24 Migration eines Altsystems zu einer Java-Enterprise-Architektur

Stefan Krieghoff, Wilhelm Hasselbring, Ralf Reussner, Niels Streekmann

In diesem Kapitel berichten wir über Erfahrungen mit der Migration kommunaler Informationssysteme hin zu einer SOA. In einer Kooperation zwischen der Forschung und einem kommunalen IT-Dienstleister wurde eine zukunftsfähige Software-Architektur erarbeitet, die aktuellen Standards entspricht und die die speziellen Randbedingungen kommunaler Verwaltungen berücksichtigt. Die selbst entwickelten Produkte des Software-Hauses sind bei ca. 200 kommunalen Kunden im Einsatz, haben lange und kostenintensive Phasen der qualitativen Stabilisierung und Funktionsanreicherung (was wir mit IQTF für Increasing Quality Times Functionality abkürzen) hinter sich und erfordern ständige Pflege, häufig bedingt durch Änderungen gesetzgeberischer Verwaltungsvorgaben.

Bei der Einführung einer neuen Software-Architektur in das Produktportfolio darf der Aufwand nicht unterschätzt werden. Insbesondere ist ein Konzept für eine Architekturmigration notwendig. Unsere Erfahrungen führten zu dem in Kapitel 10 vorgestellten *Dublo*-Muster. Dieses Beispielkapitel beschreibt den Rahmen des Projektes, das zur Beschreibung des Musters geführt hat. Bei der Umsetzung des Dublo-Architekturmusters mit den vorhandenen, erfahrenen Software-Entwicklern entstanden neue Probleme und Verzögerungen, deren Ursachen häufig im nicht technischen Bereich lagen und die man bei Beachtung bestimmter organisatorischer, psychologischer und betriebswirtschaftlicher Aspekte hätte verhindern können. Aus den gemachten Erfahrungen werden diese Aspekte isoliert und aufgezählt. Ein derartiger interdisziplinärer Ansatz, der bereits bei Definition und Auswahl einer zukünftigen Software-Architektur und des Migrationspfades dorthin verwendet wird, kann helfen, die Erfolgsquote solcher Umstellungsprojekte anzuheben.

24.1 Projektkontext

Die KDO (Zweckverband Kommunale Datenverarbeitung Oldenburg) ist ein erfolgreicher IT-Dienstleister, der domänenspezifische Software-Lösungen für kommunale Verwaltungen anbietet (`http://www.kdo.de`). Bisher basieren die Client-Server-Lösungen der KDO hauptsächlich auf Informix-Datenbanken mit Informix 4GL [IBM] und dem 4Js-Laufzeitsystem (serverseitig) inklusive dem 4Js-Windows-Frontend für Clients [J's]. Die Lösungen der KDO können wahlweise dezentral oder zentral mit der KDO als ASP betrieben werden.

Die KDO entschied sich, von der (monolithischen) Zwei-Schichten-Architektur hin zu einer standardbasierten und zukunftsorientierten Multi-Schichten-Architektur zu migrieren, die den Einsatz moderner Software-Engineering-Methoden ermöglicht. Bei der Veränderung von traditioneller hin zu komponentenbasierter Software-Entwicklung galt es, besonders auf die sich ergebenden Risiken und neuen Herausforderungen zu achten [RR03, Vit03].

Um den Übergang zu neuen Technologien zu bewältigen, entschied sich die KDO für eine Zusammenarbeit mit dem Oldenburger Informatik-Institut OFFIS. OFFIS ist ein An-Institut des Departments für Informatik der Universität Oldenburg (`http://www.offis.de`). Die hier vorgestellte Arbeit ist das Ergebnis gemeinsamer Anstrengungen der Software-Engineering-Abteilung der Universität Oldenburg, des OFFIS und der KDO. Die Kooperation wurde begleitet von einer Schulung der KDO-Mitarbeiter in objektorientierter Modellierung und Enterprise-Java-Technologien, die von OFFIS durchgeführt wurde. Dadurch wurden neue Software-Engineering-Methoden aus der universitären Forschung durch ein assoziiertes Technologie-Transfer-Institut in die industrielle Praxis übertragen.[1]

24.2 Architektur und Migrationsprozess

24.2.1 Vorhandene 4GL-Architektur

Die vorhandene Architektur ist in Abbildung 24.1 dargestellt. Informix- bzw. 4Js-4GL ist die zur Implementierung der serverseitigen Geschäftslogik verwendete Sprache. 4Js ist eine Entwicklungs- und Laufzeitumgebung, die es erlaubt, grafische Benutzungsschnittstellen auf Basis des Tcl/Tk-Toolkits [Ous94] zu generieren. Hierbei wird die Geschäftslogik fest mit dem Datenbankmanagementsystem verbunden; deswegen ist die Serverseite als eine Schicht zu betrachten.

[1]Die Autoren möchten den ehemaligen OFFIS-Mitarbeitern Holger Jaekel, Goran Martinic, Jürgen Schlegelmilch und Thorsten Teschke sowie den KDO-Mitarbeitern Marc Langnickel und Detlef Meyer für die sehr gute Mitarbeit in diesem Projekt danken.

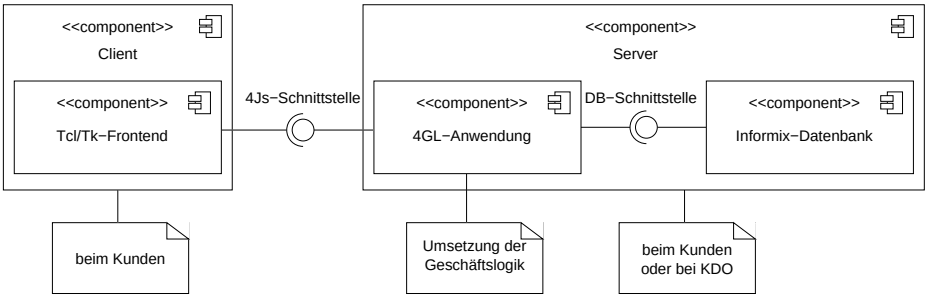

Abbildung 24.1: Vorhandene Zwei-Schichten-Architektur von Informix

24.2.2 Technologieauswahl

Die erste Aufgabe war die Auswahl der für die Realisierung zu verwendenden Komponententechnologie, deren Fähigkeiten und Eigenschaften von hoher Wichtigkeit sind. Im vorgestellten Kontext wurde J2EE [Sun08a] gewählt.

Als konkrete Werkzeuge wurden der BEA Weblogic-Applikationsserver [BEA], Together [Togb] für die Modellierung und JBuilder [JBu] als Entwicklungsumgebung ausgewählt. Zusätzlich zu kommerziellen Werkzeugen und System-Software sind verschiedene Open-Source-Systeme im Einsatz, z. B. Linux als Server-Betriebssystem.

Standardisierte und etablierte Muster sind für J2EE-Systeme dokumentiert [ACM03, BCJ+02, Bie02, Bro02, Mar02, Sun, VSW02]. Dieses wird als wichtige Vorbedingung erachtet, damit die geeigneten Architekturen für unsere Domäne entworfen werden können. Für Details zum Auswahlprozess sei auf [HRJ+04] verwiesen.

Man beachte, dass das Architekturmuster in Kapitel 10 vorgestellt wird, unabhängig ist von der eingesetzten Middleware-Technologie.

24.2.3 Prozess der Architekturauswahl

Der J2EE-Standard empfiehlt eine Multi-Schichten-Architektur. Ein typisches Beispiel mit 4 Schichten wird in Abbildung 24.2 veranschaulicht. In diesem Beispiel enthält die Clientschicht Java-Client-Anwendungen, die auf die Mittelschicht per Remote Method Invocation (RMI) zugreifen. Auf diese Mittelschicht, die aus einem Applikationsserver mit einem Container für Enterprise Java Beans (EJB) [Sun08a] besteht, kann auch aus Webcontainern heraus zugegriffen werden, wobei der Webcontainer Webbrowser-Anfragen via HTTP bedient; zur Vereinfachung der Darstellung wird auf diese Alternative verzichtet. Dieses Kapitel konzentriert sich auf die serverseitige Komponententechnologie.

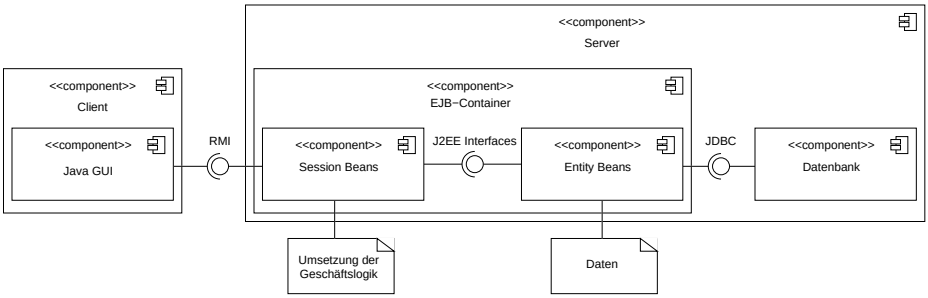

Abbildung 24.2: Die Vier-Schichten-J2EE-Zielarchitektur

Der EJB-Container verwaltet Session und Entity Beans. Entity Beans repräsentieren (passive) Datenobjekte, die persistent in Datenbanken gespeichert werden. Session Beans repräsentieren (kleine) Geschäftsprozesse, die auf die Entity Beans zugreifen, aber selbst keine persistenten Daten enthalten. Diese zwei Arten von EJBs begründen zwei logische Schichten in unserer Zielarchitektur: Geschäftsprozesse und Geschäftsobjekte. Weitere Klassifikationen von EJBs sind für unsere Diskussion nicht relevant.

Es entsteht nun die Frage, *wie* von der Zwei-Schichten-Legacy-Architektur in Abbildung 24.1 zur Vier-Schichten-Architektur in Abbildung 24.2 migriert werden kann. Abbildung 24.3 veranschaulicht einen unserer ersten Ansätze für eine Migrationsarchitektur am Beispiel einer Kfz-Zulassungs-Anwendung, die – wie zuvor beschrieben – eine Informix-4GL/4Js-Anwendung ist. Während der Migrationsphase können sowohl die alten 4GL/4Js-Informix-Clients als auch die neu entwickelten Java-Clients koexistieren, da sie gleichzeitig auf einem Endgerät lauffähig sind. Die Kommunikation zwischen Java-Clients und Applikationsserver ist durch spezielle KDO-Interfaces gekapselt, um unterschiedliche Formen der technischen Kommunikation zu unterstützen: RMI mit SSL für schnelle und sichere Kommunikation und Webservices über SOAP mit OSCI (Online Services Computer Interface) [OSC] für eine zertifiziert sichere Kommunikation über unsichere Kanäle wie das Internet. Eine ausführliche Diskussion dieser Sicherheitsbetrachtungen liegt außerhalb dieses Kapitels und ist nicht relevant für unsere Architekturdiskussion. Es ist jedoch wichtig, zu erwähnen, dass Sicherheitsbetrachtungen auf der Architekturebene häufig von Bedeutung sind und dass sie in unserer speziellen Architektur gelöst wurden, soweit dies in der Anwendungsdomäne kommunaler Informationssysteme erforderlich ist [KBS].

Die Kommunikation zwischen Applikationsserver und dem Server des Altsystems ist der kritische Punkt in dieser Architektur. Der erste Ansatz sah einen Zugriff des Applikationsservers auf die Datenbank des Legacy-Systems per JDBC vor. Die gewachsenen Datenbankstrukturen in Altsystemen offenbaren aber nicht

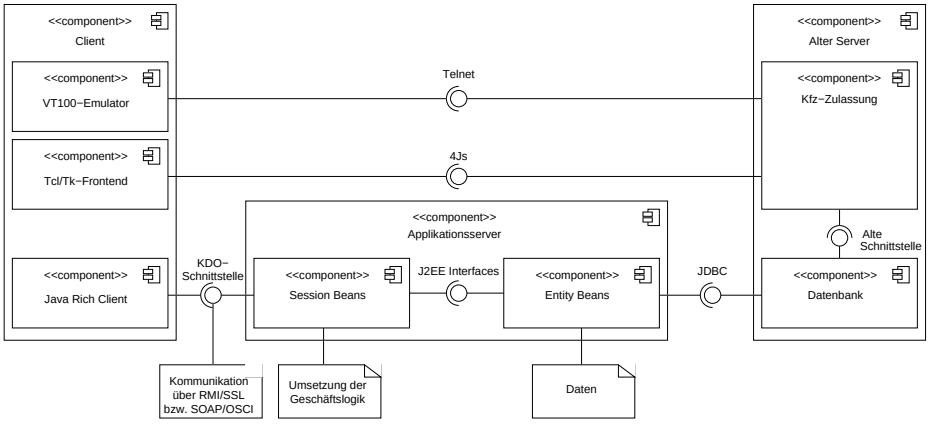

Abbildung 24.3: Erster Ansatz für eine Migrationsarchitektur

alle die für eine korrekte Benutzung erforderliche Semantik der gespeicherten Datenbankobjekte. Außerdem passen diese Strukturen in der Regel nicht zu den neu definierten Geschäftsobjekten der Mittelschicht.

So bestand unser zweiter Ansatz darin, die neuen Datenobjekte, die in neuer Technologie implementiert sind, in einer zusätzlichen Datenbank zu speichern. Dieser Ansatz hat den offensichtlichen und hochkritischen Nachteil, Konsistenzmechanismen zu benötigen, die die Daten zwischen alter und neuer Datenbank replizieren [GHOS96, Has97].

Die genannten Betrachtungen führten uns zu der schließlich ausgewählten Migrationsarchitektur, die in Abbildung 24.4 gezeigt ist. Für Legacy-Informationssysteme ist es offensichtlich, dass auf ihre internen Datenbanken niemals direkt zugegriffen werden sollte. Es ist empfehlenswert, auf sie über ein API (Application Programming Interface) zuzugreifen, das die Geschäftslogik »kennt«.

Wir haben uns hier für die Verwendung von SOAP entschlossen, als ein Kommunikationsprotokoll, das spezielle Adapter sowohl auf der EJB- als auch auf der 4GL-Seite erfordert. Die Webservice-Technologie basiert auf SOAP als Kommunikationsprotokoll, der Web Service Definition Language (WSDL) zur Schnittstellenbeschreibung und dem UDDI-Protokoll (Universal Description, Discovery and Integration), um dynamisch Webservices im Internet zu finden und zu benutzen [Lin03]. Andere Kommunikationsmechanismen wie JCA [SSN02] wären in diesem Zusammenhang ebenfalls möglich. Dieser bevorzugte Ansatz, auf Legacy-Informationssysteme über ein API zuzugreifen, ist aus vielen anderen Projekten, bei denen Altsysteme zu integrieren waren, gut bekannt [NHW+02, WF98].

Die hier gefundene Migrationsarchitektur wurde in dem Kapitel 10 vorgestellten Dublo-Architekturmuster verallgemeinert beschrieben. Die Vorteile

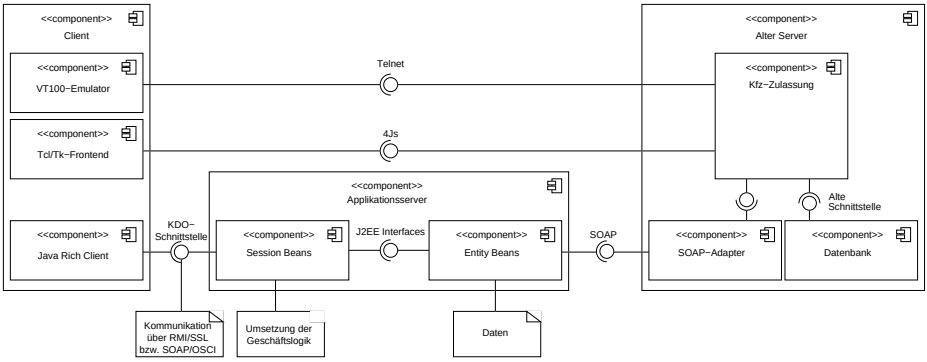

Abbildung 24.4: Endgültiger Ansatz für eine Migrationsarchitektur

einer Multi-Schichten-Architektur sind ausführlich in der Literatur vorgestellt [Bro00, FRF+02]. Multi-Schichten-Architekturen entstanden aus den Client-Server-Architekturen.

24.3 Nicht-technische Aspekte sanfter Migration

Das Software Engineering als Ingenieurdisziplin beschäftigt sich nicht nur mit technischen, sondern auch mit betriebswirtschaftlichen und psychologischen Fragestellungen. Ein hoher Anteil großer Software-Projekte überzieht das Budget und die Termine in erheblichem Maße, wobei nicht technische Gründe einen wesentlichen Anteil am Scheitern dieser Projekte haben [Gla02, MBP+04]. Im Folgenden werden wir aus der Erfahrung heraus nicht technische Aspekte aus diesen Bereichen ansprechen, deren Beachtung – je nach gesamtem Projektkontext – erheblichen Einfluss auf den Projekterfolg haben kann.

24.3.1 Betriebswirtschaftliche Aspekte sanfter Migration

Im Einsatz befindliche große Software-Systeme haben häufig – nach Fertigstellung der Erstversion – lange und kostenintensive Phasen der qualitativen Stabilisierung und Funktionsanreicherung (IQTF = Increasing Quality Times Functionality) hinter sich. Die Gesamtzeit seit Beginn der Entwicklung kann dadurch bis zu 10 Jahre in Anspruch nehmen. Die hier genannten Zeiträume stammen aus einem Entwicklungskontext mit den in Tabelle 24.1 angegebenen Kennzahlen. Die angeführten Zeiträume sind stark abhängig von verschiedensten Faktoren, wie z. B. Umfang der Software, Komplexitätsgrad der Software, Dynamik der Anforderungen, Anzahl der eingesetzten Entwickler, und deshalb nur als Beispiele zu betrachten.

Das Produkt aus Qualität und Funktionalität, kurz QTF, hat dabei über die Entwicklungsdauer etwa den in Abbildung 24.5 gezeigten Verlauf.

Anzahl	Maß
890.000	Lines of Code (4GL)
45	Personenjahre Entwicklungs- und Pflegeaufwand
9	Jahre Entwicklungsdauer
700	Dialogmasken
bis zu 6	beteiligte Mitabeiter
19.000	SQL-Statements

Tabelle 24.1: Kennzahlen

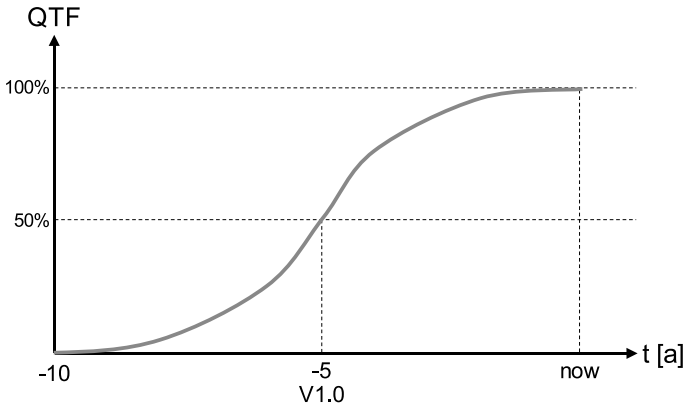

Abbildung 24.5: Produktlebenszyklus

Da erfahrungsgemäß in den wenigsten Projekten während der Entwicklungszeit auf eine völlig neue Technologie umgestellt wird, basieren solche bewährten Systeme nach beispielsweise 10 Jahren Entwicklungsdauer auf veralteten Technologien. Der Software-Hersteller muss auf diese Tatsache reagieren und sich zwischen Neuentwicklung und Migration entscheiden.

Big-Bang-Neuentwicklung

Eine vollständige Neuentwicklung nur zur Verwendung zeitgemäßer Technologien führt erst nach einer neuen IQTF-Phase zu einem Folgeprodukt mit ähnlicher Qualität und ähnlichem Funktionsumfang: Die Vorarbeiten für die

Neuentwicklung einer Folgeversion eines großen Software-Systems (Technologieauswahl, Architekturauswahl) erfordern i.d.R. etwa ein Jahr, die Entwicklung der neuen Erstversion mindestens 2–3 Jahre, gefolgt von der o.g. IQTF-Phase, die erfahrungsgemäß ebenfalls mindestens 3–4 Jahre erfordert.

Normalerweise wird man außerdem für die Neuentwicklung die fachlich erfahrenen Software-Entwickler des Erstprodukts einsetzen – diese müssen zunächst in den neuen Technologien geschult werden und dann Erfahrungen sammeln, was ebenfalls etwa ein Jahr dauert. Dadurch wird weniger an der Altversion gearbeitet, die QTF nimmt ab. Der Software-Hersteller kann also frühestens nach etwa 4 Jahren wieder mit Neuigkeiten am Markt erscheinen. Diese Zeitdauer bis zum nächsten Release kann kritisch für das Überleben eines Unternehmens werden und muss – durch geeignete langfristige Planung – unbedingt verkürzt werden, z. B. durch frühzeitigen Beginn der Arbeiten am Folgeprodukt. Es werden jetzt die Software-Entwickler des Altprodukts noch früher abgezogen, sodass die QTF der Altversion spürbar abnimmt (siehe Abbildung 24.6).

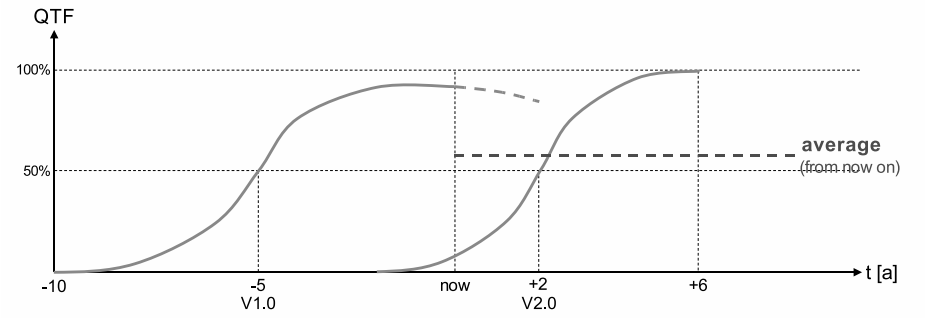

Abbildung 24.6: Produktlebenszyklen bei Neuentwicklung optimiert

Trotz Optimierung ergibt sich noch ein Zeitraum von etwa 2–3 Jahren, während dessen die Pflege des Altprodukts spürbar reduziert werden muss und noch kein Folgeprodukt zur Verfügung steht. Dies kann zu einem empfindlichen Verlust an Marktanteilen führen, auch bei den Bestandskunden, was kein Software-Hersteller in Kauf nehmen kann. Anwender müssen sich mit der Einführung des Folgeprodukts vollständig umstellen und zusätzlich auf Teile des gewohnten Funktionsumfangs verzichten.

Sanfte Migration

Eine sanfte Migration setzt auf der hohen QTF des Erstprodukts auf. Es werden in kleinen Schritten von 1–10% der jeweiligen Gesamtfunktionalität immer wieder Teile des Erstprodukts durch neu entwickelte Teile ersetzt. Man kann dies

als ein fortlaufendes Refactoring [Fow00] im Rahmen eines großen Plans, der Architekturmigration, betrachten. Dieses Vorgehen hat – wie beim Dublo-Muster bereits erläutert – den Nachteil, dass bestimmte Zusatzaufwände entstehen und die Entwicklung mit mehreren Technologien gleichzeitig erfolgen muss. Dies wird aber mehr als aufgewogen dadurch, dass man permanent ein stabiles und funktionsreiches Produkt am Markt halten kann: Die QTF des vorhandenen Produkts wird dabei potenziell bei jedem Schritt nur gering reduziert und kann anschließend direkt wieder aufgebaut werden (siehe Abbildung 24.7).

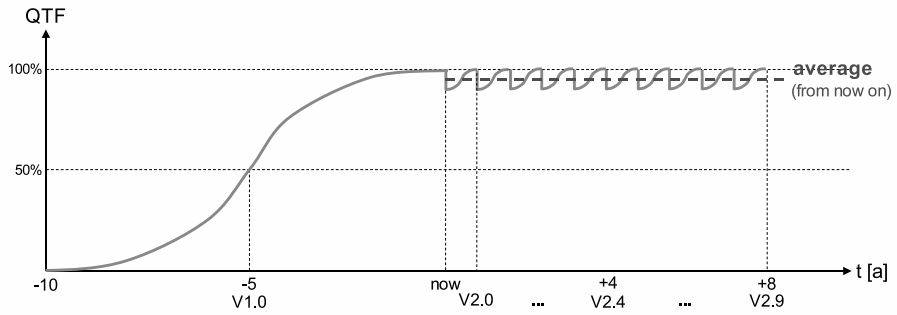

Abbildung 24.7: Produktlebenszyklus bei sanfter Migration

Der Anwender muss bei dieser Vorgehensweise evtl. mehrere Benutzungsschnittstellen gleichzeitig bedienen, die sich ihm zwar auf dem gleichen Endgerät präsentieren, aber doch unterschiedlich bedient werden. Dieser Nachteil wird i.d.R. durch die Vorteile und Verbesserungen des ersetzten Produktteils wieder hinreichend aufgehoben. Außerdem haben die Anwender bei der sanften Migration Zeit, sich langsam an die Neuerungen zu gewöhnen. Hinsichtlich des Vertriebs gilt: Bestandskunden können gehalten werden, da diese erkennen, dass das Produkt in die Zukunft migriert wird, und Neukunden können gut geworben werden, da die Zukunftsfähigkeit des Produkts darstellbar und das vorhandene Produkt vertrieblich einsetzbar ist.

Aus der internen Sicht des Software-Herstellers können anfangs zunächst die Entwickler eingesetzt werden, die den neuen Technologien offen gegenüberstehen. Entsprechend dem Migrationsfortschritt können dann immer mehr Mitarbeiter auf die neuen Technologien wechseln – die Akzeptanz bei den Anwendern und die Vertriebserfolge werden irgendwann alle Mitarbeiter mit ins Boot bringen. Zusätzlich kann bei der sanften Migration der Personaleinsatz im Bereich Entwicklung konstant gehalten werden. Weitere Hinweise zur Aufwandsabschätzung von Migrationsprojekten finden sich beispielsweise in [SPL03, Kapitel 17] und zu Einsparpotenzialen in [Sch04b].

24.3.2 Organisatorische Aspekte sanfter Migration

Der Software-Entwicklungsbereich unterliegt deutlichen Wandlungen – neue Technologien und Paradigmen tauchen schnell am Markt auf. Der Software-Hersteller muss deshalb insbesondere im Bereich der Entwicklung versuchen, ein innovationsfreundliches Klima zu schaffen. Besuch geeigneter Schulungen, Studium von Literatur und Fachzeitschriften sind als Zukunftsmotoren und nicht als Kostenfaktor anzusehen.

Es empfiehlt sich, viele Software-Entwickler am Technologiefindungsprozess zu beteiligen oder zumindest ein offenes Arbeitsklima herzustellen, in dem jederzeit Entscheidungen hinterfragt oder Kritik geübt werden kann.

Außerdem sollte frühzeitig bedacht werden, ob für die wichtigen Anfangsentscheidungen (Technologie- und Werkzeugauswahl, Definition des Migrationsweges) wegen zu geringer eigener Erfahrung nicht externe Hilfe in Anspruch zu nehmen ist.

Es sollte ein Mehrphasenmodell für die Schulungsblöcke der Software-Entwickler vorgedacht und kommuniziert werden (z. B. im 1. Jahr die ersten 10%, im 2. Jahr weitere 20% usw.), in dem sich jeder wiederfindet.

Die mit den neuen Technologien arbeitenden Software-Entwickler sollten dies in einem Umfang von mindestens 50%, besser aber 75% und mehr tun und sind entsprechend einzuplanen.

Die Software-Entwickler, die bereits an der sanften Migration arbeiten, sollten möglichst in gemeinsamen Räumen untergebracht werden, da dadurch der Sog der Alt-Technologien reduziert werden kann.

24.3.3 Psychologische Aspekte sanfter Migration

Software-Entwickler sind hochqualifizierte Mitarbeiter. Den höchsten Produktivitätszuwachs erreicht man durch Überzeugung in Bezug auf Ziel und Weg. Dies gilt im Allgemeinen für die Software-Entwicklung und auch für die sanfte Migration, denn zumindest ein Teil der Software-Entwickler sieht es häufig lieber, noch einmal neu anfangen zu können (Big Bang), da das entstehende Produkt dann »sauberer« gestaltet werden könnte.

Software-Entwickler sind möglichst entsprechend ihren persönlichen Zielen und Neigungen einzusetzen, die man bei geeigneten Mitarbeitergesprächen erfahren kann.

Erfahrene Software-Entwickler besitzen bei Verwendung der aktuellen (alten) Technologie eine hohe Produktivität. Sie sollen sich bei der sanften Migration in Technologien einarbeiten und mit Technologien beschäftigen, bei denen sie anfangs das Gefühl haben, nur eine sehr viel geringere Produktivität zu besitzen, da hierbei eben nicht mehr direkt mit der Programmierung begonnen wird. Auch hier ist immer wieder Überzeugungsarbeit erforderlich.

Neue Technologien brauchen i. d. R. auch einen neuen Entwicklungsprozess, der nach Problemerkennung erst die Anforderungsanalyse, dann die Anforderungsmodellierung und schließlich die Lösungsmodellierung notwendig macht, bevor mit der Codierung begonnen werden kann. Es empfiehlt sich, die Vorteile dieser Arbeitsweisen immer wieder zu kommunizieren.

24.4 Fazit

Software-Muster bedürfen einer empirischen Grundlage, s. dazu auch Kapitel 16. Das hier vorgestellte Migrationsprojekt war eine der Grundlagen des Dublo-Musters, das in Kapitel 10 vorgestellt wird. Über weitere Projekterfahrungen und eine Verallgemeinerung dieses Musters wird in [HBG+08] berichtet.