maßnahmen. Wenn ein System eine zentrale Stelle zur Verwaltung benötigt, ist mit dem Ausfall dieser Stelle das gesamte System betroffen. Ist ein System verteilt, bedeutet der Ausfall einer Komponente nur den Ausfall der angebotenen Dienste, nicht aber den Ausfall des gesamten Systems.

2.2.6 Mandantenfähigkeit

Mandantenfähigkeit bedeutet die unabhängige Verwaltung mehrere getrennter Unternehmen in einer Installation eines Softwaresystems. Eine Geschäftsanwendung für große Unternehmen muss mandantenfähig sein, da die Unternehmen sich oft in kleinere, rechtlich eigenständige Unternehmen gliedern.

Diese Trennung betrifft nicht nur die Datenhaltung, sondern kann auch unterschiedliche Prozessabläufe für gleiche Geschäftsvorfälle bedeuten. Eine vollständig mandantenfähige Software bietet die Möglichkeit, dass gleiche Geschäftsobjekte sich in unterschiedlichen Mandanten unterschiedlich verhalten können.

2.2.7 Rechtemanagement

Geschäftsvorfälle werden üblicherweise abteilungsübergreifend bearbeitet, wobei für jede Abteilung jeweils unterschiedliche Aspekte im Vordergrund stehen. Der Zugriff auf die gleichen Geschäftsobjekte kann verschieden aussehen. So könnte ein Abteilungsleiter zum Beispiel die Stunden seiner Mitarbeiter erfassen, aber nur die Personalabteilung dürfte den zugehörigen Stundensatz einsehen und verändern. Um solche Einschränkungen zu ermöglichen, bedarf es eines geeigneten Rechtemanagements mit der Möglichkeit, den Zugriff auf verschiedene Sichten eines Geschäftsobjekte zu konfigurieren.

Außer diesen Sichten muss es möglich sein, Rechte an der Ausführung von Prozessen zu konfigurieren. Nicht jeder Benutzer darf den Jahresabschluss eines Unternehmens anstoßen oder Rechnungen begleichen.

Ein Rechtemanagement sollte Rollen und Benutzer kennen, um gleichartige Rechte zu Rollen zusammenfassen zu können und diese dann Benutzern zuzuordnen. In großen Unternehmen mit vielen Benutzern ist das unabdingbar.

2.2.8 Internationalisierbarkeit

Ein Softwaresystem ist internationalisierbar, wenn zusätzlich zu verschiedenen Sprachen auch landestypische Unterschiede der angebotenen Prozesse umgesetzt sind. Die zweite Forderung ist die für international tätige Unternehmen bedeutendere. Es geht um die Einhaltung verschiedener staatlicher Gesetzte und Eigenheiten. Soll eine Geschäftsanwendung in großen Unternehmen eingesetzt werden, müssen gleiche Unternehmensprozesse landestypisch abgebildet werden.

2.3 Softwarequalität

Qualitativ hochwertige Software zu entwickeln ist bis heute ein komplexer Vorgang. Viele Softwareprodukte weichen von den gewünschten Anforderungen ab und überschreiten bei der Entwicklung zusätzlich das Kosten- und Zeitbudget (siehe [WeOr92]). Nicht nur die Entwicklung, sondern auch die Ermittlung der Softwarequalität ist schwierig. Es gibt zwar vielfältige Qualitätsmerkmale, viele sind aber kaum zu quantifizieren. Die Informatik ist sich aber sicher, dass ein hochwertiger Softwareprozess notwendig ist, damit überhaupt gute Software entstehen kann. Für die Güte des Softwareprozess gilt aber wiederum, direkte Auswirkungen auf Qualitätsmerkmale sind schwierig zu bewerten.

Dieser Abschnitt benennt zunächst generelle Qualitätskriterien von Software. Diese werden dann anhand von Merkmalen der Softwarearchitektur diskutiert. Der Fokus richtet sich auf objektorientierte Softwareentwicklung, da SanFrancisco objektorientiert ist. Ziel der Überlegungen ist, Softwarearchitekturmerkmale zu identifizieren, die eine gute Softwarequalität ermöglichen.

Bei der Bewertung der Qualität von Software, muss beachtet werden, wer sie bewertet. Ein Benutzer wird andere Forderungen aufstellen als ein Entwickler oder ein Administrator. Aus Anwendersicht sind folgende Merkmale entscheidend. Qualitativ hochwertige Software muss

- aufgabenangemessen,
- selbstbeschreibend,
- erwartungskonform,
- fehlertolerant,
- individualisierbar,
- lernförderlich und
- steuerbar sein (siehe [StHa99]).

Was bedeutet das für eine mögliche Softwarearchitektur ? Wie können diese Forderungen überprüft werden ? Zur Erreichung dieser Ziele werden einige Vorschläge gemacht. So kann eine vorhandene Undo-Funktionalität die Fehlertoleranz erhöhen. Verständliche Fehlermeldungen, die das Umfeld des Fehlers verdeutlichen und Maßnahmen zur Korrektur enthalten, tragen ebenso dazu bei, um nur einige zu nennen.

Zur Bewertung der Qualität einer Software müssen weitere Kriterien herangezogen werden. Gute Softwarequalität zeichnet sich durch

- Korrektheit,
- Zuverlässigkeit,
- Adäquatheit,
- Erlernbarkeit,
- Robustheit,
- Lesbarkeit,
- Erweiterbarkeit,
- Testbarkeit,
- Effizienz und
- Portabilität aus (siehe [PoBl93 S. 14]).

Hier stellt sich ebenfalls die Frage, wie die einzelnen Forderungen auf eine mögliche Softwarearchitektur wirken. Da der Zusammenhang zwischen Forderungen und Architekturmerkmalen vielfältig ist, soll die Frage andersherum beantwortet werden. Nachfolgend werden Modellierungs- und Architekturmöglichkeiten und deren Wirkung auf einzelne Forderungen beschrieben. Im nächsten Abschnitt werden Eigenschaften eines guten Softwareprozesses untersucht.

2.3.1 Objektorientierte Softwareentwicklung

Im Gegensatz zur daten- oder funktionsorientierten Herangehensweise, verbindet die objektorientierte Softwareentwicklung Daten und Operationen zu Verantwortlichkeiten von Objekten. Von Beginn an werden konzeptionelle Fragen der Anwendungsdomäne betrachtet und fokussiert. Ein objektorientierter Ansatz modelliert die Anwendungsdomäne näher an realen Dingen als daten- oder funktionsorientierte Ansätze das können. Die Modelle werden dadurch für Domänenexperten einfacher verständlich (vgl. [Ro98]).

Die objektorientierte Modellierung wirkt positiv auf die Flexibilität einer Software. Es werden von Beginn an Objekte, also Daten und Operationen, und deren Beziehungen modelliert. Die Reihenfolge der Ausführung von Operationen wird möglichst lange aufgeschoben. Dadurch ergibt sehr lange die Möglichkeit, auf Änderungen flexibel, also ohne grundsätzliche Änderung der bisherigen Modellierung, zu reagieren.

Die Erweiterbarkeit objektorientierter Software übersteigt die von daten- oder funktionsorientiert modellierter Software. Funktionen ändern sich wesentlich häufiger als Objekte, mit denen gearbeitet wird (vgl. [Ro98]).

Der objektorientierte Ansatz zur Entwicklung von Anwendungssoftware ist positiv zu bewerten. Dieser Ansatz ist einem daten- oder funktionsorientierten Ansatz in den beschriebenen Punkten überlegen. Es ist aber kein Allheilmittel, um Software zu erstellen, die alle Qualitätsmerkmale positiv beeinflusst. Die Merkmale Korrektheit, Erlernbarkeit, Adäquatheit und Robustheit werden eher durch den Softwareprozess beeinflusst.

2.3.1.1 Entwurfsmuster (*Design Pattern*)

Die objektorientierte Modellierung ermöglicht eine bessere Wiederverwendung von Arbeitsergebnissen. Dieser Aspekt wird insbesondere durch den Einsatz von Entwurfsmustern (*Design Pattern* siehe [GHJV95]) unterstützt. Die in Entwurfsmuster eingegangenen Problemlösungen lassen sich in vielen Systemen umsetzen.

Durch den Einsatz von Entwurfsmustern kann die Komplexität reduziert werden, indem bereits bekannte Konzepte komponiert werden. Das wirkt positiv auf die geforderte Korrektheit, die Erweiterbarkeit und die Testbarkeit. Oft hat die Effizienz darunter zu leiden.

2.3.1.2 Anwendungsrahmenwerke (*Application Frameworks*)

Teile eines Rahmenwerks (*Frameworks* siehe [GHJV95]) müssen konkretisiert werden, um eine lauffähige Applikation zu erhalten. Die Verwendung von Funktionalitäten eines Rahmenwerks ist viel tiefgreifender als die Verwendung von einfachen Funktionen einer Klassenbibliothek. Es werden nicht nur Methoden aufgerufen, sondern es müssen Objekte beerbt und angepasst werden, um die vorhandene Funktionalität des Frameworks an die eigenen Bedürfnisse anzupassen.

Ein Anwendungsrahmenwerk (*Application Framework*) ist ein Rahmenwerk, das bereits den Kontrollfluss einer Anwendungsdomäne enthält. Das Anwendungsrahmenwerk kann durch Einpassen einsatzspezifischer Komponenten mit passenden Schnittstellen des Rahmenwerks zu einer Anwendung erweitert werden. Es muss also nicht der komplette Kontrollfluss entwickelt werden (vgl. [FlZü97]).

Der Einsatz von Rahmenwerken soll ebenfalls die Komplexität reduzieren. Es liefert bereits die Architektur und die Aufteilung der Klassen und Objekte. Dies ermöglicht, den Fokus der Entwicklung auf die domänenspezifischen Probleme zu richten. Dadurch kann die Korrektheit und Adäquatheit positiv beeinflusst werden.

2.3.1.3 Softwarekomponenten

Eine Softwarekomponente bietet anderen Komponenten Dienste an, die über das reine liefern von Daten hinausgeht. Solche Komponenten können zu integrierten Anwendungssystemen zusammengestellt werden. Damit solche Komponenten Informationen austauschen können, ist eine vermittelnde Software notwendig. Beispiele für solche *Middleware* sind Applikationsserver (vgl. [FlZü97]).

Setzt sich eine Anwendung aus mehrere Komponenten zusammen, ergibt sich daraus eine losere Kopplung der Funktionalitäten. Einzelne Komponenten können ausgetauscht oder sogar weggelassen werden.

Der Einsatz von Softwarekomponenten wirkt positiv auf die Robustheit der Anwendung.

2.4 Softwareprozess

Ein hochwertiger Softwareprozess ist wichtig, um qualitativ hochwertige Software entstehen zu lassen. Softwareprozesse beschreiben die Abfolge von Aufgaben und Arbeitsergebnissen zur Entwicklung von Software, aber auch Methoden, um die Aufgaben durchzuführen. In der Literatur finden sich unterschiedliche Begriffe und

Abgrenzungen für ähnliche Aktivitäten. Zusammenfassend gliedert sich die Softwareentwicklung in folgende Aktivitäten: Anforderungsermittlung oder Analyse, Systemdefinition, Entwurf, Implementierung, Validation, Systemeinführung und Wartung. Um aus diesen Aktivitäten einen tragfähigen Softwareprozess zu erhalten, bedarf es des Prozessmanagements und des Qualitätsmanagements (vgl. [FlZü97]).

Die Informatik hat viele verschiedene Vorgehensmodelle zur Entwicklung von Software hervorgebracht und beschrieben. Es können gradlinige (Wasserfallmodel, V-Modell) von zyklischen (Spiralemodell, *Software-Life-Cycle-Modell*, STEPS) Modellen unterschieden werden. Ältere Modelle orientieren sich stark an der Technikkomponente von Software, neuere konzentrieren sich auf den Nutzer und den Einsatzkontext in Organisationen (vgl. [Ro98]). Einige sehen Software als Produkt und deren Entwicklung als ingenieurmäßigen Herstellungsprozess. Andere, wie STEPS (siehe Abbildung 1), lehnen diese Sichtweise von Software ab und stellen die Unterstützung des Arbeitsplatzes in den Vordergrund. Software wird als Arbeitsmittel und Kommunikationsmedium eines sozial und technisch eingebetteten Systems verstanden. Software ist bei STEPS eine Abfolge von Versionen mit Anpassungen an veränderte Arbeitskontexte und soll menschengerecht gestaltet sein (vgl. [FlZü97]).

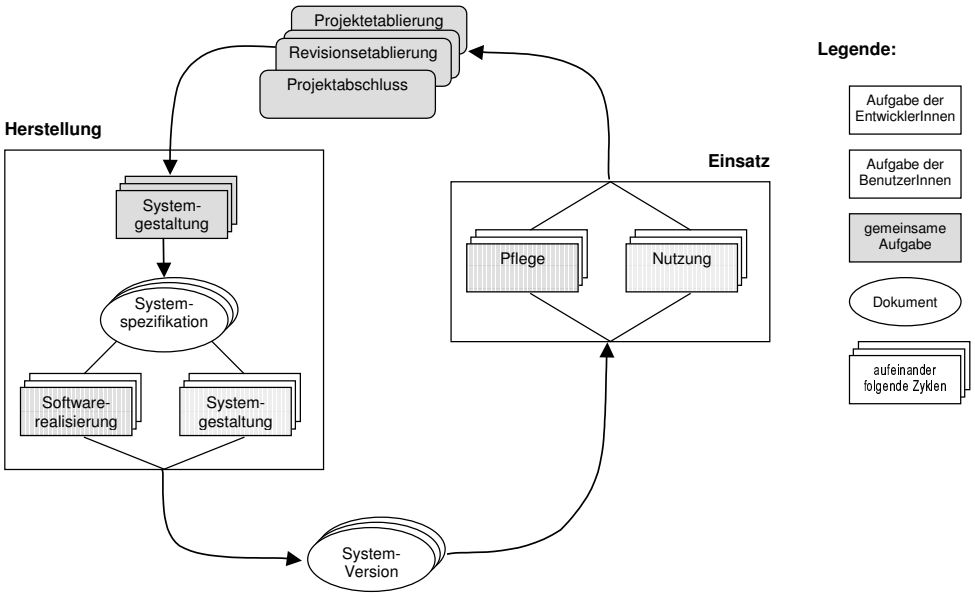

Abbildung 1: Das Vorgehensmodell von STEPS zur Entwicklung von Software

Welches Modell und damit welcher Prozess für die Entwicklung welcher Art von Software der richtige ist, kann nicht eindeutig beantwortet werden. Es ist vielmehr so, dass jede Software entwickelnde Organisation einen für sich geeigneten Prozess an die eigenen Bedürfnisse anpassen sollte (vgl. [Balz98]).

Entscheidend für die Entwicklung von Anwendungssoftware ist, ob und wie spätere Nutzer des Systems in den Softwareprozess eingebunden werden. Zum einen wird dadurch die Akzeptanz eines neuen Systems erhöht, zum anderen aber auch die Adäquatheit. Die Beteiligung ganz unterschiedlicher Personengruppen erfordert von allen Prozessbeteiligten einen Lernprozess, an dessen Ende Software steht (vgl. [FlZü97]). Ebenso wichtig wie die Beteiligung der Anwender ist die Qualitätssicherung.

Diese muss durch den Softwareprozess sichergestellt werden. Qualitativ schlechte Software kann jeglichen geplanten Nutzen wieder zunichte machen.

Heute wird ein evolutionärer Softwareprozess zur Anwendungsentwicklung einem gradlinigen Vorgehen vorgezogen. Vielfach wird auf positive Effekte des *Prototyping* hingewiesen. Ein Anwendungssystem kann so unter Einbeziehung der Benutzer sukzessive entwickelt und eingeführt werden (vgl. [Ro98]).

2.4.1 Qualitätsmanagement

Ebenso wichtig wie Qualitätsmerkmale aufzustellen, ist deren Einhaltung. Dazu können auf verschiedenen Ebenen ganz unterschiedliche Maßnahmen beitragen. Zum einen sollte ein produktbezogenes Qualitätsmanagement eingeführt werden, welches Produkte und Zwischenergebnisse anhand vorher definierter Qualitätsmerkmale prüft. Zum anderen sollte ein prozessorientiertes Qualitätsmanagement durchgeführt werden, das die Qualität des Softwareprozess an sich sicher stellt (vgl. [Balz98]).

Alle am Softwareprozess beteiligten Personen sollten ein Qualitätsbewusstsein entwickeln, um so die Qualität des Prozess und dadurch der Software zu verbessern. Qualitätssicherungsmaßnahmen können analytisch oder konstruktiv sein. Beide sind gleich wichtig für die Softwarequalität. Klassische analytische Verfahren wie das Testen von Software alleine reichen nicht aus [vgl. [Balz98]). Früher konzentrierte sich die Qualitätssicherung auf konstruktive Maßnahmen zur Verbesserung der Softwarequalität. Die Erfahrung hat aber gezeigt, dass die Softwarequalität stärker durch die Qualität des Softwareprozess beeinflusst wird. Konsequent weitergeführt, wurden deshalb verschiedene Ansätze entwickelt, um die Prozessqualität zu steigern. Dazu gehören

- der ISO 9000 Ansatz,
- der TQM Ansatz (*Total Quality Management*),
- der CMM Ansatz (*Capability Maturity Model*),
- der SPICE Ansatz und
- *Business Engineering* (vgl. [Balz98]).

Diese Verfahren im einzelnen vorzustellen, übersteigt den Rahmen dieser Arbeit.

Das Qualitätsmanagement gliedert sich in Qualitätsplanung, Qualitätslenkung und -sicherung und Qualitätsprüfung. Aufgabe der Qualitätsplanung ist, Qualitätsanforderungen an den Prozess und die Software festzulegen. Die Qualitätslenkung und –sicherung steuert und überwacht den Entwicklungsprozess mit dem Ziel, die Qualitätsanforderungen zu erfüllen. Die Qualitätsprüfung überprüft die Qualitätsanforderungen anhand messbarer Merkmale.

Testen der Software und einzelner Komponenten ist nach wie vor eine Methode der Qualitätsprüfung, also ein Teil des Qualitätsmanagements. Softwaretest verfolgen unterschiedliche Ziele. Zum einen den Abgleich mit den Anforderungen, zum anderen eine Prüfung der Qualität. Aktuell hoffen viele auf sogenannte Unit-Tests die dem *Extreme Programming*[*] entstammen und Zusicherungen automatisch prüfen. Diese

[*] Das *extreme programming (XP)* ist eine 1996 von Kent Beck beschriebene neue Art der Softwareimplementierung. Neu ist die Paarprogrammierung, was bedeutet, dass zwei

Tests können bereits in der Entwicklung durchgeführt werden und setzen somit am frühest möglichen Zeitpunkt an. Man hofft so, ungewollte Seiteneffekte von Kodeänderungen verhindern zu können.

2.5 Vereinfachung der Softwareentwicklung

Dieser Abschnitt geht der Frage nach, was die Entwicklung von Software, insbesondere einer Geschäftsanwendung, vereinfachen kann. Vereinfachen meint zweierlei: Zum einen die Beschleunigung der Entwicklung, zum anderen die Einschränkung der Komplexität. Der gesamte Softwareprozess ist in diesem Sinne durch Software nicht zu vereinfachen, da es sich um einen kreativen und kommunikativen Prozess handelt, an dem viele Menschen mit unterschiedlichen Erfahrungen beteiligt sind. Alle Beteiligten durchlaufen einen Lernprozess zur Entwicklung von Software (siehe [FlZü97]). Software kann den Arbeitskontext aber besser oder schlechter unterstützen.

Das Design und die Kodierung einer Software kann durch geeignete Software beschleunigt und vereinfacht werden. Naiv betrachtet möchte man meinen, es genügt, benötigte Funktionalitäten zu liefern. Darüber hinaus geht die Überlegung, außer Funktionalität auch Architektur bereit zu stellen. Diese Überzeugung wird jetzt genauer betrachtet.

2.5.1 Vorhandene Architektur

Wird eine Geschäftsanwendung von Grund auf neu entwickelt, wird ein erheblicher Teil der Entwicklungszeit dazu verwendet, die Infrastruktur der Software zu konzipieren. Es sind viele Fakten abzuwägen. Reicht eine reine Client/Server Architektur oder soll *Middleware* eingesetzt werden. Wird alles selbst entwickelt oder werden Teile dazugekauft ? Wie sollen die persistenten Daten gespeichert werden ?

Daneben muss entschieden werden, welche Programmiersprache und welche Werkzeuge eingesetzt werden. Dann müssen Konventionen für die Dokumentation und der Programmierung erstellt werden. Nicht zuletzt muss entschieden werden, wie die Funktionalitäten der Software miteinander kooperieren. Soll eine prozessorientierte Sicht oder eine Materialsicht bevorzugt werden ?

Es muss aber auch konkret eine Programmarchitektur festgelegt werden. Soll modular, objekt-orientiert oder ein Framework entwickelt werden ? Wie werden die einzelnen Teile gekoppelt ?

Beantwortet eine Softwareprodukt diese Fragen bereits, muss nur überprüft werden, ob es die gestellten Anforderungen erfüllt. Eine solche Software liefert bereits die entscheidenden Architekturüberlegungen für die Programmierung, aber auch für die Ausführung. Durch Verwendung der Architektur wird der Entwicklungsprozess beschleunigt. Die bestehende Architektur muss nicht neu konzipiert und entwickelt, sondern nur verstanden werden.

Entwickler(innen) gemeinsam Kode erzeugen. Zudem kann jeder überall Kode ändern. Um sicherzustellen, dass die gewünschte Funktionalität nach Änderungen immer noch vorhanden ist, werden Unit-Tests eingesetzt. Diese Tests prüfen vorher festgelegte Zusicherungen und Eigenschaften kleiner Softwareeinheiten. Weitere Informationen gibt es im Internet unter http://www.extremeprogramming.org.

2.5.2 Vorhandene Funktionalität

Bietet eine Software bereits Geschäftslogik an, kann diese die Softwareentwicklung ebenfalls beschleunigen. Hier gilt gleiches wie für die Architektur. Es muss geprüft werden, ob die vorhandene Funktionalität oder Geschäftslogik den Anforderungen entspricht. Falls ja, kann sie sofort übernommen werden. Falls nicht, muss sie leicht verändert oder erweitert werden können, was wesentlich von der Softwarearchitektur abhängt.

2.5.3 Komplexität und Erlernbarkeit

Werden Teile einer Geschäftsanwendung bereits mitgeliefert, hängt deren Einsatz von zwei Dingen ab. Die Teile müssen den Anforderungen genügen und schnell begreifbar sein. Dabei ist die Komplexität entscheidend. Sind Teile äußerst komplex, ist man schnell geneigt, die Dinge selber zu kodieren. Die Komplexität hängt von der Anzahl verschiedener Konzepte, die zu erlernen sind, und deren durchgängiger Einhaltung ab. Die Erlernbarkeit hängt maßgeblich von der Vollständigkeit und Verständlichkeit der Dokumentation ab.

2.5.4 Werkzeuge für den Softwareprozess

Der Softwareprozess und dadurch die Entwicklung wird wesentlich durch gute Werkzeuge vereinfacht. Modellierwerkzeuge können die Komplexität der Entwicklung verringern, indem sie verschiedene Abstraktionsebenen durch unterschiedliche Sichten unterstützen. Sinnvoll ist eine Weiterverwendung der Arbeitsergebnisse auf einer unteren Ebene. Ein solches Werkzeug sollte eine evolutionäre Entwicklung zulassen.

Es sind noch weit mehr Einsatzbereiche für Werkzeuge vorstellbar. Software kann z.B. die Auslieferung der Software, das Erstellen von Prototypen, das Management des Softwareprozess usw. unterstützen. Diese Bereiche sind aber eher unabhängig von der zu erstellenden Geschäftsanwendung und des Programmiermodells und werden deshalb nicht betrachtet.

2.6 Bewertungskatalog

Dieser Abschnitt fasst die vorangegangenen Punkte als Bewertungskatalog zusammen. Die Kriterien sind in drei Bereiche gegliedert. Zum einen in allgemeine Kriterien für Softwarequalität, weiterhin in Kriterien, die der Flexibilität der Software dienen und zuletzt in solche, die die Softwareentwicklung vereinfachen. SanFrancisco wird in Kapitel 8 (Seite 53) anhand der einzelnen Kriterien bewertet.

Bewertungskatalog zur Beantwortung der Fragestellung:

Kriterium	**Beschreibung**
Ziel 1) Allgemeine Kriterien	
Softwarequalität	Wodurch werden welche Qualitätskriterien in welchem Maße erfüllt ?
Robustheit	Wie fehlertolerant und fehleranfällig ist das System ? Werden Korrekturmaßnahmen angeboten ?
Ausfallsicherheit	Sind Komponenten verteilt ? Gibt es eine zentrale Instanz ?
Mandantenfähigkeit	Können Daten und Funktionalitäten pro Mandant getrennt werden ?
Rechtemanagement	Gibt es Benutzer, Gruppen und Rechte für Prozesse ?
Internationalisierbarkeit	
Softwareprozess	Welche Qualität hat der vorgeschlagene Softwareprozess ?
Ziel 2) Flexibilität	
Offenheit	Welche Standards werden eingesetzt und wie wichtig sind diese in der IT-Branche ?
Integrationsfähigkeit	Können andere Systeme integriert werden ? Wenn ja, wie ?
Erweiterbarkeit	Ist die Software erweiterbar ohne die Softwarearchitektur zu ändern ?
Skalierbarkeit	Welche Verteilungsszenarien ermöglicht die Software ?
Ziel 3) Vereinfachung der Softwareentwicklung	
Architektur	Modular, objektorientiert, als Rahmenwerk oder Komponenten ?
Funktionalität	Ist die vorhandenen Funktionalität brauchbar und verständlich ?
Erlernbarkeit	
• Dokumentation	Welche Teile sind wie dokumentiert ?
• Komplexität	
Werkzeuge	Welche Werkzeuge unterstützen welche Aufgaben des Softwareprozess ?

3 Hintergründe der IBM zur Entwicklung von SanFrancisco

Dieses Kapitel stellt die Hintergründe der IBM vor, SanFrancisco zu entwickeln. Dazu gehört im Wesentlichen die Zielsetzung, die IBM mit SanFrancisco verbindet.

IBM erkannte Anfang der 90er Jahre, dass viele Softwarehersteller den aktuellen Marktanforderungen an integrationsfähiger und leicht anpassbarer Software nicht alleine gewachsen waren. Viele Softwarehersteller hatten jahrelange Branchenerfahrung, aber keine Erfahrung im Umgang mit objektorientierten Softwaremethoden. IBM sah in objektorientierten Softwaresystemen die Grundlage zu einer neuen Generation von Applikationen und wollte Softwarehersteller tatkräftig bei der Entwicklung moderner, objektorientierter, verteilter und internetfähiger Software unterstützen.

IBM entschloss sich 1994 mit Entwicklungspartnern Softwarekomponenten zur Entwicklung verteilter, objektorientierter Geschäftsanwendungen zu erstellen. Diese Komponenten sollten eine verteilte Infrastruktur und Persistenz ermöglichen, aber auch Teile der Geschäftsprozesse bereitstellen. Das erste Treffen fand in San Francisco statt, das namensgebend für das Projekt war. Die Entwicklungspartner sollten das nötige Domänenwissen beisteuern, IBM die nötige Kompetenz zur Entwicklung objektorientierter Software.

Das SanFrancisco Projekt sollte zunächst in C++ entwickelt werden. Das stellte sich bei der Entwicklung erster Prototypen als zu komplex heraus. Darum stellte IBM 1996 die Entwicklung komplett auf Java um. Ende 1997 wurde die erste Version von IBM SanFrancisco ausgeliefert, aktuell die Version 2.1. Das bedeutet:

> **IBM SanFrancisco** ist eine Java-basierte Software zur Entwicklung verteilter, internetfähiger, geschäftskritischer Geschäftsanwendungen.

IBM will mit SanFrancisco keine betriebliche Standardsoftware erstellen, sondern ihre Kunden dazu in die Lage versetzen. So erklärt sich auch der anfängliche Entwicklungsfokus auf die Bereiche Buchhaltung, Lagerverwaltung, Bestellwesen und Vertrieb. Neuerdings kommen Komponenten in den Bereichen Personalwesen und Kundenbetreuung (*Customer Relationship Management* CRM) dazu.

3.1 Zielsetzung der IBM

IBM ist der Überzeugung, dass SanFrancisco stabil, einfach erweiterbar, schnell erlernbar und im höchsten Maße integrierbar und skalierbar sein muss, damit es zur Entwicklung von Geschäftsanwendungen verwendet werden kann.

Die gestellten Anforderungen lassen sich in drei Bereiche gliedern: Zum einen müssen Softwareentwickler von SanFrancisco überzeugt werden. Zum anderen wird eine neue

Geschäftsanwendung nur eingesetzt und bezahlt, wenn sie einen echten Vorteil gegenüber bereits eingesetzter Software verspricht. Zu guter letzt sollte jede gute Softwarelösung bei wachsenden Anforderungen mit wachsen können. Das macht die detaillierten Ziele, die IBM mit SanFrancisco verbindet, verständlich.

3.1.1 Einfacher Zugang zu objektorientierter Softwareentwicklung

Um möglichst viele Softwarehersteller dazu zu bewegen, ihre Applikationen auf SanFrancisco basierend zu entwickeln, muss SanFrancisco für sie attraktiv sein. Es gilt, durch sie verstandene und beherrschte Techniken und Systeme abzulösen. Es genügt nicht, alleine auf die Vorteile von SanFrancisco zu setzten. Der Umstieg muss den Nutzern so einfach wie möglich gemacht werden. Jeder der folgenden Punkte versucht dazu beizutragen:

- **Bis zu 40% einer Anwendung sollen bereits mitgeliefert werden.** SanFrancisco soll einen Großteil der abzubildenden Geschäftsprozesse bereits enthalten, nicht aber die Benutzungsschnittstelle. Diese wird von den Softwareherstellern entwickelt. Die bereits enthaltene Funktionalität soll bis zu 40% der fertigen Software ausmachen.
- **Das Produkt soll sich anhand von Beispielen erklären.** Anhand von Beispielen soll die prinzipielle Arbeitsweise von SanFrancisco dem Entwickler verständlich gemacht werden. Er soll sich einen Überblick verschaffen können, ohne vorher alle Mechanismen der Softwareerstellung im SanFrancisco-Umfeld verstanden und Quellkode geschrieben zu haben.
- **Eine evolutionäre Softwareentwicklung muss möglich sein.** Softwarehersteller sollen in der Lage sein, ihre Applikationen sukzessive auf SanFrancisco umzustellen. SanFrancisco muss in bestehende Informationssysteme integriert werden und Daten mit diesen gemeinsam nutzen können.
- **Anpassungen und Erweiterungen müssen einfach sein.** Die Softwarekomponenten sollen eine Grundlage für eine Vielzahl verschiedener Applikationen darstellen. Bestehende Komponenten müssen einfach erweiterbar, neue leicht zu integrieren sein. Außerdem sollen alle Komponenten weltweit einsetzbar sein, was nicht nur die multilinguale Unterstützung meint, sondern vielmehr die Abbildung rechtlicher Unterschiede verschiedener Länder bei der Durchführung von Geschäftsprozessen.
- **Persistente Applikationsobjekte sollen durch die Firma, die SanFrancisco einsetzt, frei wählbar gespeichert werden können.** Firmen, die SanFrancisco einsetzen, soll bei der Speicherung persistenter Applikationsobjekte möglichst viel Freiheit gelassen werden. Es sollen verschiedene objektorientierte und relationale Datenbanken unterstützt werden. In der Entwicklungsphase und zu Demonstrationszwecken ist sogar eine simple Speicherung im File-System hilfreich.
- **Dem Entwickler sollen integrierte Werkzeuge die Arbeit erleichtern.** IBM meint, dass ein System zur Softwareentwicklung durch Werkzeuge besticht, die dem Entwickler wiederkehrende Aufgaben abnehmen. Es ist denkbar, Modellierwerkzeuge und Kodegeneratoren bereitzustellen. Weiterhin wünschenswert ist eine integrierte Entwicklungsumgebung. Ein Vorgehensmodell erleichtert unerfahrenen Entwicklern den Zugang zur objektorientierten Softwareentwicklung.

3.1.2 Marktvorteil durch SanFrancisco Applikationen

IBM geht davon aus, dass die Einführung neuer Software in Unternehmen stets mit großem Aufwand verbunden ist. Sollen bestehende Softwarelösungen ersetzt werden, muss die neue Software beachtliche Vorteile gegenüber der bereits eingesetzten aufweisen. Neue Software muss einen Marktvorteil generieren. IBM hat diese Überlegung in die Anforderungen an SanFrancisco aufgenommen.

- **Ein gutes objektorientiertes Design soll Anpassungen erleichtern.** SanFrancisco soll ein gutes, robustes objektorientiertes Design erhalten. Dadurch sollen SanFrancisco Applikationen leicht erweiterbar und schnell an Änderungen von Betriebsabläufen anpassbar sein.
- **Auf SanFrancisco basierende Applikationen sollen auf einen oder mehrere Server und Clients verteilt werden können.** Für den Umgang mit verteilten Geschäftsobjekten soll SanFrancisco dem Entwickler geeignete Dienste, wie verteilte Transaktionsverwaltung, Namensdienste, Zugriffskontrolle und Persistenz anbieten. Diese Infrastruktur soll dem Entwickler eine einfache und transparente Verteilung der Applikationslogik auf mehrere Server und Clients ermöglichen.
- **Umsetzung des MVC Konzepts, wodurch die Benutzungs-schnittstellen austauschbar ist.** SanFrancisco basierte Applikationen sollen das *Model-View-Controller* (MVC) Konzept umsetzen. Hier bedeutet die Umsetzung des Konzepts eine Trennung des Geschäftsobjekts (*Model*) von dem Nutzungskontext (*Controller*) und der Darstellung (*View*). So kann jeder der drei Bereiche ausgetauscht werden, ohne Änderungen an den übrigen vornehmen zu müssen.
- **SanFrancisco-Komponenten müssen robust und getestet sein.** Die Stabilität einer Applikation ist für Unternehmen ein wichtiges Auswahlkriterium. IBM möchte mit SanFrancisco fertige, robuste und getestete Komponenten anbieten.

3.1.3 Lösungen für verschiedene Hardware und Betriebssysteme

SanFrancisco soll für verschiedene Hardware und Betriebsysteme angeboten werden. Softwarehersteller haben so die Möglichkeit, auf Unternehmen optimal zugeschnittene und dadurch kostengünstige Lösungen anzubieten.

Konkret soll SanFrancisco folgende Betriebsysteme unterstützten:

- Auf der Clientseite: Windows 95, Windows NT, MacOS und alle anderen Betriebsysteme, für die eine Java Virtual Machine (JVM) existiert
- Auf der Serverseite: Windows NT, AIX, OS/400, Solaris, HP-UX

Zusätzlich soll SanFrancisco folgende Applikationsarchitekturen unterstützen:

- Web-Clients mit Einbeziehung von Java Applets,
- Web-Clients mit Einbeziehung von Java Servlets und
- Reine Clientapplikationen.

4 Ausführungsumgebung von SanFrancisco

Das Kapitel 3 bot einen Überblick über die Zielsetzungen der IBM bei der Entwicklung von SanFrancisco. Dadurch wurde eine erste Vorstellung davon vermittelt, was SanFrancisco eigentlich ist. Dieses Kapitel beschreibt nun detailliert die Ausführungsumgebung von SanFrancisco und mögliche Verteilungsszenarien einer auf SanFrancisco basierenden Applikation. Dazu werden die beteiligten Komponenten vorgestellt, die diese Verteilung ermöglichen. Ein weiterer wichtiger Aspekt zur Laufzeit ist die Speicherung von Daten. Daher wird die Verwaltung persistenter Objekte in SanFrancisco im zweiten Teil dieses Kapitels untersucht. Abschließend wird ein kurzer Überblick über Administrationswerkzeuge gegeben, um dieses Kapitel abzurunden.

Anhand dieser Betrachtung lassen sich Rückschlüsse auf Forderungen aus dem Kapitel 2 ableiten, konkret zur Skalierbarkeit, zur Ausfallsicherheit und über die Datenhaltung.

4.1 Das Logische SanFrancisco Netzwerk (*Logical SanFrancisco Network*)

Um Objekte einer Applikation auf verschiedene Rechner zu verteilen, bedarf es einer geeigneten Kommunikationsinfrastruktur. Bei SanFrancisco heißt diese Logisches SanFrancisco Netzwerk (*Logical SanFrancisco Network*) (LSFN) und besteht aus verschiedenen sogenannten Prozessen. Ein Prozess bündelt Dienste (*Services*), die mittels des TCP/IP (*Transmission Control Protocol / Internet Protocol*) Protokolls kommunizieren. Einige dieser Dienste sind an die Funktionalität entsprechender CORBA* Dienste angelehnt. SanFrancisco ist jedoch keinesfalls CORBA-konform, geschweige denn ein vollständiger CORBA *Object Request Broker* (ORB). Bei der Realisierung der Objektkommunikation wurden lediglich Überlegungen der *Object Management Group* (OMG) übernommen.

Der SanFrancisco Sprachgebrauch gliedert die Begriffe Rechner, Prozess und Dienst unüblich. Auf einem Rechner arbeiten verschiedene Prozesse. Die Prozesse, entgegen der sonst üblichen Verwendung des Begriffs, fassen verschiede Dienste zusammen, sind also langlebig. Dienste übernehmen einzelne, wohldefinierte Aufgaben innerhalb des Logischen SanFrancisco Netzwerks. Ein Prozess ist also eine Ausführungsumgebung für Dienste. Darüber hinaus werden gleiche Namen für unterschiedliche Dinge verwendet. Um Verwechslungen zu vermeiden, wird folgende Konvention eingehalten: Handelt es sich um einen Prozess oder Dienst wird dies dem Namen angefügt, ansonsten ist ein Rechnertyp gemeint.

* Die *Common Object Request Broker Architecture (CORBA)* ist ein Standard der Object Management Group (OMG), welcher Objektkommunikation zwischen Applikationen verschiedener Hersteller und verschiedener Plattformen ermöglicht (siehe www.omg.org).

Jedes Logische SanFrancisco Netzwerk (LSFN) besteht aus genau einem Globalen Server Manager, einem oder mehreren Servern, die Geschäftsobjekte anbieten, und einem oder mehreren Clients (siehe Abbildung 2).

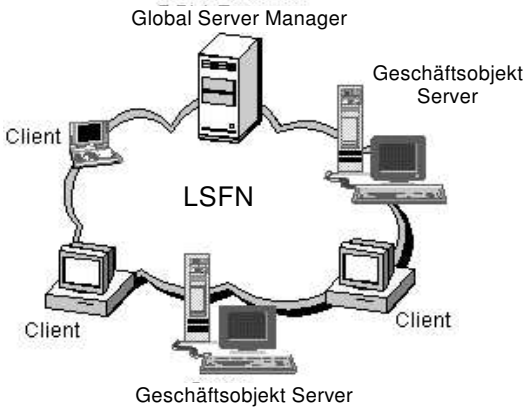

Abbildung 2: Das Logische SanFrancisco Netzwerk (LSFN)

Es ist möglich, das LSFN lokal auf einem Rechner zu installieren oder, wie in der Abbildung angedeutet, beliebig zu verteilen. Die klassische Client-Server Variante ist natürlich ebenfalls möglich.

Welche Prozesse auf diesen Rechnern welche Dienste anbieten, wird anschließend erläutert.

4.1.1 Prozesse und Dienste des Logischen SanFrancisco Netzwerks

Auf jedem Rechner des LSFN arbeiten Prozesse von denen es vier verschiedene Typen gibt: Den Globalen Server Manager (GSM) Prozess, den Server Manager Prozess, Geschäftobjektprozesse und den Client-Prozess. Innerhalb eines LSFN gibt es genau einen GSM Prozess, alle anderen Prozesse können beliebig oft vorkommen. Geschäftsobjektprozesse können auf dem Global Server Manager und Geschäftsobjekt-Servern eingerichtet werden. Die vier Prozesstypen übernehmen verschiedene Aufgaben des Netzwerks, sie bieten nämlich unterschiedliche Dienste an (siehe Abbildung 3).

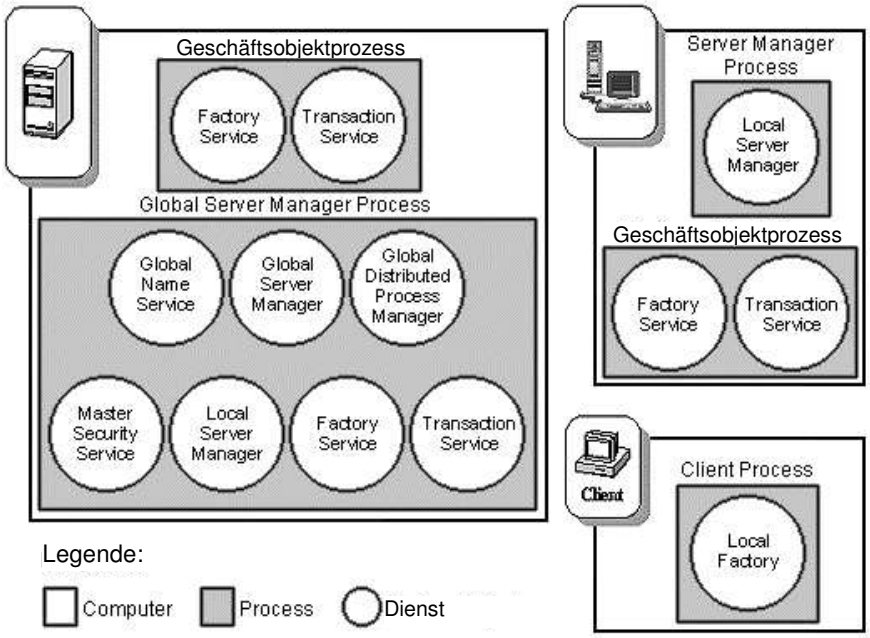

Abbildung 3: Die vier Prozesstypen eines Logischen SanFrancisco Netzwerks mit zugehörigen Diensten

Der Globale Server Manager Prozess kontrolliert das gesamte Logische SanFrancisco Netzwerk und verwaltet alle eingerichteten Geschäftsobjektprozesse. Ein Server Manager Prozess startet für den Rechner lokale Geschäftsobjektprozesse. Ein solcher ist ebenfalls im Globalen Server Manager enthalten. Die Geschäftsobjektprozesse ermöglichen den Zugriff auf Geschäftsobjekte, sind also Behälter (*Container*) für diese. Die Clientprozesse dienen der Kommunikation mit den Servern und können lokale, nicht persistente Objekte verwalten.

Der Globale Server Manager Prozess stellt alle kritischen Dienste zur Verfügung. Fällt dieser Server oder einer seiner Dienste aus, kann keine Kommunikation innerhalb des LSFN mehr stattfinden.

Es werden im folgenden die einzelnen Aufgaben aller Dienste (*Services*) beschrieben. Die Gruppierung der Dienste zu Prozessen kann der Abbildung 3 entnommen werden. Die Aufgaben der Dienste kann nicht immer detailliert angegeben werden, da in frei verfügbarer Literatur kaum Auskunft über die Arbeitsweise der Dienste gegeben wird.

4.1.1.1 Der Globale Server Manager Dienst (*Global Server Manager Service*)

Der Globale Server Manager Dienst verwaltet alle Geschäftsobjektprozesse des Logischen SanFrancisco Netzwerks. Er verfügt über alle Konfigurationsinformationen dieser Prozesse. Zusätzlich ist er für die Weiterleitung von Clientanfragen an die entsprechenden Geschäftsobjektprozesse zuständig. Fällt dieser Dienst aus, funktioniert die LSFN Kommunikation nicht mehr.

4.1.1.2 Der Bennennungsdienst (*Global Name Service*)

Der Benennungsdienst (*Global Name Service*) verwaltet den Namensraum des LSFN. Dieser Dienst verwaltet alle Informationen zur Anlage von Geschäftsobjekten. Soll eine neue Instanz eines Geschäftsobjekts angelegt werden, weiß dieser Dienst, welche

Klasse zu verwenden ist und welcher Rechner diese Instanz erzeugen muss. Außerdem löst der Dienst sogenannte Benutzer Aliase (*User Alias)* auf. Ein Benutzer Alias ist ein vom Entwickler vergebener Name für ein persistentes Objekt, um dieses einfacher und schneller abfragen zu können.

4.1.1.3 Der Lokale Server Manager Dienst (*Local Server Manager Service)*

Der Lokalen Server Manager Dienst (*Local Server Manager Service*) startet bei der ersten Anforderung von Daten den entsprechenden Geschäftsobjektprozess. Jeder Server, der Geschäftsobjektprozesse anbietet, muss diesen Dienst installieren, also auch der Globale Server Manager Prozess.

4.1.1.4 Der Hauptsicherheitsdienst (*Master Security Service*)

Der Hauptsicherheitsdienst (*Master Security Service*) übernimmt die Authentifizierung der Benutzer für das Lokale SanFrancisco Netzwerk. Der Hauptsicherheitsdienst wird von den Server Sicherheitsdiensten kontaktiert, damit diese ihre Aufgabe erfüllen können.

4.1.1.5 Der Server Sicherheitsdienst (*Server Security Service*)

Der Server Sicherheitsdienst (*Server Security Service*) autorisiert Benutzeranfragen an Geschäftsobjektprozesse. Es wird der Zugriff auf Server und die dort ausgeführten Aktionen unterschieden. Jede Geschäftsanwendung kann eigene Aktionen definieren, wie zum Beispiel das Ausführen einer bestimmten Transaktion. Der Zugriff auf den Server sowie auf einzelne Aktionen kann dann konfiguriert werden. Dieser Dienst prüft die Berechtigung einer Benutzeranfrage und lässt diese zu oder lehnt sie ab.

4.1.1.6 Der Factory Dienst (*Factory Service*)

Jeder Geschäftsobjektprozess muss den Factory Dienst (*Factory Service*) installieren. Dieser Dienst legt neue Geschäftsobjekte an und verwaltet bereits existierende. Der Dienst realisiert die einzelnen Behälter (*Container*) für die Geschäftsobjekte.

4.1.1.7 Der Transaktionsdienst (*Transaction Service*)

Der Transaktionsdienst (*Transaction Service*) überwacht alle Transaktionen aller Objekt eines Geschäftsobjektprozesses. Außerdem übernimmt dieser Dienst die Wiederherstellung (*Recovery*) im Fehlerfall.

4.1.1.8 Der Fehlerdienst (*Problem Service*)

Der Fehlerdienst (*Problem Service*) ist optional und ermöglicht dem Administrator Fehler der Dienste aufzuspüren und zu beheben. Alle Probleme der Dienste werden protokolliert.

4.1.1.9 Der Konfliktsteuerungsdienst (*Conflict Control Service*)

Bei der Durchführung von Aktionen können Konflikte auftreten, wenn zum Beispiel eine Aktion A nicht durchgeführt werden darf, während die Aktion B durchgeführt wird. Es kann auch Aktionen geben, die sinnvoll nur einmal durchzuführen sind. Ein Beispiel wäre die Erstellung eines Jahresabschlusses. Der Konfliktsteuerungsdienst (*Conflict Control Service*) bietet dem Administrator die Möglichkeit, solche Konflikte zu erkennen und aufzulösen.

4.1.1.10 Der Logische Sperrdienst (*Logical Lock Service*)

Der Logische Sperrdienst (*Logical Lock Service*) bietet Applikationen die Möglichkeit, logische Sperren bei Modifikationsoperationen zu setzen. Dadurch können Benutzer Objekte einsehen, während ein anderer diese verändert. Bei lang andauernden Transaktionen ist ein solches Vorgehen sinnvoll. Der Dienst lässt jeweils nur eine schreibende Sperre auf ein Objekt zu. Der Logische Sperrdienst ist optional.

4.1.2 Die Rolle der Dienste in Beispielszenarios

Die bisherige Beschreibung der Rechner- und Prozesstypen sowie der dazugehörigen Dienste muss unbefriedigend bleiben. Es war geplant das Zusammenwirken der Dienste anhand einfacher Szenarios vorzustellen. So wäre die Funktion jedes Dienstes einsichtig gewesen. In der zugänglichen Literatur über SanFrancisco ist aber nicht mehr in Erfahrung zu bringen, als oben bereits dargestellt wurde.

4.2 Persistenz SanFrancisco basierter Geschäftsobjekte

Geschäftsanwendungen müssen Daten effizient persistent speichern. Zusätzlich muss die Datenintegrität zu jedem Zeitpunkt gewährleistet sein. SanFrancisco Geschäftsobjekte können persistent und transaktionssicher gespeichert werden. Dazu bietet SanFrancisco mehrere Möglichkeiten an. Geschäftsobjekte können im Dateisystem eines Rechners oder in Relationalen Datenbanken (RDB) gespeichert werden.

Werden Geschäftsobjekte im Dateisystem gespeichert, wird für jede Klasse ein Ordner und für jedes persistente Objekt eine Datei angelegt. Die Attribute des Objekts werden serialisiert in die Datei gespeichert. Diese Art der Persistenz ist äußerst ineffizient. Zu Demonstrationszwecken oder in der Entwicklungsphase ist sie dennoch hilfreich.

Datenbank Management Systeme sind effizienter beim Auffinden von Daten als das Dateisystem. Häufig genutzte Suchpfade können durch Indizes unterstützt und beschleunigt werden. Datenbanksysteme bieten darüber hinaus weitere Vorteile. Sie bieten die bessere Skalierbarkeit durch die Möglichkeit des *Load-Balancing* oder der Replikation. Daten aus Datenbanken stehen anderen Applikation leichter zur Verfügung, da diese direkt darauf zugreifen können (zum Beispiel Reportingtools wie Crystal Reports), um nur einige Vorteile zu nennen.

SanFrancisco unterstützt folgende Relationale Datenbank Management Systeme (RDBMS):

- IBM DB2 Vers. 5.5 für Microsoft Windows NT 4
- IBM DB2 / 6000
- Oracle8 für Windows und AIX
- IBM DB2 / 400 für AS/400 (Vers. 4.3 oder später)
- Microsoft SQL Server 7.0 für Windows NT 4

Objektorientierte Datenbanken werden durch SanFrancisco nicht unterstützt, was eigentlich verwundert, da Java und damit SanFrancisco objektorientiert ist. Objekte können nicht ohne weiteres in relationalen Datenbanken gespeichert werden. Vorher muss eine nicht triviale Abbildung (*Mapping*) der Objekte auf Relationen durchgeführt

werden (siehe Abbildung 4). Ein Grund für die fehlende Unterstützung objektorientierter Datenbanken ist sicherlich, dass diese sich nicht am Markt durchsetzen konnten. Relationale Datenbanken sind bisher bei weitem performanter (siehe [AKC00]).

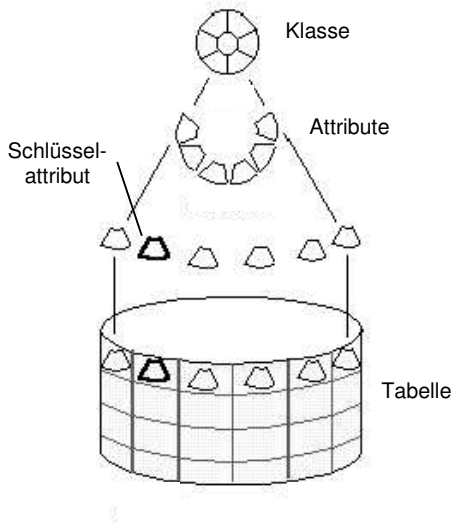

Abbildung 4: Abbildung von Objekten einer Klasse auf Relationen einer relationalen Datenbank

Für die Speicherung von Objekten in Relationen einer relationalen Datenbank gilt prinzipiell: Klassen werden durch Tabellen, Objekte durch Zeilen und Attribute durch Spalten abgebildet (siehe Abbildung 4). Um ein Objektmodell vollständig relational abzubilden, müssen insgesamt Abbildungen

- von Klassen,
- von komplexen Datentypen,
- von Subklassen,
- von Beziehungen verschiedener Klassen und Objekte und
- die Abbildung von Kardinalitäten dieser Beziehungen

definiert werden (siehe [AKC00]).

SanFrancisco bietet zwei Arten der Abbildung an, eine automatische und eine manuelle. Die automatische Schema Abbildung (*Default Schema Mapper DSM*) ermittelt per Reflexion der Klasse und durch Anwendung von einfachen Regeln eine Tabellenstruktur, die Objekte dieser Klasse aufnimmt. Die manuelle Schema Abbildung (*Extended Schema Mapper ESM*) bietet Möglichkeiten zum Eingriff in die automatische Abbildung. Die manuelle Abbildung wird in eine Datei gespeichert (SML-Datei), welche der Ersetzung der automatischen Regeln dient. Über die manuelle Schema Abbildung ist es unter anderem möglich, bestehende Altdaten aus Legacy-Systemen in SanFrancisco Applikationen zu nutzen.

Es ist zu prüfen wie SanFrancisco diese Abbildungen konkret durchführt und welche zusätzlichen Möglichkeiten die manuelle Abbildung bietet.

4.2.1 Objekt-Relationale Abbildung von SanFrancisco

Klassen werden generell auf Tabellen und Attribute einer Klasse auf Spalten abgebildet. Tabellenzeilen werden als Objekte aufgefasst. Die Abbildung einfacher Datentypen (`boolean`, `int`, `Date`) der Attribute erfolgt in korrespondierende Datentypen der relationalen Datenbank, was auch für *Wrapper*-Klassen (`Integer`, `Boolean`) gilt. Komplexen Attributtypen (`Array`, `Object`, `Collection`) entsprechen keine Datentypen relationaler Datenbanken. Für diese Attributtypen sind Abbildungen zu definieren. Hier ergeben sich die ersten Eingriffsmöglichkeiten durch die manuelle Schema Abbildung (ESM).

String Attribute: Bei der Abbildung von String Attributen in Spalten müssen verschiedene Längen und Zeichenformate berücksichtigt werden. String Attribute werden automatisch in VARCHAR(128) Spalten im *Unicode*-Format* gespeichert. Das bedeutet, dass maximal 64 Zeichen gespeichert werden können, da ein *Unicode*-Zeichen 2 Bytes beansprucht.

ESM: Die manuelle Schema Abbildung bietet die Möglichkeit, die Länge der Tabellenspalte und das Zeichenformat anzupassen. String Daten können in *Unicode*, UTF8 und der Datenbank eigenen Zeichenkodierung gespeichert werden.

Array Attribute: Array Attribute werden standardmäßig als Datenstrom (*Stream*) in eine LONG VARBINARY Spalte geschrieben.

ESM: Array Attribute lassen sich manuell durch drei weitere Abbildungen in eine Datenbank speichern:

- Arrayelemente werden in einer weiteren Tabelle abgelegt.
- Arrayelemente werden in weiteren Spalten gespeichert.
- Arrayelemente werden in verschiedenen Tabellen gespeichert.

***Handle* Attribute:** *Handle* sind Verweise auf andere Objekte. Diese Verweise werden als Datenstrom in VARBINARY(128) Spalte geschrieben.

ESM: Ein Handle besteht mindestens aus einer Klassen ID und einer Objekt ID. Es kann eine Behälter ID dazukommen. Außerdem kann die automatisch von SanFrancisco erzeugte Objekt ID durch einen selbst definierten Key ersetzt werden. Diese Attribute können in einzelne Spalten abgebildet werden. Die Behälter ID und Klassen ID können sogar entfallen. Diese müssen dann als Standardwerte in der Beschreibungsdatei (SML-Datei) festgelegt werden.

***Dependent* Attribute:** *Dependent* Attribute sind abhängige Objekte. Diese Objekte werden prinzipiell ebenso wie normale Objekte abgebildet, also deren Attribute in Spalten gespeichert. Trifft eine der folgenden Bedingungen zu, wird das *Dependent* Attribut als Datenstrom in eine LONG VARBINARY Spalte geschrieben:

- Das *Dependent* Objekt enthält mehr als ein Attribut, das als Datenstrom abgebildet wird.
- Das *Dependent* Objekt ist ein `Interface`.

---

* Unicode gibt jedem Zeichen seine eigene Nummer, systemunabhängig, programmunabhängig, sprachunabhängig (siehe www.unicode.org).

- Das *Dependent* Objekt ist abstrakt.
- Das *Dependent* Objekt enthält einen Verweis auf ein Objekt der selben Klasse.

ESM: Die manuelle Schema Abbildung erlaubt, *Dependent* Attribute in weiteren Tabellen zu speichern.

***Entity* Attribute**: Wie ein *Dependent* Attribut sind *Entity* Attribute Objekte und werden anlog abgebildet.

Subklassen: Subklassen werden standardmäßig in jeweils separaten Tabellen gespeichert. Jede Klasse wird durch eine eigene Tabelle abgebildet. Alle Attribute, also auch geerbte, werden durch Spalten abgebildet.

ESM: Die manuelle Schema Abbildung bietet mehrere Möglichkeiten, Subklassen relational abzubilden. Zur Verdeutlichung wird folgendes Beispiel herangezogen. Angenommen die Klasse Manager ist Subklasse der Klasse Mitarbeiter. Diese beiden Klassen können wie folgt in Tabellen gespeichert werden:

- Gemeinsame Attribute werden in einer Tabelle abgelegt. Alle spezifischen Attribute der Klasse Manager werden in einer weiteren Tabelle gespeichert. Um auf die Klasse des Objekts schließen zu können, wird die Klassen ID in einer Spalte der Tabelle Mitarbeiter gespeichert.

Tabelle Mitarbeiter

ObjectId	ClassId	Name	DeptNo	Level
1234122	48898	J.J. Black	A101	B1
3445221	48898	R.I. Red	A101	A4
282910	55432	I.O. Dick	A101	A8

Tabelle Manager

ObjectId	ReportTo
282910	D333

- Die Klassen ID kann entfallen, wenn gilt: Wird keine Zeile in der Tabelle Manager mit derselben `ObectId` gefunden, handelt es sich um ein Mitarbeiter Objekt.
- Beide Klassen lassen sich in einer Tabelle wie folgt darstellen:

Tabelle Mitarbeiter

ObjectId	ClassName	Name	DeptNo	Level	ReportTo
1234122	Mitarbeiter	J.J. Black	A101	B1	Null
3445221	Mitarbeiter	R.I. Red	A101	A4	Null
282910	Manager	I.O. Dick	A101	A8	D333

- Hier kann wie im oberen Beispiel der Klassenname entfallen wenn eine entsprechende Regel definiert wird. Ist die Spalte `ReportTo` ungleich `Null`, handelt es sich um ein Manager-Objekt.

Wie Beziehungen und Kardinalitäten von Beziehungen abgebildet werden, ist aus der Literatur nicht zu entnehmen. Die Werkzeuge lassen vermuten, dass Fremdschlüssel zwischen Tabellen definiert werden, da dazu Einstellungen angeboten werden. Wie das Ergebnis konkret aussieht, lässt sich nur erahnen und wird hier deshalb nicht vorgestellt.

4.2.2 Anfragen an die Datenbank delegieren

SanFrancisco bietet dem Softwareentwickler eine *Object Query Language* (OQL) zum Auffinden von Objekten an. Die Abfragesprache ist unabhängig von der Art und Weise der Persistenz (Posix oder RDB) und damit auch von der eingesetzten Datenbanksoftware.

Das Ziel der manuellen Schema Abbildung besteht darin, Anfragen an die Datenbasis durch die Datenbank durchführen zu lassen. Dieses Verfahren wird Abfragedelegation (*Query Pushdown*) genannt. Konkret bedeutet dies, unspezifische Anfragen, also solche, die Geschäftsobjekte nicht über die Objekt ID identifizieren, an die Datenbank zu delegieren. Möchte man eine Übersicht über alle Mitarbeiter mit Namen Müller erhalten, so könnte eine *Object Query Language* (OQL) Anfrage wie folgt aussehen:

```
Select e from Emps e where (e.getName() = "Müller");.
```

Diese Anfrage kann dadurch beantwortet werden, dass alle Mitarbeiter Objekte instanziiert werden und dann der Inhalt des Attributs `name` mit dem Text `"Müller"` verglichen wird. Andererseits könnte die Datenbank die Spalte `name` mit dem Text `"Müller"` vergleichen und nur die entsprechenden Zeilen zurückliefern, die dann instanziiert werden. Ein Objekt zu instanziieren ist ein teurer Vorgang, wodurch die zweite Möglichkeit die effizientere ist. Die *Structured Query Language* (SQL) Anweisung zu obigem Beispiel könnte wie folgt aussehen:

```
select * from Emps where name = "Müller".
```

Die Transformation einer OQL Anfrage in die korrespondierende SQL Anfrage übernimmt SanFrancisco, falls diese möglich ist. Die Transformation ist allerdings nur für einfache Attributtypen möglich.

4.2.3 Automatische oder manuelle Objekt-Relationale Abbildung

Die automatische Schema Abbildung ist ein sehr einfacher Weg, Objekte relational zu speichern. Für große Geschäftsanwendungen, die große Datenmengen verwalten, scheint die automatische Schema Abbildung allerdings nicht geeignet zu sein. Um eine maximale Performanz der Anwendung zu erzielen, ist eine manuelle Schema Abbildung notwendig. Diese Abbildung kann die Verwendung der Daten berücksichtigen und Optimierungen vornehmen. Sie erlaubt den Einsatz der Abfragedelegation (*Query Pushdown*) an die Datenbank. Interessant ist die automatische Schema Abbildung trotzdem, da die manuelle Abbildung der Objekte aufwendig ist. Für selten genutzte Geschäftsobjekte oder solche, die nur in wenigen Ausprägungen vorkommen, lohnt sich der Aufwand nicht. Die Kombination der manuellen und automatischen Schema Abbildung ist daher sicherlich optimal.

4.3 Administrationswerkzeuge

Im Lieferumfang von SanFrancisco sind bereits die notwendigen Administrationswerkzeuge enthalten. Es gibt ein Werkzeug zur Verwaltung der Benutzer, eines zum Rechtemanagement und eines zur Konfiguration des Lokalen SanFrancisco Netzwerks (*Local SanFrancisco Networks*).

Die Benutzerverwaltung kennt Gruppen und einzelne Benutzer, die verschiedenen Gruppen angehören können. Pro Benutzer kann angegeben werden, wie sich dieser an das System anmelden kann. Das dazu nötige Kennwort kann in Güte und Gültigkeit konfiguriert werden.

Das Rechtemanagement unterscheidet Serverberechtigungen und Berechtigungen an speziellen Ressourcen. Solche Ressourcen können benannte Aktionen sein, die während der Entwicklung angelegt und benannt wurden. Der Zugriff auf solche Ressourcen wird Gruppen oder Benutzern zugeordnet.

Das wichtigste Werkzeug ist das zur Konfiguration des Lokalen SanFrancisco Netzwerks. Hier werden alle Rechner und deren Prozesse definiert und konfiguriert.

Alle Werkzeuge sind als eigenständige Java-Programme realisiert, die auf allen unterstützten Serverplattformen lauffähig sind.

5 Software-Architektur des SanFrancisco Rahmenwerks

In den vorherigen Kapiteln wurde SanFrancisco vorgestellt und danach die Komponenten und Dienste der SanFrancisco Ausführungsumgebung beschrieben. Dieses Kapitel untersucht die Architektur der Software, also die Schnittstelle zwischen SanFrancisco und dem Softwareentwickler. Beginnend mit den Designüberlegungen von IBM wird die Schichtenarchitektur von SanFrancisco, die Funktionen der Schichten und deren Interaktion beschrieben.

Wie bereits erwähnt, ist IBM SanFrancisco ein Java-basiertes Rahmenwerk zur Entwicklung von Geschäftsanwendungen. Bei der Designphase von SanFrancisco wurde großer Wert auf die Verwendung von Entwurfsmustern (*Design Pattern*) gelegt. Es existierte nicht zu jedem Problem ein adäquates Muster. Daher wurden neue SanFrancisco Entwurfsmuster entwickelt und in Buchform veröffentlicht (siehe [CCG00]).

Außer Entwurfsmuster einzusetzen, wurde SanFrancisco in klar abzugrenzende Schichten gegliedert. Jede dieser Schichten (*Layer*) übernimmt unterschiedliche Aufgaben (siehe Abbildung 5) und ist verschieden nah an Geschäftsabläufen orientiert. Es sind sechs Schichten auszumachen: Die von IBM SanFrancisco gelieferten Schichten finden sich in der Mitte, die Schichten darunter stellen die Hardware-Plattformen und die *Java Virtual Machine* (JVM) dar, die Schicht darüber enthält die Anpassungen der Softwarehersteller, die SanFrancisco einsetzen.

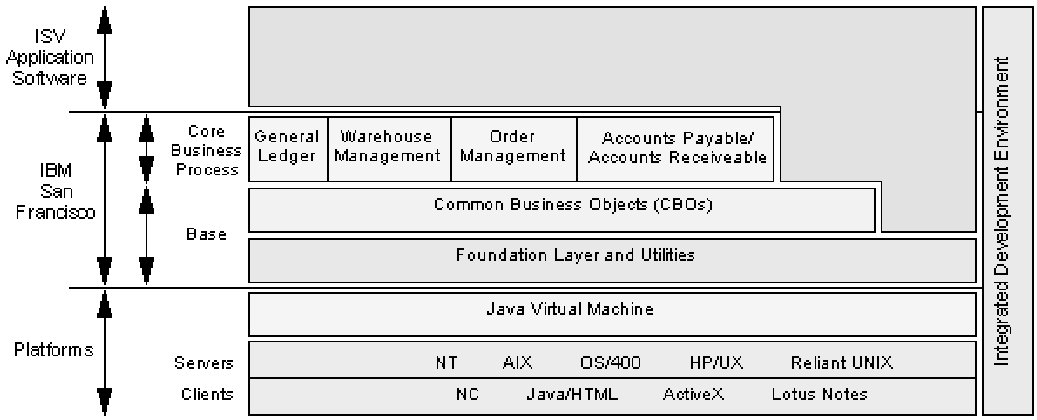

Abbildung 5: Die Software-Architektur des SanFrancisco Frameworks

SanFrancisco gliedert sich in eine Fundamentschicht (*Foundation-Layer*), eine Allgemeine Geschäftsobjektschicht (*Common Business Objects Layer*) und die Kerngeschäftsprozessschicht (*Core Business Process Layer*). Jede dieser Schichten kann von Softwareherstellern erweitert und verändert werden.

Die Fundamentschicht enthält alle in Kapitel 3 beschriebenen Dienste und bildet damit die Grundlage des Logischen SanFrancisco Netzwerks (*Logical SanFrancisco Network* LSFN). Die Allgemeine Geschäftsobjektschicht enthält Businessobjekte, die alle Geschäftsanwendungen nutzen (zum Beispiel: Firma oder Bank). Die Kerngeschäftsprozessschicht implementiert spezielle Businessobjekte und Funktionalitäten bestimmter Geschäftsfelder (zum Beispiel: Warenlager, Kostenstelle oder Bestellung).

Nachfolgend wird nur die Fundamentschicht detailliert betrachtet. Den anderen beiden ist das nächste Kapitel gewidmet. Die folgenden Informationen sind die für den Softwareentwickler entscheidenden. Alle Basiskonzepte von SanFrancisco finden sich hier wieder.

5.1 Die Fundamentschicht (*Foundation Layer*)

Die Fundamentschicht *(Foundation Layer)* implementiert die wesentlichen Säulen von SanFrancisco. Zum einen ist das die Kommunikations- und Dienststruktur des Logischen SanFrancisco Netzwerks *(Logical SanFrancisco Networks LSFN)*, zum anderen sind das Funktionalitäten, die Softwareentwickler bei der Arbeit unterstützen. Die Funktionalität des LSFN bleibt dem Softwareentwickler im wesentlichen verborgen.

Die Fundamentschicht implementiert SanFrancisco Basiskonzepte wie die Anlage von Geschäftsobjekten, Zugehörigkeit *(Ownership)* von Objekten, Unterstützung von Lokalität und Persistenz, die Softwareentwickler nutzen können. Zusätzlich zu den Basiskonzepten wird Funktionalität zum Auffinden von Geschäftsobjekten, zur Unterstützung von Sicherheitsmechanismen und ein Benachrichtigungsservice angeboten.

IBM führt eine eigene SanFrancisco Objekthierarchie ein, die auch Behälterklassen beinhaltet (siehe Abbildung 6). Ein SanFrancisco Geschäftsobjekt ist immer eine Subklasse der Klassen `Entity` oder `Dependent`, die wiederum eine Subklasse der Klasse `BusinessObject` ist.

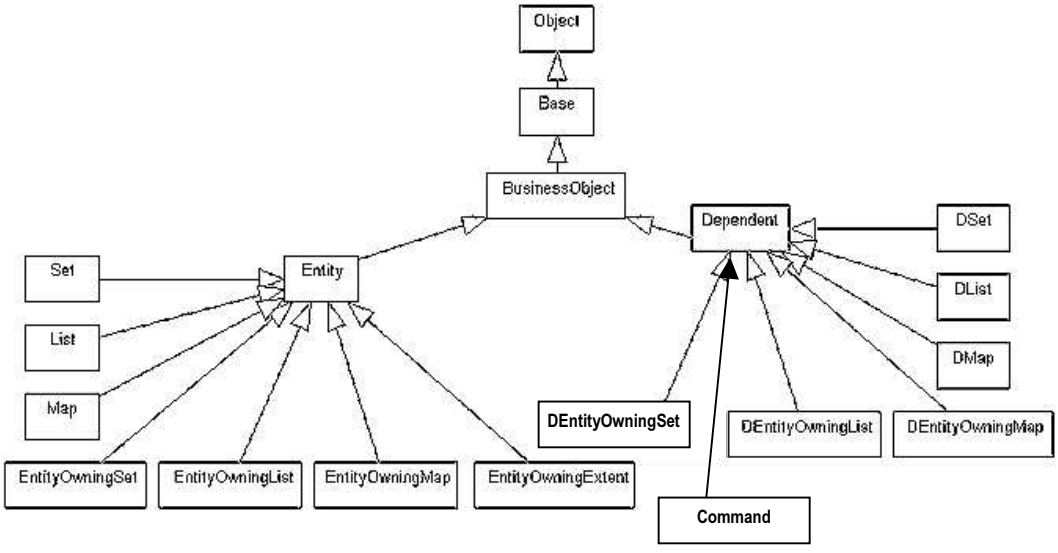

Abbildung 6: Die Klassenhierarchie der SanFrancisco Basisklassen

Subklassen von `Entity` sind persistente Objekte und können innerhalb von Transaktionen verändert werden. Von anderen Objekten abhängige Objekte, also nicht eigenständige Objekte, werden von `Dependent` abgeleitet. Diese Objekte sind transient, können aber als *Member* eines persistenten Objekts ebenfalls persistent sein. Eine weitere wichtige Basisklasse ist die von `Dependent` abgeleitete Klasse `Command`. Subklassen von `Command` implementieren Aktionen, bzw. Teile von Geschäftsprozessen. Führen `Commands` Datenänderungen durch, können `undo()` und `redo()` Methoden implementiert werden.

Die Implementierung eines SanFrancisco Geschäftsobjekts sieht die Implementierung von drei Klassen vor. Es muss eine Interface-Klasse, eine Factory-Klasse und eine Implementierungsklasse entwickelt werden. Die Interface-Klasse definiert alle Methoden und Attribute, die das Geschäftsobjekt anbietet. Die Factory-Klasse wird verwendet, um eine neue Instanz dieses Geschäftsobjekts zu erzeugen. Die Implementierungsklasse implementiert die tatsächliche Programmlogik. Da diese Konstruktion, bis auf die Implementierungsklasse, für jedes Geschäftsobjekt ähnlich aussieht, bietet sich die Verwendung eines Kodegenerators an. Zusätzlich zu diesen drei Klassen können eine *Stub* und eine *Skeleton* Klasse für die entfernte Verwendung der Objekte hinzukommen.

5.1.1 Anlage eines Geschäftsobjekts

Da Geschäftsobjekte sich wesentlich von „normalen" Java Objekten unterscheiden (Persistenz, Lokalität), werden sie anders als in Java üblich erzeugt. Der `new` Bezeichner von Java reicht hier nicht aus und darf nicht zur Anlage von SanFrancisco Geschäftsobjekten verwendet werden. Statt dessen wird dazu die `BaseFactory` herangezogen, die über die globale Klasse `Global` erreichbar ist (`Global.factory()`). Dieses `BaseFactory`-Objekt bietet die Funktionen `createEntity(...)` und `createDependent(...)` zur Anlage von Geschäftsobjekten an. Diese Methoden benötigen unter anderem einen projektspezifischen Namen der Klasse von der eine Instanz angelegt werden soll, das sogenannte `ClassToken`. Wo das Geschäftsobjekt tatsächlich erzeugt wird, hängt von Einstellungen ab. Die Dienste der Fundamentschicht ermitteln alle zur Erzeugung des Objekts zusätzlich notwendigen Informationen und legen das Objekt an. Anhand des `ClassTokens` werden die Java-Klassen ermittelt, die zur Erzeugung des Objekts notwendig sind. Diese sind so im laufenden Betrieb austauschbar. Als Ergebnis wird ein SanFrancisco Objekt-Handel (Handel) zurückgegeben. Über dieses Handel lässt sich das Objekt zu jeder Zeit eindeutig ermitteln. Es folgt ein Kodebeispiel zur Anlage einer Person:

```
boolean finished = false;
Person person;

// Look for a special factory to create a person
PersonFactory factory = (PersonFactory)Global.factory().
                getSpecialFactory("prototypes.Person");

// Use the baseFactory if there is no special factory
 if (factory == null) {
    person = (Person)Global.factory().createEntity
        ("prototypes.Person", access, locationHandle, null); // no key

    // initialize the person and cleanup if it fails to initialize
    try {
       person.initialize(name,adress,telefon,email);
       finished = true;
```

Dies ist das `ClassToken`

```
    } finally {
       if (!finished) {
          person.uninitialize();
          Global.factory().deleteEntity(person);
       }
    }

 // Otherwise use the special factory to create the person
 } else {
    person = factory.create(access, locationHandle,
                                          name,adress,telefon,email);
 }

 return person;
```

5.1.2 Transaktionen und Sperrmechanismen

Um mehreren Benutzern gleichzeitig Zugriff auf das selbe Geschäftsobjekt zu gewähren, bietet die Fundamentschicht Transaktionen und Sperrmechanismen an. Transaktionen gewährleisten, dass Datenänderungen an einem oder mehreren Objekten vollständig oder gar nicht durchgeführt werden. Sperren stellen sicher, dass Objekte, die ein Benutzer ändert, nicht gleichzeitig von anderen Benutzern verändert werden können. Beide Konzepte greifen ineinander.

Objekte können optimistisch oder pessimistisch gesperrt werden. Wird ein Objekt mit einer pessimistischen Sperre belegt, kann kein anderer Benutzer dieses Objekt verändern oder ansehen. Optimistische Sperren lassen das Lesen von Objekten zu, auch wenn andere Benutzer dieses bereits verwenden und gesperrt haben. Wollen mehrere Benutzer das selbe Objekt verändern und ist es optimistisch gesperrt, wird nur die Änderung des ersten Benutzers durchgeführt. Für alle anderen muss eine Fehlerbehandlung durchgeführt werden. Es ist also abzuwägen, wie häufig dieser Fall auftritt, um dann die geeignete Sperrstrategie zu verwenden. Für den SanFrancisco Entwickler sind die verschiedenen Konzepte transparent nutzbar.

5.1.3 Objektzugehörigkeit (*Ownership*)

Abhängigkeiten zwischen Geschäftsobjekten werden in SanFrancisco entweder durch Zuordnung oder durch Zugehörigkeit umgesetzt. Der wesentliche Unterschied ist, dass einem besitzenden Objekt zugehörige Objekte gelöscht werden, falls der Besitzer gelöscht wird. Das verwundert den objektorientierten Softwareentwickler nicht weiter, werden doch alle Objekte, wenn es keine Referenz mehr auf sie gibt, entfernt. Hier handelt es sich aber um persistente Objekte, die auch im persistenten Speicher entfernt werden müssen. Daher bietet SanFrancisco alle *Collection* Typen (`Set`, `List`, usw.) auch als sogenannte *Owned Collections* (`EntityOwningSet`, `EntityOwningList`, usw.) an, die genau diese Aufgabe übernehmen.

5.1.4 Benachrichtigungsdienst (*Notification Service*)

SanFrancisco kennt zwei Arten der Umsetzung des Beobachter (*observer/observable*) Paradigmas. Das Objekt, das ein anderes beobachtet, erhält zu bestimmten Anlässen Benachrichtigungen. Die Objekte müssen dazu entweder das Java eigene `java.util.Observer` Interface implementieren oder das `Entity` Interface von SanFrancisco. Die SanFrancisco Variante ermöglicht eine transaktionale, entfernte und persistente Überwachung.

5.1.5 Abfrage von Businessobjekten (*Querying Collections*)

Persistente Geschäftsobjekte können anhand von `QueryCommands` abgefragt werden. Diese `Commands` operieren auf *Collections* und liefern als Ergebnis transiente Teilmengen dieser *Collections*. Um Abfragen zu formulieren, ist eine Object Query Language vorhanden. Diese ist an die SQL92 angelehnt und so erweitert, das sich Objekte abfragen lassen.

5.1.6 Sichere Abschnitte (*Secure Section*)

SanFrancisco unterstütz zwei Sicherheitsmechanismen. Benutzer müssen sich authentifizieren und sind zu bestimmten Aktionen autorisiert. Sichere Abschnitte sind eine wesentliche Funktionalität, Autorisationen festzulegen. Bei der Entwicklung werden dazu die Sicheren Abschnitte benannt und angelegt. Der Administrator kann Benutzer berechtigen, diese benannten Abschnitte ausführen zu dürfen oder nicht.

6 Geschäftslogik von SanFrancisco

Das vorherige Kapitel ist auf die Schichtung von SanFrancisco eingegangen. Dort wurde die Fundamentschicht und dort angesiedelten Konzepte und Funktionalitäten vorgestellt. Die Geschäftslogik von SanFrancisco wird in diesem Kapitel vorgestellt. Sie ist in zwei Schichten gegliedert, in die allgemeine Geschäftsobjektschicht (*Common Business Objects Layer*) und in die Kerngeschäftsprozessschicht (*Core Business Processes*), die auf der Fundamentschicht aufbauen (siehe Abbildung 5).

Dieses Kapitel klärt zunächst die prinzipielle Ausrichtung der SanFrancisco Komponenten. Sind sie prozessorientiert oder funktional ? Werden Prozessablaufe für die gesamte Organisation festgelegt oder kann der Entwickler sie beeinflussen ? Die nachfolgenden zwei Abschnitte beschreiben die jeweiligen Funktionalitäten der beiden Schichten. Daraus lässt sich die Güte und der Umfang der mitgelieferten Geschäftslogik ableiten. Das Kapitel endet mit der Betrachtung der Einsetzbarkeit der Geschäftslogik. Die Verwendbarkeit der Funktionalität ist wesentlich, um die Softwareentwicklung zu beschleunigen bzw. vereinfachen.

6.1 Organisationsleitbild von SanFrancisco

SanFrancisco liefert bereits eine Menge an Geschäftsvorfälle orientierte Objekte und Funktionalität mit. Entscheidend für den erfolgreichen Einsatz ist, welches Organisationsleitbild diesen zugrunde liegt (vgl. 2.1, S. 5).

SanFrancisco Geschäftslogik ist nur als Rahmen vorhanden. Alle wichtigen Geschäftsobjekte, aber auch deren Beziehungen, sind nur vorgedacht. So gibt es natürlich das Geschäftsobjekt Bestellung, das in Beziehung zur Lieferung steht. Wie die Objekte genau ausgeprägt sind, ob also ein Rabat oder Skonto vergeben wird, bleibt dem Entwickler überlassen. Klar ist, dass der Wert der Bestellung nach der Lieferung als offener Posten an die Finanzbuchhaltung weitergegeben wird. Solche allgemeinen Objekte und Vorgänge sind bereits vorhanden, nicht aber die konkrete Ausprägung. Es ist nicht vorgeschrieben, ob die Funktionalitäten als Aufgaben einer Prozesskette oder als Einzeltätigkeiten im Arbeitskontext von Mitarbeitern angesehen werden.

SanFrancisco ist kein Workflow-Management-System. Es bietet keine Mechanismen zur Definition von Prozessen zur Laufzeit an. Ebenso gibt es keine Funktionalität, den Prozessfortschritt zu überwachen. Gleichwohl können Prozessabläufe in Form von `Command`-Objekten kodiert werden. Darüber hinaus werden Mechanismen zur Nachrichtenübermittlung angeboten. Zusammen können so Prozessabläufe teilautomatisiert werden.

Das in der Praxis vorherrschende Leitbild der ablaufgesteuerten Unternehmens-organisation kann in der SanFrancisco Literatur ebenfalls deutlich wiedergefunden werden. Die Geschäftsobjekte von SanFrancisco bieten Methoden an, die in Form von `Command`-Objekten zu vordefinierten Abläufen zusammengestellt werden sollen. Die

Mitarbeiter sollen nicht die Gestaltungsverantwortung für die Unternehmensprozesse erhalten (vgl. [IbmCF99]).

Da die konkrete Ausgestaltung der zu entwickelnden Geschäftsanwendung einem vollständigen Softwareprozess unterliegt, kann dieses Leitbild durchbrochen werden und die Unterstützung des qualifizierten Sacharbeiters umgesetzt werden. Prozessabläufe können in verschiedenen Varianten angeboten werden, also in Standardfälle und Spezialfälle unterschieden werden. Dazu muss jeweils ein `Command`-Objekt implementiert werden, das durch eine Benutzerinteraktion angestoßen wird.

In welchem Umfang die allgemeinen Geschäftsobjektschicht und die Kernprozess-schicht bereits nötige Funktionalitäten abdecken, wird jetzt aufgezeigt.

6.2 Die Allgemeine Geschäftsobjektschicht (*Common Business Objects Layer*)

Sowohl die allgemeine Geschäftsobjektschicht (*Common Business Layer CBO*) als auch die Kernprozessschicht enthalten SanFrancisco Geschäftsobjekte. Sie unterscheiden sich nicht prinzipiell, sondern konzeptionell. Die allgemeine Geschäftsobjektschicht beinhaltet domänenübergreifende Geschäftsobjekte, die Kerngeschäftsprozessschicht domänenspezifische Geschäftsobjekte und Abläufe von Geschäftsvorfällen.

In der Allgemeinen Geschäftsobjektschicht sind drei verschiedene Gruppen von Objekten auszumachen:

- Allgemeine Geschäftsobjekte, die von mehreren Anwendungsdomänen genutzt werden,
- Objekte, die Funktionalitäten verschiedener Anwendungsdomänen anbieten und
- Objekte, die buchhalterische Belange außerhalb der Buchhaltungsdomäne umsetzten. Dieses Teilframework wird allgemeines Finanzinterface (*Common Function Financial Interface*) genannt.

Die erste Gruppe von Objekten enthält eine Reihe von Geschäftsobjekten, die mehreren Anwendungsdomänen gemein sind. Dazu zählen zum Beispiel die Firma (*Company*) und der Businesspartner (*BusinessPartner*), aber auch Businessobjekte wie Währung (*Currency*), Nummerierungen (*NumberSeries*) und Maßeinheiten (*UnitOfMeasure*).

Anwendungsdomänen gemeinsame Funktionen bilden die zweite Gruppe. Diese Funktionen sind sehr allgemein und flexibel gehalten, um verschiedene Probleme lösen zu helfen. Es sind kodierte Pattern, wie IBM es nennt. Ein Beispiel für ein solches Pattern sind gecachte Saldos (*Cached Balances*). Anstatt bei jeder Anfrage ein Saldo anhand aller Umsätze zu errechnen, kann der Saldo kontinuierlich fortgeschrieben und gespeichert werden. Eine solches Aufsummieren kann in verschiedenen Kontexten deutlich komplexer sein. Allen solchen Saldos gemeinsame Funktionen sind bereits vorhanden.

Die dritte Gruppe von Objekten ermöglicht die Manipulation buchhalterischer Daten ohne Kenntnis der einzelnen Prozesse und Geschäftsobjekte der Hauptbuchhaltung. Wird zum Beispiel Ware verkauft, muss die Lagermenge aktualisiert werden und bei

Zahlungseingang die Geldmenge des Geschäftskontos erhöht werden. Dazu ist es nicht notwendig, die Hauptbuchhaltung direkt anzusprechen. Diese muss nicht einmal eingesetzt werden. Die Allgemeine Geschäftsobjektschicht bietet spezielle Schnittstellen und Geschäftsobjekte (z.B.: *BankAccount*) an, um Daten an die Hauptbuchhaltung zu übergeben.

6.3 Die Kerngeschäftsprozessschicht (*Core Business Processes*)

Die Kerngeschäftsprozessschicht bietet Grundgerüste verschiedener Anwendungsdomänen an. Namentlich sind das in der Version 1.3 die Hauptbuchhaltung, die Kostenrechnung, die Lagerverwaltung und die Auftragsabwicklung. Zu jeder Domäne werden alle als wichtig erkannten Geschäftsobjekte und die Hauptprozesse als Rahmenwerk angeboten. Diese müssen nach Bedarf und Anforderung erweitert werden. Die Funktionalitäten der einzelnen Domänen können miteinander interagieren, bedingen sich aber nicht. Es ist also möglich, die Lagerverwaltung ohne die Hauptbuchhaltung einzusetzen. Jeder dieser Domänen ist in verschiedene Kategorien unterteilt, die thematisch gemeinsame Prozesse und Geschäftsobjekte gruppieren.

Die folgenden Abschnitte stellen die vier Domänen vor, insbesondere den Umfang der bereits vorhanden Objekte und Funktionalitäten.

6.3.1 Hauptbuchhaltung *(General Ledger)*

Ziel der Hauptbuchhaltung ist es, den Geldfluss eines Unternehmens transparent zu machen. Es müssen gesetzliche Anforderungen genauso wie unternehmerische Belange erfüllt werden. Da gerade gesetzliche Anforderungen sich von Land zu Land unterscheiden, kann SanFrancisco keine allgemeingültige sofort einsetzbare Lösung anbieten. Es wird ein Vorgehen vorgeschlagen, das durch viele Objekte und Funktionalitäten unterstützt wird. Bevor die Hauptbuchhaltung eingesetzt werden kann, müssen folgende drei Bereiche analysiert werden. Es müssen

- der Kontenplan,
- täglichen Aktivitäten und
- periodischen Aktivitäten

bestimmt und festgelegt werden (siehe [IbmCF99]).

Für jeden Bereich bietet SanFrancisco eine Menge Objekte und Funktionalitäten an. Diese sind, wie für alle Domänen üblich, in Kategorien aufgeteilt. Die Hauptbuchhaltung besteht aus folgenden Kategorien:

Kontenplan (*Posting Combinations*): Diese Kategorie bietet Geschäftsobjekte an, um einen Kontenplan aufzustellen. Der Kontenplan enthält verschiedene Kontenarten, Prüfbedingungen für Buchungen und verschiedene Konten.

Journal: Das Journal erlaubt die Anlage, die Weiterführung und den Abschluss aller Buchungen.

Bilanz (*Balance*): Diese Kategorie gruppiert alle Objekte und Funktionalitäten, um Auswertungen zu definieren. Es können Auswertungen über Kontengruppen, in verschiedenen Perioden oder Währungen angelegt werden.

Budget (*Budgets*): Neben der Auswertung der Einnahmen und Ausgaben ist ebenso eine Budgetierung vorgesehen.

Kontenabschluss (*Closing*): Grundlegende Funktionalitäten für periodische Abschlüsse sind vorhanden.

Bank: Diese Kategorie enthält Funktionalitäten, um die Bankgeschäfte zu überwachen. Der Abgleich von Zahlungsaus- und –eingängen ist vorhanden.

Neubewertung (*Revaluation*): Funktionalitäten zur Neubewertung von Währungsbeständen, die sich von der Währung der Hauptbuchhaltung unterscheiden, ist bereits vorhanden.

Der Einsatz und die Kombination von Geschäftsobjekten und Funktionalitäten dieser Kategorien sollte das Führen einer Hauptbuchhaltung ermöglichen.

6.3.2 Kostenrechnung (*Accounts receivable and payable*)

Die Kostenrechnung veranschaulicht alle Transaktionen eines Unternehmens mit seinen Geschäftspartnern. Hier werden die Saldos der Debitoren und Kreditoren geführt. Geht Ware in der Warenannahme ein, kann die beiliegende Rechnung sofort ins System eingetragen werden. Die Zahlung erfolgt aber erst, wenn der zuständige Einkäufer die Rechnung geprüft hat. Die Kostenrechnung verwaltet dazu Buchungen und Erfassung, die zu Buchungen werden können.

Wie für die Hauptbuchhaltung, müssen für die Kostenrechnung die Basisinformationen, tägliche und periodische Aktivitäten bestimmt und angelegt werden. Die Kostenrechnung gliedert sich in folgende Kategorien:

Kontentypen (*Ledger Types*): Eine Kostenrechnung unterscheidet Debitoren und Kreditoren. Die dazu nötigen Objekte finden sich in dieser Kategorie. Darüber hinaus können beliebige andere Kontentypen angelegt werden, z.B. für Mitarbeiter.

Konten: Hier finden sich die Geschäftsobjekte, die Konten repräsentieren.

Partner: Die Geschäftsobjekte dieser Kategorie repräsentieren die Geschäftspartner eines Kontos.

Bankkonto: Diese Kategorie erweitert das Bankkonto der allgemeinen Geschäftsobjektschicht um Funktionalitäten zur Zahlungsabwicklung.

Zahlungsmethoden: Hier finden sich Geschäftsobjekte, die verschiedene Zahlungsmethoden repräsentieren.

Erfassungen (*Log Items*): Diese Kategorie bietet Geschäftsobjekte zur Erfassung zukünftiger Buchungen an. Es gibt Objekte, um Rechnung, Gutschriften oder Zahlungen abzubilden. Alle zur Buchung notwendigen Daten können angegeben werden. Eine solche Erfassung kann, nachdem sie genehmigt wurde, in eine Buchung überführt werden.

Buchungen: Buchungen können entweder direkt oder aus Erfassungen erzeugt werden.

Ratenzahlung: Die Geschäftsobjekte dieser Kategorie erlauben, Zahlungspläne für Erfassungen oder Buchungen anzugeben. Jedem Zahlungsplan können eigene Nachlässe und Zahlungszeitpunkte zugeordnet werden.

Zuteilung (*Allocation*): Diese Kategorie enthält Funktionalitäten, um getätigte Finanzaktivitäten den zugehörigen Buchungen zuzuordnen. Der Abgleich findet anhand der Zahlungspläne statt. Teilzahlungen können auch ohne passenden Zahlungsplan zugeordnet werden.

Zahlungsanweisung: Diese Kategorie dient dazu, Zahlungen zu verwalten. Die nötigen Konten- und Bankbewegungen werden dann automatisch erzeugt.

Wie anhand der Kategorien zu erkennen ist, bietet die Kostenrechnung ebenfalls einen umfangreichen Fundus nötiger Geschäftsobjekte an.

6.3.3 Lagerverwaltung *(Warehouse Management)*

SanFrancisco enthält einen Rahmen zur Umsetzung einer funktionsreichen Lagerverwaltung. Zentrale Geschäftsobjekte dieser Domäne sind Produkte und Lager. Der Umfang der Funktionalitäten soll anhand der Kategorien der Domäne veranschaulicht werden. Die Lagerverwaltung gliedert sich in die Kategorien:

Transportinformationen: Diese Kategorie bietet Funktionalitäten zur Verwaltung von Transportzeiten anhand von Adressen, von Transporteuren und den Produkten an.

Rechnungswesen: Objekte und Funktionalitäten dieser Kategorie ermöglichen die Integration mit der Hauptbuchhaltung. Alle Lageraktivitäten können einfach an die Finanzbuchhaltung weitergegeben werden.

Produkt: Alle wichtigen Geschäftsobjekte zur Verwaltung komplexer Produkte sind vorhanden. Es können Gebinde- und Losgrößen, Lagerorte und Seriennummern verwaltet werden. Zusätzlich beinhaltet diese Kategorie alle Funktionalitäten zur Produktverwaltung, wie Verfügbarkeitsprüfungen, Vorausplanung, Kosten usw.

Nachbestellung: Funktionalitäten zur Nachbestellung von Produkten, finden sich hier. Eine Nachbestellung kann bei Lieferanten oder anderen Lagern erfolgen.

Inventur: Alle für eine Inventur nötigen Funktionalitäten sind vorhanden. Die Mehr- bzw. Fehlbestände können an die Buchhaltung weitergegeben werden.

Bestandstransaktionen: Alle Lagerbewegungen werden als Transaktion verwaltet. Diese können manuell oder automatisch ausgelöst werden.

Warenausgang: Die Geschäftsobjekte dieser Kategorie ermöglichen die Auswahl der Produkte, die versandt werden sollen. Die Auswahl wird durch betriebliche oder monetäre Strategien unterstützt.

Warenannahme: Die zur Annahme von Ware nötigen Geschäftsobjekte sind in dieser Kategorie zusammengefasst.

Planung: Geschäftsobjekte zur Planung welche Produkte in welchen Lagern in welcher Menge vorgehalten werden, finden sich in dieser Kategorie.

Qualitätskontrolle: Geschäftsobjekte zur Verarbeitung der Ergebnisse der Qualitätskontrolle sind vorhanden.

Versand: Der Versand von Produkten kann anhand von Objekten dieser Kategorie überwacht werden.

Lieferrückstand: Der Umgang mit Lieferengpässen ist ebenfalls bereits in einer Kategorie vorgedacht.

Die Geschäftsobjekte und Funktionalitäten dieser Kategorien decken alle wesentlichen Bereiche der Lagerhaltung bereits ab. Die zentralen Lager- und Versandstrategien fehlen natürlich, sind aber als Rumpf vorhanden und müssen erweitert werden.

6.3.4 Auftragsabwicklung (*Order Management*)

Die Auftragsabwicklung von SanFrancisco ist eng mit den Strukturen der Lagerverwaltung verknüpft, ohne diese jedoch vorauszusetzen. Die Integration erfolgt über Geschäftsobjekte der allgemeinen Geschäftsobjektschicht. Die wichtigsten Geschäftsobjekte der Auftragsabwicklung neben dem Auftrag selbst sind Produkte und Partner. Die Geschäftsobjekte sind in folgende Kategorien gruppiert:

Produkt: Die Geschäftsobjekte dieser Kategorie erweitern die Möglichkeiten der Lagerhaltung. Es kann festgelegt werden, ob ein Produkt eingekauft oder verkauft werden darf. Falls Produkte nicht vorhanden sind oder nicht bestellt werden können, werden Substitutionen vorgeschlagen. Für den Einkauf und den Verkauf können unterschiedliche Gebinde erlaubt sein.

Partner: Diese Kategorie bietet Geschäftsobjekte an, um Lieferanten und Kunden zu repräsentieren.

Preispolitik und Rabatte: Die Geschäftsobjekte ermöglichen eine äußerst flexible Preisgestaltung eines Auftrags und dazugehöriger Produkte. Die Ermittlung des Preises ist ebenso wie der Rabatt an die Bedingungen des jeweiligen Auftrags gebunden. Sie können vom Auftraggeber, vom Produkt und vom Auftragsvolumen abhängen.

Lieferverzug: Die Kategorie Lieferverzug unterscheidet Verzug von Lieferanten und internen Verzug aus eigenen Lagern.

Kontobewegung: Die buchhalterischen Auswirkungen eines Auftrags können mittels der Funktionalitäten dieser Kategorie an die Buchhaltung übertragen werden.

Auftragsverfolgung: Diese Kategorie enthält alle Geschäftsobjekte, um den Zyklus eines Auftrags zu verfolgen. Dabei gibt es Möglichkeiten, die Korrektheit, die Vollständigkeit und Integrität zu prüfen.

Planung: Die Geschäftsobjekte dieser Kategorie ermöglichen die Planung und Umplanung des Versands. Es kann bestimmt werden, aus welchem Lager das Produkt entnommen oder eingelagert wird.

Produktauswahl: Diese Kategorie bietet Funktionalitäten zur Produktauswahl an.

Warenannahme: Diese Kategorie ist mit der gleichnamigen der Lagerverwaltung integriert.

Qualitätskontrolle: Die Funktionalitäten zur Qualitätskontrolle entsprechen denen der Lagerhaltung.

Versand: Diese Kategorie beinhaltet Funktionalitäten zur Unterstützung verschiedener Versandwege.

Auftragsbestätigung: Zur Bestätigung von Auftragen bietet diese Kategorie alle nötigen Objekte an.

Rechnung: Rechnungen zugehöriger Aufträge können anhand von Funktionalitäten dieser Kategorie erstellt werden. Die Weiterleitung an die Buchhaltung ist ebenfalls vorgesehen.

Die Funktionalitäten obiger Kategorien sollte die Entwicklung einer leistungsfähigen Auftragsverwaltung möglich machen. Viele Sonderfälle sind in Form von Funktionalitäten und Geschäftsobjekten bereits vorhanden.

6.4 Verwendbarkeit der vorhandenen Geschäftslogik

Die Fülle der Geschäftsobjekte zu betrachten ist nicht ausreichend, um deren Verwendbarkeit für reale Softwareprojekte einzuschätzen. In zwei Konferenzbeiträgen haben Schmitzer und Ließmann die Hauptbuchhaltung *(General Ledger*) und die Auftragsabwicklung (*Order Management*) von SanFrancisco auf deren Einsetzbarkeit hin untersucht (vgl. [SchmiLi99] und [SchmiLi00]). Die dort beschriebenen Erfahrungen werden hier zusammenfassend wiedergegeben.

Der Ausgangspunkt zur Untersuchung der Hauptbuchhaltung von SanFrancisco waren Bedürfnisse eines Unternehmens des produzierenden Gewerbes. Dabei mussten die rechtlichen Bestimmungen des HGB eingehalten werden. Als Vorbild für den tatsächlichen Kontenrahmen wurde der Industriekontenrahmen (IKR) des Bundesverbandes der Deutschen Industrie (BDI) herangezogen und verwirklicht. Schmitzer und Ließmann kommen zu dem Schluss, dass das SanFrancisco Rahmenwerk Anwendungsentwicklern flexible Methoden zur Verfügung stellt, mit denen sich Kontenpläne schnell realisieren lassen (vgl. [SchmiLi99]).

In einem weiteren Beitrag untersuchten sie die Auftragsabwicklung von SanFrancisco und verglichen die Funktionalitäten mit in betriebswirtschaftlicher Literatur beschriebenen Anforderungen. Herangezogen wurde der Prozess Auftragsabwicklung, der in Auftragsannahme, Auftragsprüfung und Versand unterteilt wird. In ihrer Beurteilung halten sie fest, dass die Auftragsabwicklung von SanFrancisco alle wesentlichen Funktionen abdeckt. Sie weisen ausdrücklich auf die Flexibilität von SanFrancisco hin. Negativ merken sie die deutliche Ausrichtung der Funktionalität auf

Unternehmen an, die auf Vorrat Standarderzeugnisse ohne Varianten herstellen. Sie halten SanFrancisco geeignet für Softwareunternehmen mit gutem Branchenwissen, aber weniger geeignet für eine schnelle Unternehmensinterne Lösung. Dort raten sie zu Black-Box Rahmenwerken, wie z.B. von SAP oder Baan (vgl. [SchmiLi00]).

Über die Kostenrechnung und die Lagerverwaltung finden sich keine Erfahrungsberichte dieser Art. Sollte deren Einsetzbarkeit ähnlich zu den beiden beschriebenen sein, bietet SanFrancisco eine ganze Menge verwertbarer Funktionalitäten an.

7 Vorgehen zur Entwicklung SanFrancisco basierter Software

Wie in Kapitel 3 erwähnt, hat IBM als Hauptziel die optimale Unterstützung des Softwareentwicklungsprozess durch SanFrancisco benannt. So verwundert es nicht, dass IBM ein konkretes Vorgehen zur Umsetzung von Softwareprojekten mit SanFrancisco erarbeitet hat, das *SanFrancisco Application Development Roadmap*. Das Vorgehen und die beteiligten Werkzeuge werden in diesem Kapitel vorgestellt. Zunächst werden die einzelnen Phasen des *Roadmaps* erläutert, um dann den Werkzeugansatz von IBM vorzustellen. Abschließend wird exemplarisch der SFBuilder vorgestellt.

Das *SanFrancisco Application Development Roadmap* ist eine Abfolge von Aufgaben, um geplant und strukturiert Softwareprojekte zu realisieren. IBM unterstützt die Aufgaben durch Checklisten, Dokumentvorlagen, Beispiele und Tools.

Das *Roadmap* stellt verschiedene Aspekte der Softwareentwicklung zusammen. Außer dem Vorgehen zur Ermittlung der Anforderungen, dem Design und der Kodierung der Software werden auch Maßnahmen zur Qualitätssicherung und generelle Überlegungen zum Projektmanagement beschrieben. Diese beiden Aspekte eines Softwareprojekts sollen hier nicht genauer betrachtet werden. Nur soviel, die Maßnahmen zur Qualitätssicherung werden ebenfalls durch Checklisten und eine Reihe von Anregungen unterstützt. Die Maßnahmen können dazu beitragen, die Softwarequalität zu erhöhen. Das Projektmanagement definiert verschiedene Rollen der beteiligten Personen und weist ihnen verschiedene Verantwortlichkeiten zu. Es werden verschiedene Arten beschrieben, ein Softwareprojekt zu planen und vor allem diese Planung dann auch zu verfolgen.

Die nächsten Abschnitte beschreiben die einzelnen Aufgaben des *SanFrancisco Application Development Roadmaps*.

7.1 Das SanFrancisco *Application Development Roadmap*

Das *SanFrancisco Application Development Roadmap* ist IBMs Vorschlag zur Realisierung SanFrancisco basierter Geschäftsanwendungen. Der Softwareprozess gliedert sich in sechs verschiedene Phasen (siehe Abbildung 7). Diese Phasen können mehrmals durchlaufen werden. IBM empfiehlt ein solches evolutionäres Vorgehen ausdrücklich, um dadurch Lerneffekte aller Beteiligten zu nutzen. Es ist nicht gemeint, die gleichen Anforderungen mehrmals zu ermitteln, zu analysieren und zu realisieren, sondern komplexe Anforderungen in kleine überschaubare Teile zu gliedern und diese nacheinander durch die sechs Phasen zu verwirklichen.

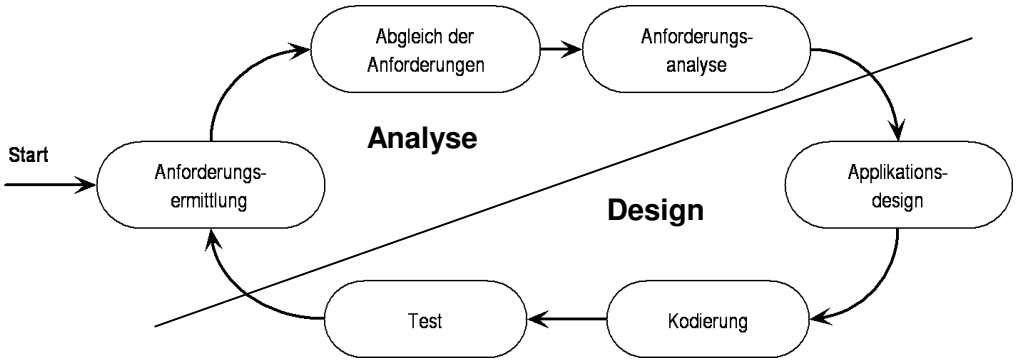

Abbildung 7: Das *SanFrancisco Roadmap* zur Entwicklung von Geschäftsanwendungen

Jede der sechs Phasen ist detailliert beschrieben und gliedert sich in verschiedene Arbeitsschritte. Durch vorgefertigte Dokumente, Tabellen und Checklisten wird eine gute Vorstellung möglicher Ergebnisse der Arbeitsschritte vermittelt. Das Vorgehen, die einzelnen Phasen und die Arbeitsschritte sind jeweils keine von IBM erdachten Methoden, sondern eine Zusammenstellung gängiger Methoden verschiedener Softwareprozesse. Das *Roadmap* ist mit anderen objektorientierten Vorgehensmodellen vergleichbar. Einige Arbeitsschritte werden durch die SanFrancisco Werkzeugstrategie optimal unterstützt.

7.1.1 Anforderungsermittlung

Ziel der Anforderungsermittlung ist die Dokumentation der Anforderungen der zu entwickelnden Geschäftsanwendung. Dazu werden verschiedene Methoden beschrieben, zum einen die Prozessmodellierung (*Process Modeling Technique*), zum anderen die Anwendungsfallanalyse (*Use Cases*) von Ivar Jacobson. Für welche der beiden Methoden man sich entscheidet, bleibt dem Entwicklungsteam überlassen.

Beiden Vorgehensweisen gemeinsam ist die Art und Weise, Informationen über Geschäftsprozesse zu sammeln. Sie werden aber unterschiedlich festgehalten. Bei der Prozessmodellierung liegt der Fokus auf der Ermittlung sogenannter *Tasks*, die dann zu Prozessen zusammengeführt werden. Die Anwendungsfallanalyse ermittelt alle für das System wichtigen Akteure und deren Nutzungskontexte.

Ergebnis der Anforderungsermittlung ist eine erste Beschreibung und Übersicht aller zu implementierenden Prozesse.

In dieser Phase wird außerdem vorgeschlagen, bereits technische Anforderungen und Anforderungen zur Einführung der Software festzuhalten.

7.1.2 Abgleich der Anforderungen

Die durch die Anforderungsermittlung gewonnen Anforderungen werden zu Applikations-Szenarios (*Application Scenario*) erweitert. Die Hauptaufgabe des Abgleichs ist festzustellen, inwieweit SanFrancisco Funktionalitäten der ermittelten Szenarios bereits abdeckt. Das Ergebnis ist ein Verständnis davon, welche SanFrancisco Funktionalitäten wie erweitert werden müssen.

7.1.3 Anforderungsanalyse

Die Anforderungsanalyse gruppiert mehrere Aufgaben. Ziel ist, die Applikations-Szenarios zu detaillieren um daraus ein erstes Klassendiagramm zu erarbeiten. Außerdem wird eine erste Benutzungsschnittstelle konzipiert, die als Prototyp den späteren Anwendern vorgestellt werden kann.

Die Applikations-Szenarios werden durch eine Abfolge von Aktionen beschrieben, die bei der Ausführung des Szenarios durchgeführt werden. Der Fokus liegt darauf, Eingaben, Annahmen, Konsequenzen und Ausgaben für die Aktionen zu ermitteln. Die Applikations-Szenarios sollen möglichst klein sein.

Als nächstes wird die Benutzungsschnittstelle zur Durchführung der Applikations-Szenarios entworfen. Für stark interaktive Szenarios wird empfohlen, Prototypen der Benutzungsschnittstelle zu entwickeln und diese durch Endanwender beurteilen zu lassen.

Um ein Klassendiagramm aus Analysesicht zu erstellen, müssen alle Applikations-Szenarios auf ihre Verantwortlichkeiten untersucht werden. Daraus ergeben sich Verantwortlichkeiten einzelner Klassen. SanFrancisco Framework-Klassen können diese vollständig, teilweise oder gar nicht erfüllen. Im ersten Fall ist nichts zu tun. Gibt es nur teilweise Überschneidungen der Verantwortlichkeiten zwischen Klasse und Applikations-Szenario, wird eine neue Klasse angelegt, die von der bestehenden abgeleitet wird. Wurden neue Verantwortlichkeiten ermittelt, ist eine neue Klasse zu erzeugen.

Werden neue Klassen angelegt, müssen die Verantwortlichkeiten dokumentiert werden. IBM schlägt dazu CRC* (*Class Responsibility Collaborator*) Karten vor. Anhand der CRC-Karten lässt sich ein Klassendiagramm erstellen, welches noch keine Design-Elemente wie *getter*- und *setter*-Methoden oder User-Interface Klassen enthalten soll.

In dieser Phase können bereits erste Objekt-Interaktionsdiagramme erstellt werden. In der Analysephase ist das optional, in der Designphase aber verpflichtend. Alle nicht trivialen Aktionsabfolgen müssen dann dargestellt sein. Welche Technik man dabei wählt, bleibt dem Entwickler überlassen.

Die Analysephase wird durch den Abgleich des Klassendiagramms mit dem SanFrancisco Pattern Katalog abgeschlossen. Ziel ist die strikte Anwendung der definierten Pattern in allen Bereichen. Danach ist das endgültige Klassendiagramm der Analysephase zu erstellen.

Am Ende der Analysephase ergibt sich ein detailliertes Verständnis der umzusetzenden Geschäftsprozesse und aller zu entwickelnden Klassen, ohne jedoch Kode geschrieben zu haben.

---

* „CRC-Karten sind Karteikarten, auf denen die Klasse, ihre Verantwortlichkeiten und andere Beteiligte notiert werden. [...] Eine Klasse ist verantwortlich für das Wissen, über das ihre Objekte verfügen sollen (Attribute) und für die Operationen, die ihre Objekte ausführen müssen, um ihre Aufgaben und Ziele zu erreichen. Beteiligte sind andere Klassen, zu denen Beziehungen existieren müssen, damit die Klasse ihre Aufgaben erfüllen kann." [Oest97]

7.1.4 Anwendungsdesign

Die Designphase überführt die Ergebnisse der Analyse strukturiert in Applikationskode. Hier sind wiederum einige Einzelaufgaben durchzuführen.

Die Applikations-Szenarios müssen um Fehlerfälle und Fehlerbehandlung, aber auch um konkrete Datenstrukturen ergänzt werden. Hier können erste Verweise auf vorhandene Datenstrukturen, z.B. aus Legacy Datenbanken, ermittelt und dokumentiert werden.

Die Benutzungsschnittstelle muss entworfen werden, falls das in der Analysephase nicht schon geschehen ist. Die Benutzungsschnittstelle ist mit vorhandenen Standards und *Guidelines* abzugleichen. Sie sollte späteren Nutzern vorgestellt werden, um ihnen Gestaltungsmöglichkeiten zu geben.

Das Klassendiagramm der Analysephase muss um alle Elemente der Designphase ergänzt werden. Im wesentlichen sind das alle zusätzlichen Klassen, die erforderlich sind, um das SanFrancisco *Programming Model* zu erfüllen, also solche wie die Implementationsklasse, die Interfaceklasse, die Factoryklasse und die Stub- und Skeletonklasse. Außerdem sind alle Klassen der Benutzungsschnittstelle zu ergänzen.

Für komplexe Klassen oder Applikations-Szenarios sollten Objekt-Interaktionsdiagramme erstellt werden. Die Objekt-Interaktionsdiagramme der Analysephase müssen erweitert werden. Vor allem die Fehlerbehandlung ist einzufügen, falls nicht schon vorhanden. Ebenfalls müssen alle Designentscheidungen berücksichtigt werden. Wurde sich dafür entschieden eine kontinuierliche Summe mitzuführen, so ist die erforderliche Summe nicht mehr aus allen Summanden zu errechnen, sondern bei jeder Änderung zu aktualisieren. Diese Änderung muss auch im Objekt-Interaktionsdiagramm nachvollzogen werden.

Ist das Klassendiagramm und damit das Klassenmodell der Designphase fertig, müssen die Datenstrukturen der Objekte auf relationale Datenstrukturen abgebildet werden (siehe Kapitel 4.2.1). Falls Datenstrukturen anderer Softwaresysteme bereits bestehen und weiterverwendet werden sollen, müssen sie berücksichtigt werden.

Ist das Anwendungsdesign beendet, ist der Koderahmen für die Applikation fertig und muss noch um die eigentliche Programmlogik erweitert werden, was in der nächsten Phase geschieht.

7.1.5 Kodierung

Die durch das Anwendungsdesign modellierten Klassen und Interaktionen werden durch die Kodierung maschinenlesbar umgesetzt. Zur schnelleren Umsetzung ist eine automatische Kodegenerierung hilfreich.

Der entwickelte Kode ist durch einfache sogenannte Feature-Tests auf Fehlerfreiheit und Erwartungskonformität zu prüfen. Die Ergebnisse dieser Tests sollten in Testplänen dokumentiert werden.

7.1.6 Testen

Die Testphase ist Teil der Qualitätssicherungsmaßnahmen und ist nicht Teil des eigentlichen Entwicklungs-*Roadmaps*. Da das Testen bei dem Vorgehen der IBM einen

sehr hohen Stellenwert einnimmt, wird das Testen hier als Abschluss des *Roadmaps* in dieses integriert.

Bei der Kodierung werden bereits erste Tests durchgeführt. Diese einfachen Tests werden jetzt zu sogenannten Integrations-Tests erweitert. Diese Integrations-Test sind deutlich umfangreicher und können die korrekte Durchführbarkeit mehrerer Applikations-Szenarios verifizieren. Wie die einfachen Feature-Tests wird ebenfalls ein Testplan aufgestellt und die Ergebnisse dokumentiert. Falls ein Test fehlschlägt, wird ein Fehlerbericht angefertigt.

7.2 SanFrancisco Werkzeuge zur Applikationsentwicklung

Die wesentliche Intention von SanFrancisco, die Entwicklung komplexer Geschäftsanwendungen zu erleichtern, soll durch die Integration von Werkzeugen erzielt werden. Diese Werkzeuge sollen integraler Bestandteil von SanFrancisco und eine Entwicklungsunterstützung für den gesamten Lebenszyklus einer Geschäftsanwendung sein (siehe Abbildung 8).

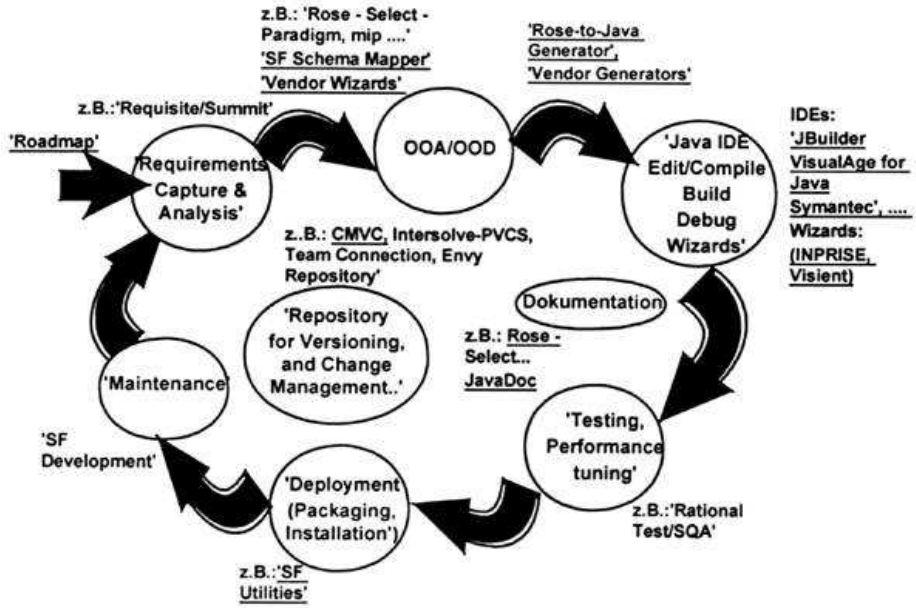

Abbildung 8: Phasen und Werkzeuge des Lebenszyklus einer Geschäftsanwendung

Wie obige Abbildung verdeutlicht, bleibt es dem Entwickler überlassen, welches Werkzeug in der jeweiligen Phase eingesetzt wird. Es sind oft mehre Werkzeuge denkbar. Damit Softwareentwickler nicht notwendigerweise zusätzliche Werkzeuge einkaufen müssen, liefert IBM ein wesentliches Werkzeug bereits mit. SanFrancisco enthält ein graphisches Modellierwerkzeug zur Erzeugung von Klassendiagrammen, welches exemplarisch im nächsten Abschnitt vorgestellt wird.

7.2.1 Der SFBuilder

Der SFBuilder ist ein visuelles Designwerkzeug zur Modellierung von UML-Klassendiagrammen. Der Vorteil gegenüber anderen im Grunde mächtigeren Werkzeugen ist, dass dieses alle SanFrancisco Besonderheiten kennt. Vor allem die Klassenhierarchie, die ein Geschäftsobjekt ausmachenden Klassen und die

SanFrancisco Pattern. Der Architekt modelliert die einzelnen Geschäftsobjekte und deren Beziehungen, nicht aber die Factory-, die Implementations- und die Interfaceklasse. Die erzeugt der SFBuilder automatisch. Um die Typen der Geschäftsobjekte zu verdeutlichen, wurde die UML Notation um folgende Symbole erweitert:

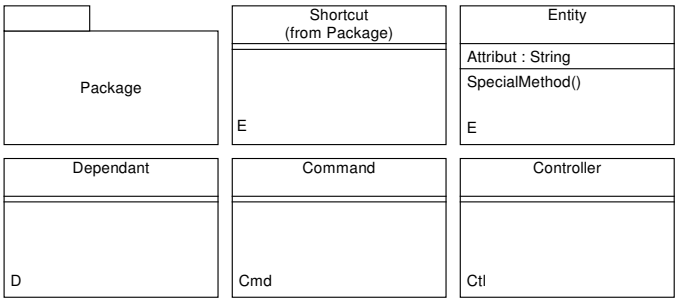

Abbildung 9: Erweiterungen der UML Symbole um SanFrancisco eigene Klassentypen

Die Buchstaben in der linken unteren Ecke des Symbols geben den Typ des SanFrancisco Objekts an, also die Klasse von der sie abgeleitet ist (vgl. Abschnitt 5.1.1, S. 36).

Der SFBuilder kennt die SanFrancisco Pattern. Soll ein Pattern verwendet werden, ist dieses nur noch in das aktuelle Diagramm zu kopieren. Ein Pattern kann außer Klassen und Beziehungen auch Kode enthalten.

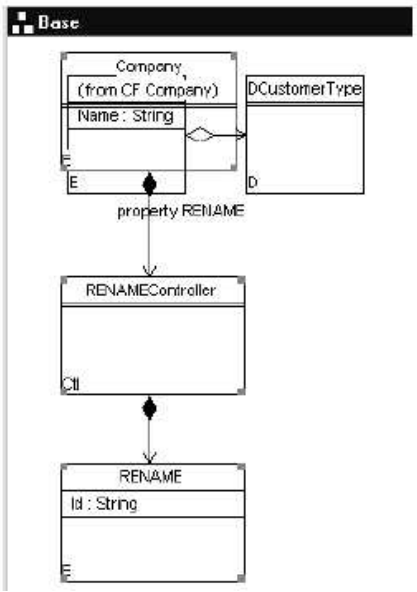

Abbildung 10: Das Controlled Entity Pattern im SFBuilder

Ist die Modellierung abgeschlossen, können die Methoden direkt im SFBuilder implementiert werden. Dazu wird der Kode einfach in einem Eigenschaftenfeld eingetragen. Abschließend kann das Werkzeug den gesamten Kode erzeugen und kompilieren.

8 Bewertung von SanFrancisco

Dieses Kapitel fügt die Arbeit der Kapitel 2 bis 7 zusammen. SanFrancisco wird anhand des in Kapitel 2 aufgestellten Bewertungskatalogs untersucht. Zu jedem Kriterium werden die zugehörigen SanFrancisco Merkmale aufgeführt. Die Zusammenfassung am Ende des ersten Abschnitts gewichtet die Ergebnisse und liefert eine Gesamteinschätzung von SanFrancisco. Der zweite Abschnitt geht auf die Praxisrelevanz von SanFrancisco ein. Dort werden zwei Projekte kurz vorgestellt. Ziel dieser Arbeit ist zu prüfen, ob SanFrancisco ebenso flexibel wie Individualsoftware ist und ob die Entwicklung einer Geschäftsanwendung durch den Einsatz von SanFrancisco vereinfacht wird. Der dritte Abschnitt beantwortet die Fragenstellung dieser Arbeit.

8.1 Überprüfung des Bewertungskatalogs

Der in Kapitel 2 aufgestellte Bewertungskatalog findet sich nachfolgend wieder. Den einzelnen Kriterien werden die passenden SanFrancisco Merkmale gegenübergestellt.

8.1.1 Allgemeine Kriterien

Wichtige Kriterien, welche die Flexibilität und die Vereinfachung der Softwareentwicklung nicht beeinflussen, werden als allgemeine Kriterien zusammengefasst. Für eine Geschäftsanwendung sind diese trotzdem entscheidend, da sie nicht einfach umzusetzen sind. Hier wird untersucht, inwieweit SanFrancisco diese allgemeinen Kriterien erfüllt.

Softwarequalität: Die Softwarequalität, genauer die Korrektheit, die Adäquatheit und Erlernbarkeit, kann nur im Zusammenhang mit dem Einsatzkontext bewertet werden. Ein qualitativ hochwertiger Softwareprozess kann wesentlich dazu beitragen, Softwarequalität zu erhöhen. Die Benutzungsschnittestelle und die automatisierten Prozesse bilden den zentralen Teil der Entwicklung einer SanFrancisco basierten Geschäftsanwendung, unterliegen damit also dem Softwareprozess. Zuverlässigkeit kann erreicht werden, da die SanFrancisco Infrastruktur zuverlässig, getestet und robust ist. Die zu entwickelnden Funktionalitäten werden in überschaubare Teile zerlegt (`Commands`), so dass diese dadurch zuverlässig und testbar bestimmten Verantwortlichkeiten entsprechen. Die Lesbarkeit, Erweiterbarkeit und Testbarkeit wird durch den Einsatz objektorientierter Softwaremethoden, insbesondere von Entwurfsmustern und Rahmenwerken, verbessert. Java als Entwicklungsplattform ermöglicht eine gute Portabilität, welche sich in der Liste der unterstützen Betriebsysteme und Hardware wiederspiegelt. Die SanFrancisco Softwarearchitektur bietet also viele Ansätze, darauf basierend Geschäftsanwendung guter Qualität zu erstellen.

Robustheit: SanFrancisco verwaltet Konflikte und bietet die Möglichkeit, diese aufzulösen. Alle auftretenden Probleme werden protokolliert und können eingesehen werden. Detaillierte Fehlermeldungen können in jedem Dialog immer an der gleichen

Stelle eingesehen werden. Vorherige Fehler sind ebenfalls einsehbar. SanFrancisco ist in Komponenten unterteilt, die miteinander kooperieren. Einzelne Komponenten können ausgetauscht oder weggelassen werden. Insgesamt wirkt SanFrancisco robust.

Ausfallsicherheit: Die Ausfallsicherheit einer SanFrancisco basierten Geschäftsanwendung hängt von der Erreichbarkeit des Global Server Managers ab. Diese zentrale Stelle des Logischen SanFrancisco Netzwerks ist die Schwachstelle der SanFrancisco Infrastruktur.

Mandantenfähigkeit: SanFrancisco ist voll mandantenfähig. Die Daten verschiedener Firmen können getrennt werden. Zusätzlich können die Daten unterschiedlich konfiguriert werden, also unterschiedliche Attribute enthalten. Mehr noch, es können sogar unterschiedliche Methoden und Prozesse für gleiche Geschäftsvorfälle definiert werden. Daten und Funktionalitäten von Firmen können auf verschiedenen Servern eingerichtet werden, also auch hardwaremäßig getrennt werden.

Rechtemanagement: Die Rechteverwaltung von SanFrancisco basiert auf Benutzern und Gruppen, denen Berechtigungen zugeordnet werden. Wozu Berechtigungen befähigen, wird während der Entwicklung mittels benannter Sicherer Abschnitte (*Secure Sections*) festgelegt. Das ausführen eines `Commands` könnte zum Beispiel eine solche Berechtigung erfordern. Denkbar ist auch, das Öffnen einer Eingabemaske zu schützen.

Internationalisierbarkeit: SanFrancisco bietet die Möglichkeit, international einsetzbare Geschäftsanwendungen zu entwickeln. Bezeichnungen von Geschäftsobjekten können in verschiedenen Sprachen angelegt werden. Wichtiger ist jedoch, dass Funktionalitäten landestypisch umgesetzt und konfiguriert werden können. Das sieht SanFrancisco ebenfalls vor.

Softwareprozess: Das *SanFrancisco Application Development Roadmap* ist, was die einzelnen Arbeitsschritte der Softwareentwicklung betrifft, sehr detailliert. Das Prozessmanagement und die Qualitätssicherung werden erwähnt, nicht aber ihrer Bedeutung angemessen beschrieben. Es wird auf die Wichtigkeit verwiesen, ohne dabei ins Detail zu gehen. Der Nutzen des *Roadmaps* gleicht einer Orientierungshilfe, die dazu genutzt werden kann, einen eigenen qualitativ hochwertigen Softwareprozess zu definieren.

8.1.2 Flexibilität

Die Flexibilität einer Geschäftsanwendung bedeutet nicht nur die Anpassung und Erweiterung der Funktionalitäten, sondern viel mehr die optimale Anpassung der Software an organisatorische Anforderungen eines Unternehmens. Da die Dialogfolge und Benutzungsschnittstelle von SanFrancisco nicht festgelegt ist, bietet sich die Möglichkeit, unterschiedliche Geschäftsanwendungen zu realisieren.

Durch die Verwendung von `Commands` und die Möglichkeit, diese manuell auszuführen oder in vorgefertigte Prozessabläufe zu integrieren, bietet sich eine organisatorische Flexibilität. Die vorhandenen Funktionalitäten der Anwendungsdomänen sind lose gekoppelt. Die Datenhaltung von SanFrancisco ist konfigurierbar und kann vorhandene Daten in vorhandenen Strukturen nutzen. SanFrancisco bietet damit die Möglichkeit, flexibel in bestehende Softwaresysteme integriert zu werden. Funktionale Flexibilität

bietet SanFrancisco durch die konsequente Verwendung objektorientierter Softwaremethoden, wie Entwurfsmuster.

Nachfolgend werden SanFrancisco Merkmale den Kriterien des Bewertungskatalogs zugeordnet.

Offenheit: SanFrancisco basiert auf den Industriestandards Java und SQL. Durch den Einsatz von Java lassen sich leicht weitere Standards wie TCP/IP *Sockets*, RMI, XML, SOAP oder EJB integrieren. SanFrancisco speichert persistente Daten in relationalen Datenbanken. Das ist der Offenheit förderlich, da die Daten von anderen System genutzt werden können.

Integrationsfähigkeit: Dadurch das SanFrancisco alle Java Kommunikations-möglichkeiten (siehe Offenheit) zur Verfügung stehen und SanFrancisco prozessorientiert ist, können softwareübergreifende Prozesse umgesetzt werden. Es ist möglich persistente Prozesse, also solche, die auf Ergebnisse anderer Softwaresysteme warten, einzurichten. Der manuelle Eingriff in auf diese Wiese integrierte Prozesse ist nur für Problemfälle nötig.

Erweiterbarkeit: Wie das SanFrancisco *Roadmap* deutlich macht, ist SanFrancisco innerhalb der vorhanden Anwendungsdomänen flexibel erweiterbar. Es lassen sich sogar weitere Domänen integrieren. Die Trennung von Daten, Darstellung und einfachen Prozessen (`Commands`) fördert die Erweiterbarkeit zusätzlich. Sollten neue Attribute nötig werden, können diese ohne Kodeänderung durch Konfigurationen dazugefügt werden.

Skalierbarkeit: Das Logische SanFrancisco Netzwerk ermöglicht die beliebige Verteilung von Serverkomponenten. Das *Scheduling* des Betriebssystems kann durch die Einrichtung mehrere Prozesse und dadurch mehrere *Java Virtual Machines* optimal ausgenutzt werden, ebenso die Speicherverwaltung. Die Leistung von Mehrprozessormaschinen kann vollständig genutzt werden. Ist die Rechenleistung ausgeschöpft, kann dem System ein weiterer Rechner mit weiteren Prozessen zugefügt werden. SanFrancisco ist gut skalierbar.

8.1.3 Vereinfachung der Softwareentwicklung

Wie in Kapitel 2 bereits herausgestellt wurde, kann eine vorhandene Softwarearchitektur, vorhandene Funktionalität und geeignete Werkzeuge die Softwareentwicklung vereinfachen.

Architektur: Die SanFrancisco Softwarearchitektur ist sinnvoll, einsichtig und vor allem tragfähig. Die Unterscheidung in Fundament-, Allgemeine Geschäftsobjekt- und Kernprozessschicht ist nachvollziehbar. Die Kommunikationsinfrastruktur, die eine verteilte Rechnerkonfiguration zulässt, ist durchdacht und stabil. Gut gelöst ist die Trennung der einzelnen Anwendungsdomänen. Die Architektur ist offen gestaltet und lässt sich in vielerlei Hinsicht verändern und erweitern. Es können neue Anwendungsdomänen dazugefügt werden oder bestehende weggelassen werden. Die persistente Datenhaltung ist integriert und bedarf keines komplexen Designs.

Funktionalität: Die vorhandene Geschäftslogik der Anwendungsdomänen Buchhaltung, Lagerhaltung und Auftragsabwicklung sind vielfältig und ebenfalls tragfähig. Geschäftsvorfälle dieser Anwendungsdomänen lassen sich gut durch

vorhandene Geschäftsobjekte und Funktionalitäten abdecken. Dabei wird Wert auf größtmögliche Flexibilität gelegt. Sollten Objekte oder Funktionalitäten nicht ausreichen, können zusätzliche angelegt und eingepasst werden.

Erlernbarkeit: Wie in Kapitel 2 angemerkt, hängt die Erlernbarkeit zum einen von der Verständlichkeit und Vollständigkeit der Dokumentation, zum anderen von der Komplexität des zu Lernenden ab. SanFrancisco ist alleine durch den Umfang, die Auftragsabwicklung enthält z.B. 400 Geschäftsobjekte (1200 Klassen), komplex. Durch den konsequenten Einsatz von Entwurfsmustern, wird versucht diese Art der Komplexität zu reduzieren. Die über 100 (vgl. [SchmiLi99]) eingesetzten Entwurfsmuster müssen aber auch erst verstanden sein, damit sie erkannt und deren Vorteile genutzt werden können. Die Dokumentation der Funktionalitäten ist umfangreich und selten unvollständig. Die Dokumente sind verständlich aber dokumentübergreifend schlecht strukturiert. Nötige Informationen finden sich in unterschiedlichen Dokumenten ohne nötigen Verweis aufeinander. Für Einsteiger werden Einführungskurse angeboten, die aber in keiner Form in der Dokumentation Eingang gefunden haben. Wer sich selbstständig in SanFrancisco einarbeiten möchte, muss mit einem hohen Einarbeitungsaufwand rechnen, der sich dann aber auszahlt (siehe [SchmiLi00]).

Werkzeuge: In Ankündigungen verfolgt IBM eine großmundige Werkzeugstrategie und propagiert sogar eine integrierte Entwicklungsumgebung. Davon ist in der Praxis wenig zu sehen. Es gibt für einzelne Aufgaben Werkzeuge, wie das vorgestellte Modellierwerkzeug SFBuilder, die aber einfach gehalten sind und für die Entwicklung komplexer Softwareprojekte nicht ausreichen. Das *Roadmap* wird durch keinerlei Werkzeuge unterstützt.

8.1.4 Zusammenfassung

Die vielfältigen Merkmale zeigen, dass SanFrancisco eine komplexe und leistungsfähige Software zur Entwicklung von Geschäftsanwendungen ist. Wie aus der Vorstellung der vorhandenen Geschäftslogik abzuleiten ist, ist diese verwendbar und gleichzeitig sehr flexibel an Bedürfnisse eines Unternehmens anpassbar. SanFrancisco kann seine Stärken aber erst ausspielen, nachdem die Entwickler einen Lernprozess durchlaufen haben. Je nach Vorkenntnissen des Entwicklungsteams in objektorientierten Methoden, ist dieser Prozess einfacher oder schwierig. Die Dokumentation ist ausreichend, SanFrancisco zu verstehen, könnte aber für Einsteiger besser strukturiert sein. Das Fehlen von Werkzeugen, die speziell auf SanFrancisco ausgelegt sind, kommt erschwerend hinzu. Dem Entwickler könnten viele Routinetätigkeiten abgenommen werden. Trotzdem spricht die Leistungsfähigkeit von SanFrancisco dafür, es einer Individualprogrammierung vorzuziehen. Die Architektur des Rahmenwerks und die Kommunikationsinfrastruktur sind tragfähig. SanFrancisco ist einfach erweiterbar und im Hohen Maße integrationsfähig. Besonders herauszustellen ist die Skalierbarkeit. Insgesamt ist SanFrancisco in vollem Umfang dazu geeignet, darauf basierende komplexe Geschäftsanwendungen für große Unternehmen zu entwickeln.

8.2 Einsatz von IBM SanFrancisco in der Praxis

Die Relevanz von SanFrancisco in der Praxis ist schwer zu ermitteln. Viele Firmen bekennen sich zu SanFrancisco, die meisten davon bieten aber nur Entwicklungsdienste an. Kaum eine Firma weist darauf hin, dass eines ihrer Produkte auf SanFrancisco

basiert. Es findet sich oft nur der Hinweis auf eine Partnerschaft mit IBM. Zwei Projekte lassen sich aber dennoch benennen, die kurz vorgestellt werden.

Sehr früh, nämlich bereits Ende 1997, hat sich die IBS Consist B.V. für den Einsatz von SanFrancisco entschieden. Ihre Erfahrungen sollen hier zusammengefasst werden. IBS Consist entwickelt seid über 20 Jahren Finanz- und Personalverwaltungssoftware für Unternehmen ab 100 Mitarbeitern. IBS Consist hat über 1200 Kunden, hauptsächlich in Holland und Belgien, und ihre Software dort über 1400 mal installiert. Zielplattform sind Mainframe Systeme von IBM, als Programmiersprache wird die prozedurale Sprache RPG verwendet. IBS Consist erkannte die zunehmende Trennung in Anbieter von Komplettlösungen, wie SAP, Baan, Oracle usw., und kleineren spezialisierteren Anbietern. Sie sahen und sehen sich als Teil eines *best-of-breed* Ansatzes, der aber zunehmend eine stärkere Integration der Anwendungen erforderlich macht, um mit den großen Softwarelösungen konkurrieren zu können. Für eine einfache Integration verschiedener Softwaresysteme, halten sie eine gemeinsame Infrastruktur, wie sie es nennen *Backbone Technology*, für notwendig. In Java und vor allem SanFrancisco sahen sie diese Infrastruktur bereits umgesetzt. Es wurde zunächst ein Prototyp entwickelt, um die Fähigkeiten von SanFrancisco zu untersuchen. Dieser Test war zufriedenstellend. Danach wurden nach und nach Applikationen für den Kundeneinsatz entwickelt, die in Lego Manier zusammengebaut werden können. IBS Consist ist mit der Leistungsfähigkeit von SanFrancisco im Allgemeinen zufrieden, das Konzept hat sich als tragfähig erwiesen. Zwei große Probleme haben sich aber heraus kristallisiert. Unterschätz wurde der Aufwand, erfahrenen RPG-Programmierern in die objektorientierte Softwareentwicklung einzuweisen. Als zweites großes Problem stellte sich der Abgleich der Anforderungen mit den Funktionalitäten von SanFrancisco heraus (nachzulesen in [Sa98], [Sa00]).

Die Firma skyva International geht einen anderen Weg. Für sie sind die individuellen Geschäftsprozesse eines Unternehmens zentral. Ihre Lösung basiert darauf, durch einen geeigneten Prozess die Unternehmensprozesse zu modellieren und daraus direkt eine Software zu generieren. Vorgesehen ist die Integration von Fremdsystemen. Ihre Softwarelösung unterstützt diesen skyva Modellierungs- und Integrationsprozess. Bei der Modellierung kann auf vorgefertigte Funktionalitäten und Lösungen zurückgegriffen werden. Es sind Werkzeuge zur Prozessmodellierung, Datenmodellierung und Dialoggestaltung vorhanden. Die skyva Lösung basiert im Kern auf SanFrancisco (vgl. [Skyva99]).

8.3 Beantwortung der Fragestellung

Zur Erinnerung, die Fragestellung dieser Arbeit lautet:

> **Fragestellung**: In welchem Maße wird die Entwicklung komplexer Geschäftsanwendungen durch IBM SanFrancisco vereinfacht und inwieweit gleicht die Flexibilität einer IBM SanFrancisco basierten Anwendung der einer Individualsoftware ?

Ob SanFrancisco die Softwareentwicklung vereinfacht, hängt von der Anwendungsdomäne und von der Größe des Projekts ab. Der nicht unerhebliche Lernaufwand muss sich amortisieren können. Große Projekte profitieren von der SanFrancisco Architektur und Funktionsfülle, was den Lernaufwand rechtfertigt. Das gilt zum Teil auch für Geschäftsanwendungen anderer Domänen, für die SanFrancisco

keine Geschäftsprozesse anbietet. Die Vorteile von SanFrancisco, wie Skalierbarkeit, Datenhaltung, mögliche Verteilung, kommen selbst hier zum tragen (vgl. [SchmLi00]). Falls nur die SanFrancisco Infrastruktur verwendet werden soll, sollte auch andere *Middleware* in Betracht gezogen werden. SanFrancisco ist nicht zu einem Industriestandard geworden. Andere Techniken, wie EJB, CORBA, COM+ und .NET, werden sich durchsetzen.

Der zweite Teil der Fragestellung kann bejaht werden. Der Flexibilität von SanFrancisco sind kaum Grenzen gesetzt. Wird diese Freiheit voll ausgeschöpft und entfernt man sich vollständig von den SanFrancisco Strukturen, so gehen auch die Vorteile verloren. Eine Vereinfachung der Softwareentwicklung ist dann nicht mehr gegeben. Wird das SanFrancisco Programmiermodell eingehalten, leidet die Flexibilität nicht wesentlich. Die Benutzungsschnittstelle, die Verteilung der Prozesse aber auch die Gestaltung und Unterstützung der Unternehmensprozesse unterliegen dem Softwareprozess. Sie können deutlich flexibler umgesetzt werden, als Standardsoftware es ermöglicht.

IBM hat also tatsächlich den Entscheidungspolen Einführung von Standardsoftware und Individualprogrammierung eine weitere gangbare Option dazugefügt.

9 Zusammenfassung

Diese Arbeit untersuchte, inwieweit IBM SanFrancisco die anwendungsorientierte Softwareentwicklung unterstützt. Es hat sich herausgestellt, dass SanFrancisco einen dritten Pol in dem Spannungsfeld von Standardsoftware und Individualprogrammierung darstellt. SanFrancisco ist geeignet, komplexe Geschäftsanwendungen darauf basierend zu entwickeln.

Um diese Bewertung vorzunehmen, wurde zunächst der Gegenstandsbereich definiert. Generelle Überlegungen zu Geschäftsanwendungen, Softwareentwicklung und Vereinfachung dergleichen, führten zu Kriterien eines Bewertungskatalogs. SanFrancisco sollte daran gemessen werden. Es hat sich ergeben, dass die Entwicklung von Anwendungssoftware ein äußerst komplexer Vorgang ist. Anwendungssoftware muss vielen Kriterien genügen, die in die Softwarearchitektur eingehen. Ebenso wichtig zur Verbesserung der Softwarequalität stellte sich der Softwareprozess heraus. Als wesentlicher Teil, wurde das Qualitätsmanagement genauer betrachtet.

Nachfolgend wurden vier Aspekte von SanFrancisco, nämlich die Ausführungs-umgebung, die Softwarearchitektur, bereits vorhandene Funktionalitäten und der Softwareprozess untersucht. Ziel dabei war, Merkmale von SanFrancisco zu identifizieren, die eine fundierte Beurteilung zulassen.

Die Untersuchung der Ausführungsumgebung hat gezeigt, dass SanFrancisco eine leistungsfähige Systemarchitektur zur Verfügung stellt. Darauf aufbauend lassen sich verteilte, komponentenbasierte Anwendungssysteme implementieren. SanFrancisco Geschäftsobjekte sind ohne Entwicklungsaufwand persistent und können in relationalen Datenbanken gespeichert werden. Es hat sich gezeigt, dass persistente Objekte ganz unterschiedlich gespeichert werden können. Daten von Fremdsystemen können direkt genutzt werden.

Die Untersuchung der Softwarearchitektur ergab eine sinnvolle Schichtung. SanFrancisco ist als Rahmenwerk aufgebaut. Die Benutzungsschnittstelle und die vorhandenen Geschäftsprozesse müssen konkretisiert werden. Zur Entwicklung von SanFrancisco wurden konsequent Entwurfsmuster eingesetzt. Viele Funktionalitäten werden dem Entwickler bereits geliefert. Dazu gehört die Persistenz der Geschäftsobjekte aber auch die mögliche verteilte Ausführungsumgebung.

Die Vorstellung der mitgelieferten Geschäftslogik zeigte, dass SanFrancisco nicht für jede Anwendungsdomäne bereits Geschäftslogik vorhält. Der Fokus lag zunächst auf den Bereichen Buchhaltung, Warenwirtschaft und Auftragsabwicklung. Die dort vorhandene Funktionalität ist umfangreich und verwendbar, was zwei Erfahrungsberichte bekräftigen.

IBM schlägt ein Vorgehensmodell zur Entwicklung von SanFrancisco basierten Geschäftsanwendungen vor. Die genauere Betrachtung des *Roadmaps* ergab einen phasenorientierten Prozess, der geeignet sein kann, qualitativ hochwertige

Anwendungssoftware entstehen zu lassen. Das Vorgehen ist nicht zwingend. Es soll den Einstieg in objektorientierte Softwareentwicklung und ein strukturiertes Vorgehen erleichtern.

Die Bewertung von SanFrancisco schließlich und die Beantwortung der Fragestellung ergaben, dass komplexe Geschäftsanwendungen in großen Unternehmen gut auf IBM SanFrancisco basierend entwickelt werden können.

9.1 Ausblick

Diese Arbeit hat viele Aspekte von SanFrancisco untersucht und vorgestellt. Zusätzlich interessant ist der Vergleich von SanFrancisco mit anderen Anwendungsrahmenwerken. Der direkte Vergleich von *Middleware* wie CORBA oder EJB mit SanFrancisco verspricht ebenfalls spannende Ergebnisse. Außerdem könnte der Frage nachgegangen werden, welche Qualität die vorhandene Geschäftslogik hat.

Nicht ausreichend geklärt werden konnte die konkrete Rolle der verschiedenen Dienste des Lokalen SanFrancisco Netzwerks. Die Darstellung einer Benutzeranfrage hätte das Zusammenspiel gut verdeutlichen können. Es findet sich aber keine solche Beschreibung in frei zugänglicher Literatur.

Ein weiteres Kapitel hätte die SanFrancisco Entwurfsmuster, es gibt über 100, vorstellen können. Die finden sich in einem Buch (siehe [CCG00]), das ich bei Interesse als Lektüre empfehle.

Aufgrund nicht ausreichender Informationen, konnte das Schema-Mapping von Geschäftsobjekten in Datenbankrelationen ebenfalls nicht ausreichend geklärt werden. Offen ist, wie Objektbeziehungen in der Datenbank abgebildet werden.

IBM stellt die Weiterentwicklung von SanFrancisco ein. SanFrancisco als vielversprechender Ansatz im Spannungsfeld von Standardsoftware und Individualprogrammierung wird also keine Neuerungen und Erweiterungen mehr erfahren. Welche Gründe dazu geführt haben ist fraglich. Waren es ökonomische Gründe, was diese Arbeit vermuten lässt, oder ist das Vorgehen, Applikationen auf zugekaufte Anwendungsrahmenwerke basierend zu entwickeln, nicht tragfähig ? Die Komponententechnologie Enterprise Java Beans von Sun und entsprechender EJB Container wird die Entwicklung von Komponenten sicherlich beschleunigen. Anzeichen dafür finden sich bereits im Internet. Es haben sich Projekte zusammengefunden, die sogenannte Business-Frameworks entwickeln, teilweise sogar frei.

Literaturverzeichnis

[AKC00] AKÇIÇEK, CELAL: *Objekttransformationen bei einem Paradigmenwechsel: Konzepte und Werkzeuge zur Abbildung von Programmobjekten auf Datenbankrelationen*, Universität Hamburg, Informatik, Studienarbeit, Oktober 2000

[Balz98] BALZERT, HELMUT: *Software-Management, Lehrbuch der Software-Technik: Software-Qualitaetssicherung, Unternehmensmodellierung*, Bd. 2, Heidelberg 1998, Spektrum, Akad. Verl.

[Bo98] BOHER, K. A.: *Architecture of the San Francisco Frameworks*, In: IBM Systems Journal, Jg. 37 (1998), Nr. 2.

[Boch00] BOCHYNEK, RAFAEL: *Ein Framework für E-Business*, Universität Mannheim, Lehrstuhl Wirtschaftsinformatik II, Seminararbeit, Oktober 2000
www.bwl.uni-mannheim.de/Niedereichholz/Angebot/Veranstaltungen/Ein_Framework_fur_E-Bussiness.pdf

[Bu00] BUNTING, R.: *Bridging the framework modeling and implementation gap*, In: IBM Systems Journal, Jg. 39 (2000), Nr. 2

[CCG00] CAREY, JAMES AND CARLSON, BRENT AND GRASER, TIM: *SanFrancisco(tm) Design Patterns: Blueprints for Business Software*, March 2000, Addison Wesley Publishing Company

[EBL00] EMDE BOAS-LUBSEN, H. VAN: *Business Component Prototyper for SanFrancisco: An experiment in architecture for application development tools*, In: IBM Systems Journal, Jg. 39 (2000), Nr. 2

[Fei00] FEINDT, JAN: *IBM SanFrancisco*, Universität Oldenburg, Informatik, Seminararbeit, 2000
kosobar.offis.uni-oldenburg.de/seminar/ausarbeitungen/Jan%20Feindt%20-%20IBM%20SanFrancisco.pdf

[FlZü97] FLOYD, C. UND ZÜLLIGHOVEN, H.: *Softwaretechnik*, In: RECHENBERG/POMBERGER (HRSG.), *Informatik-Handbuch*, Hanser Verlag, München, Wien, S. 641-667, 1997

[GHJV95] GAMMA, E. AND HELM, R. AND JOHNSON, R. AND VLISSADES, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, 1995

[GT97] GAUTHIER, CHUCK AND TRUXAL, DAVE: *IBM SanFrancisco application development environment*, IBM White Paper, 1997

[Han96] HANSEN, HANS R.: *Wirtschaftsinformatik I*, 7. Aufl., Stuttgart 1996, Lucius und Lucius

[He98] HENN, JUERGEN: *IBM SanFrancisco, Object-oriented infrastructure and reusable business components for distributed, multi-platform business applications – implemented entirely in Java*, In: Software – Concepts & Tools, Jg. 19 (1998), S. 37-48, Springer-Verlag, 1998

[HMF92] HESSE, WOLFGANG AND MERBETH, GÜNTER AND FRÖLICH, RAINER: *Software-Entwicklung: Vorgehensmodell, Projektführung, Produktverwaltung*, Oldenbourg Verlag, München, 1992

[IbmAD98] IBM CORP. (HRSG.): *Administering and Configuring San Francisco*, IBM Produkt Dokumentation, In: SanFrancisco Evolution Kit (V1R3), 1998

[IbmCF99] IBM CORP. (HRSG.): *IBM SanFrancisco: Concepts and Facilities*, IBM SanFrancisco Product Information, Mai, 1999, www.ibm.com/software/ad/sanfrancisco/library/concepts.html

[IbmPG98] IBM CORP. (HRSG.): SanFrancisco *Programmers Guide*, IBM Produkt Dokumentation, In: SanFrancisco Evaluation Kit (V1R3), 1998

[IbmRM98] IBM CORP. (HRSG.): *SanFrancisco Roadmap*, IBM Produkt Dokumentation, In: SanFrancisco Evaluation Kit (V1R3), 1998

[IbmTS97] IBM CORP. (HRSG.): *IBM SanFrancisco Technical Summary*, IBM Report, 1997

[JA00] JOHNSON, V. M. AND AUSTRENG, D. K.: *IBM SanFrancisco: Moving into the marketplace*, In: IBM Systems Journal, Jg. 39 (2000), Nr. 2

[MA99] MATZEN, BJARNE: *Building SanFrancisco Applications with SFBuilder version 1.1*, IBM Redbook, 1999 www.redbooks.ibm.com

[MPZL99] MATZEN, BJARNE AND PATIKAL, SHANKARA AND ZULIANI, FERNANDO AND LEE, EUNJOONG: *SanFrancisco Schema Mapping: Object Persistence and Legacy Integration*, IBM Redbook, July 1999 www.redbooks.ibm.com

[Oest97] OESTEREICH, BERND: *Objektorientierte Softwareentwicklung: mit der Unified modeling language*, München 1997, Oldenbourg Verlag

[Pich99] PICH, NORBERT: *Komponentenorientierte betriebliche Anwendungssysteme*, 1999, In: K. Turowski (Hrsg.): Tagungsband des 1. Workshops Komponentenorientierte betriebliche Anwendungssysteme (WKBA 1), Magdeburg, S. 52-58, www-wi.cs.uni-magdeburg.de/workshops/komponenten/tagungsband.html

[PO98] POLAN, M. G.: *Using the SanFrancisco Frameworks with VisualAge for Java*, In: IBM Systems Journal, Jg. 37 (1998), Nr. 2

[PoBl93] PLOMBERG, GUSTAV UND BLASCHEK, GÜNTHER: *Software Engineering: Prototyping und objektorientierte Software-Entwicklung*, München 1993, Hanser

[PSL98] SHARI AND LAWRENCE AND PFLEEGER: *Software engineering: theory and practice*, Upper Saddle River 1998, Prentice-Hall

[Ri00] RICKE, D. D.: *Technical Note: A three-dimensional framework for information technology solutions*, In: IBM Systems Journal, Jg. 39 (2000), Nr. 2

[Ro98] ROLF, ARNO: *Grundlagen der Organisations- und Wirtschaftsinformatik*, Berlin 1998, Springer-Verlag

[Sa00] SALM, ROB VAN DER: *Technical Note: IBS Consist two years later: The LEGO Brick dream*, In: IBM Systems Journal, Jg. 39 (2000), Nr. 2

[Sa98] SALM, ROB VAN DER: *Introducing shareable frameworks into a procedural development environment*, In: IBM Systems Journal, Jg. 37 (1998), Nr. 2

[SchmLi00] SCHMITZER, BENNO UND LIEßMANN, HARALD: *Rahmerwerk zur Integration von Software-Komponenten – Untersuchung und kritische Betrachtung des Teil-Frameworks „Order Management" im IBM SanFrancisco Framework*, 2000, In: K. Turowski (Hrsg.): Tagungsband des 2. Workshops Komponentenorientierte betriebliche Anwendungssysteme (WKBA 2), Magdeburg, S. 95-114, www-wi.cs.uni-magdeburg.de/workshops/komponenten2/tagungsband.html

[SchmLi99] SCHMITZER, BENNO UND LIEßMANN, HARALD: *Entwicklung von Komponenten für das betriebliche Rechnungswesen auf Basis des IBM SanFrancisco Frameworks – ein Erfahrungsbericht*, 1999, In: K. Turowski (Hrsg.): Tagungsband des 1. Workshops Komponentenorientierte betriebliche Anwendungssysteme (WKBA 1), Magdeburg, S. 75-87, www-wi.cs.uni-magdeburg.de/workshops/komponenten/tagungsband.html

[Skyva99] SKYVA INTERNATIONAL (HRSG.): *skyva Solution Environment und Methodologies*, Juni 1999

[StHa99] STAHLKNECHT, PETER UND HASENKAMP, ULRICH: *Einführung in die Wirtschaftsinformatik*, 9. Aufl., Berlin 1999, Springer

[WeOr92] WELTZ, FRIENDRICH UND ORTMANN, ROLF G.: *Das Softwareprojekt: Projektmanagement in der Praxis*, Frankfurt/Main 1992, Campus-Verlag

[WZD98] WEBER, OWEN AND ZULIANI, FERNANDO AND DOUGLAS, LOTUS AND SINGH, ASHOK AND AGGARWAL, ABHINAV AND VAN EMDE BOAS, GHICA: *SanFrancisco Evaluation Kit (V1R3): Getting Started*, IBM Redbook, November 1998.

Erklärung

Hiermit versichere ich, dass die vorliegende Studienarbeit von mir selbstständig erstellt wurde und ich außer den angegebenen Quellen keine fremden Hilfsmittel verwendet habe.

Hamburg, 6. Dezember 2001

_____

Unterschrift (Anthony Schmude)