Klaus Turowski
(Hrsg.)

Tagungsband

4.Workshop Komponentenorientierte betriebliche Anwendungssysteme

11.Juni 2002

Augsburg

Gesellschaft für Informatik
Arbeitskreis 5.10.3
Komponentenorientierte betriebliche Anwendungssysteme

Tagungsleitung

Prof. Dr. Klaus Turowski
Lehrstuhl für Betriebswirtschaftslehre, insbesondere Wirtschaftsinformatik II
Universität Augsburg
Universitätsstraße 16, 86135 Augsburg
Phone: +49(821)598-4431; Fax : -4432
E-Mail: klaus.turowski@wiwi.uni-augsburg.de
URL: http://wi2.wiwi.uni-augsburg.de

Programmkomitee

Prof. Dr. S. Conrad
Ludwig-Maximilians-Universität München

Prof. Dr. R. Flatscher
Wirtschaftsuniversität Wien

Prof. Dr. U. Frank
Universität Koblenz-Landau

Prof. Dr. E. Ortner
Universität Darmstadt

Prof. Dr. C. Rautenstrauch
Otto-von-Guericke-Universität Magdeburg

Prof. Dr. E. Sinz
Universität Bamberg

Prof. Dr. K.Turowski (Vorsitz)
Universität Augsburg

ISSN 1619-9014

Vorwort

Dieser Tagungsband enthält die schriftlichen Beiträge zum *vierten Workshop komponentenorientierte betriebliche Anwendungssysteme* (WKBA 4), der vom GI-Arbeitskreis 5.10.3 „Komponentenorientierte betriebliche Anwendungssysteme“ und dem Lehrstuhl für Betriebswirtschaftslehre, insbesondere Wirtschaftsinformatik II der Universität Augsburg am 11. Juni 2001 in Augsburg veranstaltet wurde.

Das Thema des Workshops war die komponentenbasierte Gestaltung betrieblicher Anwendungssysteme. Ausgehend von dieser Themenstellung, wurde um die Einreichung von Beiträgen gebeten, die insbesondere folgende Themen aufgreifen:

- Standardisierung und Spezifikation von Fachkomponenten
- Komposition, Configuration Management
- Komponenten-Anwendungs-Frameworks, Business Objects
- Kopplungstechniken und (fachliche) Konfliktbehandlung
- Architekturen, Komponenten-System-Frameworks, Middleware
- Komponentenmärkte
- Domänenanalyse und Komponentenrepositories
- Spezifische Fragestellungen des Information Managements
- Praktische Erfahrungen

Aus den Einreichungen wurden fünf Beiträge ausgewählt, die aktuelle Forschungsergebnisse aus dem Bereich der Wirtschaftsinformatik darstellen oder Erfahrungen aus der betrieblichen Praxis dokumentieren. Jeder der eingereichten Beiträge wurde in einem anonymen Begutachtungsverfahren (double blind) von mindestens zwei Mitgliedern des Programmkomitees begutachtet.

Dieser Tagungsband enthält die schriftliche Fassung der zu Vortrag und Veröffentlichung angenommen Workshopbeiträge.

Abschließend sei allen gedankt, die durch die Einreichung eines Beitrags zum Gelingen des Workshops beigetragen haben. Besonderer Dank gebührt den Mitgliedern des Programmkomitees für die Begutachtung der eingegangenen Beiträge und Herrn Andreas Krammer für die Mitwirkung bei der Organisation des Workshops.

Augsburg, im Juni 2002

Klaus Turowski

Inhaltsverzeichnis

Die Spezifikation – kritischer Erfolgsfaktor der Komponentenorientierung

Sven Overhage[+]

[+] *Technische Universität Darmstadt, Fachgebiet Wirtschaftsinformatik I – Entwicklung von Anwendungssystemen, Institut für Betriebswirtschaftslehre, Hochschulstraße 1, D-64289 Darmstadt, Tel.: +49 (6151) 16-6688, Fax: -4301, E-Mail: overhage@bwl.tu-darmstadt.de, URL: http://www.winf1.bwl.tu-darmstadt.de*

Zusammenfassung. Die Schaffung einer konzeptionellen Terminologie, die von konkreten Technologien der Implementierung abstrahiert, und eines einheitlichen Spezifikationsrahmens zur Dokumentation der Außensicht von Komponenten sind wesentliche Voraussetzungen für den Erfolg der komponentenorientierten Anwendungsentwicklung. In diesem Beitrag wird vor allem die Bedeutung eines einheitlichen Spezifikationsrahmens betont, der als Basis für Vorgehensmodelle, Werkzeuge und die Entstehung von Komponentenmärkten dient und somit als kritischer Erfolgsfaktor zu betrachten ist. Dieser Spezifikationsrahmen basiert auf einer konzeptionellen Terminologie, die sich auf die am Markt vorhandenen Technologien für die Implementierung abbilden lässt. Sein Nutzen wird an verschiedenen Anwendungsbeispielen und einem Marktplatz für den Handel mit Software-Komponenten, der im Rahmen eines Dissertationsprojektes an der TU Darmstadt entwickelt wurde, veranschaulicht.

Schlüsselworte: Komponente; Framework; Spezifikation; Vorgehensmodell; Komponentenmarkt; Komponenten-Repository

1 Einleitung

Die Komponentenorientierung bringt für die Wirtschaftsinformatik und die betriebliche Anwendungsentwicklung eine Reihe von Vorteilen: Durch die Implementierung vieler einzelner Komponenten, die sich jeweils auf die Erfüllung einer zentralen Funktionalität konzentrieren, ist eine bessere Wiederverwendung von Software möglich. Dies führt im Allgemeinen zu einer Verbesserung der Produktivität (Entwicklung konzentriert sich auf Neues) sowie der Qualität (Komponenten sind bereits getestet) und somit zu einer Senkung der Kosten in der Anwendungsentwicklung.

Das Vorhandensein einer Vielzahl von Softwarebausteinen (Komponenten) ermöglicht außerdem (im Idealfall) die individuelle Konfiguration von Anwendungen nach den Vorgaben des Anwenders. So lässt sich beispielsweise eine Anwendung der Finanzbuchführung flexibel den gesetzlichen Erfordernissen anpassen, indem die Komponente „Bilanzierung“ ausgetauscht wird (und so Bilanzen nach verschiedenen Verfahren erstellt werden können: Handelsgesetzbuch – HGB, US Generally Accepted Accounting Principles – US-GAAP, International Accounting Standard – IAS etc.). Neben diesen Vorteilen für das Customizing ergeben sich durch die feinere Granularität von Anwendungen auch Verbesserungen für deren Wartung durch den Entwickler, die sich zu einer Wartung der von einer Änderung betroffenen

Komponenten vereinfacht. Schließlich ist das Vorhandensein eines ausgereiften Konzepts von Bauteilen (Komponenten) immer auch ein (akademisches) Kriterium für die Bewertung der Reife einer Ingenieursdisziplin.

Trotz dieser Vorteile hat sich die Umsetzung der komponentenorientierten Anwendungsentwicklung in der Praxis als schwierig erwiesen. So lassen sich zwar für das Design von grafischen Benutzungsoberflächen und einfache Dienste wie beispielsweise das Umrechnen zwischen verschiedenen Währungen häufig Komponenten finden. Komplexere Dienste mit domänenspezifischer Funktionalität (z.B. Komponenten zur Bilanzierung, Gewinn- und Verlustrechnung etc.) sind bislang jedoch selten als Bausteine implementiert worden, sondern meist Bestandteil komplexer monolithischer Anwendungen.

Eine Ursache für diese Schwierigkeiten der Komponentenorientierung in der Praxis ist der Mangel an technischen Standards, der die Konfigurierung von Anwendungen aus Softwarekomponenten immer dann erschwert, wenn diese in verschiedenen Technologien realisiert wurden. Hier zeichnet sich vor allem durch die Entwicklung XML basierter Web-Services (vgl. [Bett2001]) ein deutlicher Fortschritt ab. Die Anwendung dieser Technologie ermöglicht es dem Entwickler, aus beinahe jedem Stück (Legacy-) Software eine Komponente zu machen, die ihre Dienste für den Aufruf durch andere Komponenten an einer (technisch) standardisierten Schnittstelle zur Verfügung stellt.

Darüber hinaus fehlen bislang vor allem fachliche Standards in den (betrieblichen) Anwendungsbereichen (Domänen), was die Zusammenarbeit von Komponenten verschiedener Entwickler bzw. Hersteller erschwert (vgl. [Ortn1999a]). So ist eine Komponente für die Bilanzierung des einen Herstellers beispielsweise nur dann sinnvoll an die Anwendung zur Finanzbuchhaltung eines anderen anschließbar, wenn beide unter dem Begriff „Bilanzierung“ das Gleiche verstehen. Beim Austausch von Komponenten ist daher darauf zu achten, dass im Rahmen der Dokumentation die Erfassung von Fachtermini, die bei der Entwicklung einer Komponente implementiert wurden, mitsamt den zugehörigen Definitionen explizit ermöglicht wird. Auf diese Weise wird die Entstehung fachlicher Standards begünstigt und der Entwickler einer Anwendung in die Lage versetzt, Komponenten auch in Bezug auf ihre fachliche Eignung zu begutachten.

Ein weiterer Schlüsselfaktor für die Probleme in der Praxis ist das Fehlen einer (konzeptionellen) Komponenten-Terminologie mit einem einheitlichen Spezifikationsrahmen, ohne den kein gemeinsames Verständnis für die Beschreibung von Komponenten herzustellen ist. Ein solcher Spezifikationsrahmen bildet nicht nur die Grundlage für die Entwicklung von Werkzeugen (CASE Tools) und Vorgehensmodellen. Er ist vor allem die Basis für den Austausch von Komponenten zwischen Projekten (in einem unternehmensinternen Komponentenmarkt) oder gar Unternehmen (in einem offenen Markt) und damit die Entstehung eines Komponentenmarktes. Er ist somit von zentraler Bedeutung für den Erfolg der komponentenorientierten Anwendungsentwicklung (vgl. [Grif1998]).

In diesem Beitrag wird ein einheitlicher Spezifikationsrahmen für Komponenten vorgestellt, der auf einem Standardisierungsprozess aufsetzt, an dem Universitäten und Industrie gleichermaßen beteiligt waren (vgl. [Turo2002]). Er besitzt dadurch die Chance, in der komponentenorientierten (betrieblichen) Anwendungsentwicklung eine herausragende Geltung zu erlangen und von Werkzeugen auf breiter Ebene unterstützt zu werden.

Auf Basis dieses Spezifikationsrahmens werden anschließend ein einfaches Vorgehensmodell der komponentenorientierten Anwendungsentwicklung und mögliche Werkzeuge für dessen

Unterstützung beschrieben. Der Beitrag endet mit der Vorstellung eines elektronischen Marktplatzes für den Handel mit Software-Komponenten, der am Fachgebiet Wirtschaftsinformatik I der TU Darmstadt entwickelt wurde. Dieser Marktplatz implementiert den vorgeschlagenen Spezifikationsrahmen und integriert sich somit nahtlos in den Entwicklungsprozess, wie er hier beschrieben wird.

2 Eine konzeptionelle Komponenten-Terminologie

Im Zentrum der komponentenorientierten Anwendungsentwicklung steht der Begriff der „Komponente“ (Software-Komponente, Baustein), der in der Literatur vielfach (und nicht immer übereinstimmend) definiert wurde.

Einige Autoren (vgl. [Orfa1996]) definieren sie als objektorientiertes Konstrukt und verstehen Komponenten als spezielle Objekte, die neben Kapselung, Vererbung und Polymorphismus vor allem die Eigenschaft der Unabhängigkeit (von einem Programm, einer Programmiersprache bzw. einer Implementierung) besitzen. Für die Praxis ist diese Definition einer Komponente gleich aus mehreren Gründen zu eng gefasst. Zum einen erzwingt sie bei der Entwicklung von Komponenten den Einsatz objektorientierter Technologien (Vererbung, Polymorphismus) und lässt Komponenten, die auf nicht objektorientierten Technologien basieren (z.B. in Cobol implementiert wurden), nicht zu. Zum anderen erzwingt sie Komponenten mit einer sehr feinen Granularität (da Komponenten spezielle Objekte sind und diese häufig sehr feingranular implementiert werden). So sind insbesondere fertige Anwendungen als Komponenten (z.B. für integrierte Systeme) nicht zugelassen.

Allgemeinere Definitionen (vgl. [Souz1998] und [Szyp1998]) lassen in Bezug auf die einzusetzende Technologie mehr Spielraum. Obwohl es auch hier zahlreiche verschiedene Begriffsbildungen gibt, lässt sich doch ein gemeinsamer Kern an wichtigen Eigenschaften finden, der auf folgende Definition führt (in Anlehnung an [Turo2001]):

Eine Komponente besteht aus verschiedenartigen (Software-) Artefakten. Sie ist wieder verwendbar, abgeschlossen und vermarktbar, stellt Dienste über wohl definierte Schnittstellen zur Verfügung, verbirgt ihre Realisierung und kann in Kombination mit anderen Komponenten eingesetzt werden. Zur Zeit der Entwicklung einer Komponente sind diese Kombinationen noch nicht unbedingt vorhersehbar.

Zu den (Software-) Artefakten sind insbesondere ausführbarer (compilierter) Code oder Quellcode zu zählen, sowie darüber hinaus die Spezifikation, (Anwender-) Dokumentation und (automatisierte) Tests. Als abgeschlossen bezeichnet man eine Komponente, falls sie in der Lage ist, die von ihr erfüllte Aufgabe zu erfüllen, ohne auf andere Komponenten angewiesen zu sein (in der Regel lässt man dabei allerdings Abhängigkeiten in Bezug auf das Vorhandensein einer bestimmten Middleware zu). Setzt man die Abgeschlossenheit voraus, kann man eine Komponente als ein für sich selbst stehendes Gut verstehen, dass auf einem (offenen oder unternehmensinternen) Komponentenmarkt gehandelt werden kann (also vermarktbar ist). Nach dieser Definition spielt die Granularität für den Komponentenbegriff zunächst keine Rolle. Es kann somit Komponenten unterschiedlichster Granularität geben, von elementaren Komponenten bis hin zu Anwendungen, die (beispielsweise im Enterprise Application Integration – EAI) ebenfalls als Komponenten aufgefasst werden können.

2.1 Unterschiedliche Formen der Wiederverwendung

Eine besondere Aufmerksamkeit ist der Wiederverwendung von Komponenten während der Anwendungsentwicklung zu schenken, da es hierbei mehrere Arten mit unterschiedlichen Vor- und Nachteilen für den Anwender bzw. Hersteller gibt. Abbildung 1 zeigt diese verschiedenen Arten der Wiederverwendung schematisch.

	logisch	**physisch**
White-Box	Opensource	—
Black-Box	Enterprise Java Beans, Microsoft DCOM	XML Web-Services, CORBA

Abbildung 1 Wiederverwendung von Komponenten

Zunächst ist zu unterscheiden, ob für die Wiederverwendung einer Komponente der Quellcode zur Verfügung gestellt wird, oder ob lediglich kompilierte (also direkt ausführbare) Artefakte zur Verfügung stehen. Im ersten Fall lassen sich Komponenten wieder verwenden, indem ihr Quellcode auf den Anwendungsfall angepasst wird. Man nennt diese Wiederverwendung durch Anpassung auch White-Box-Wiederverwendung, da die Innensicht der Komponente dem Anwendungsentwickler bekannt und veränderbar ist. So ist es beispielsweise möglich, eine vorhandene Komponente zur Bilanzierung nach einem Verfahren (z.B. HGB) durch Veränderung des Quellcodes zu einer Komponente anzupassen, die nach den Vorschriften eines anderen Verfahrens (z.B. IAS) bilanziert. White-Box-Wiederverwendung tritt typischerweise bei der Verwendung von Codebibliotheken oder Opensource Software auf.

Im anderen Fall lassen sich Komponenten nur durch Auswahl einer für den Anwendungszweck geeigneten Variante wieder verwenden. Dem Anwendungsentwickler ist lediglich bekannt, was die Komponente tut, aber nicht wie sie es tut. Daher spricht man im Falle der Wiederverwendung durch Auswahl auch von Black-Box-Wiederverwendung. Wird also eine Komponente zur Bilanzierung nach einer bestimmten Vorschrift gesucht, braucht man eine geeignete Variante der Komponente „Bilanzierung“, die genau dieses leistet. Diese Variante muss von einem Hersteller am Markt zur Verfügung gestellt werden. Black-Box-Wiederverwendung sollte den Regelfall am Softwaremarkt darstellen.

In der Praxis entpuppen sich die vermeintlichen Vorteile der White-Box-Wiederverwendung, die meist höhere Flexibilität durch bessere Anpassung verheißen, schnell als Nachteil. Zur Anpassung wird einerseits komplexes Fach- und Implementierungswissen benötigt. Andererseits kann von neuen, vom Hersteller verbesserten Versionen der Komponente nicht mehr automatisch profitiert werden. Die Black-Box-Wiederverwendung entlastet den Anwendungsentwickler von diesen Nachteilen, setzt allerdings eine entsprechende Variantenzahl jeder Komponente voraus. Dies kann in einem funktionierenden Komponentenmarkt jedoch sicher erwartet werden, zumal es dem Hersteller von Komponenten gelingt, sein (im Quellcode verborgenes) Implementierungswissen durch den Vertrieb von Black-Box-Komponenten

besser zu schützen. Durch das Konzept der Black-Box-Wiederverwendung wird der Anwendungsentwickler im Idealfall zum Konfigurator, der Anwendungen aus Komponenten zusammensetzt.

Über die obige Diskussion hinausgehend lassen sich (zumindest für Black-Box-Komponenten) logische und physische Wiederverwendung unterscheiden. Logische Wiederverwendung liegt immer dann vor, wenn eine Komponente repliziert und vom Hersteller an den Anwendungsentwickler ausgeliefert wird, was derzeit dem Regelfall am Softwaremarkt entspricht. Von physischer Wiederverwendung wird hingegen gesprochen, wenn ein entfernter Aufruf der Komponente (die beim Hersteller oder Service Provider gespeichert ist) erfolgt. Diese Art der Wiederverwendung wird vor allem durch XML Web-Services und das Application Service Providing (ASP) weitere Verbreitung finden.

Für die physische Wiederverwendung spricht hauptsächlich die weniger kostenintensive Nutzbarkeit von (Software-) Diensten durch den Anwender (der weder Hardwareressourcen bereitstellen noch sich um die Installation kümmern muss) sowie die effizientere Wartbarkeit der Komponenten durch den Hersteller, da nur wenige Installationsbasen zu ändern sind. Diesen Vorteilen steht jedoch eine Reihe von Nachteilen gegenüber. So sind seitens des Herstellers (bzw. Anbieters) solcher Dienste zunächst Fragen der Skalierbarkeit und Verfügbarkeit zu klären, für die er eine gewisse Garantie geben muss (Quality of Service). Darüber hinaus besteht bei Anwendern immer dann eine starke Zurückhaltung, wenn sie für den Aufruf einer entfernten Komponente kritische Daten über eine potenziell unsichere Leitung übermitteln sollen. So ist es beispielsweise für die Nutzung einer Komponente zur Bilanzierung notwendig, kritische Zahlen aus der Gewinn- und Verlustrechnung als Daten zu übergeben. Nicht zuletzt diese Zurückhaltung bei den Anwendern ist einer der Gründe, warum die Entwicklung des ASP die Erwartungen bislang nicht erfüllen konnte.

Es steht zu erwarten, dass sich (am offenen Markt) für die logische und physische Wiederverwendung von Softwarekomponenten voraussichtlich unterschiedliche Geschäftsmodelle entwickeln werden (vgl. [McKi2001]). Die logische Wiederverwendung ist in der Regel durch das Vorhandensein von Pauschalpreisen (für Überlassung, Verkauf oder Vermietung) einer uneingeschränkten Nutzungserlaubnis gekennzeichnet, während sich für die physische Wiederverwendung eher nutzungsabhängige Entgelte (Pay per Use, Flat Fee Modelle mit Maximallast etc.) zu etablieren scheinen. Beim Konfigurieren einer Anwendung aus Bausteinen bzw. bei der Herstellung von Komponenten ist daher die Art der Wiederverwendung (und damit die Art und Weise der Einbindung von Komponenten in das System) stets mit Bedacht zu berücksichtigen bzw. zu planen.

2.2 Komponentenarten für die Anwendungsentwicklung

Die gegebene Definition des Begriffs „Komponente“ ist bewusst recht allgemein gehalten und umfasst viele verschiedene Arten von Komponenten. Soweit es für die Architektur einer (komponentenorientierten) Anwendung nützlich sein kann, sollen einige Arten näher erwähnt werden (vgl. Abbildung 2). Zunächst lässt sich unterscheiden, ob eine Komponente anwendungsinvariante (generische) oder anwendungsspezifische (domänenspezifische) Dienste erbringt. Diese Unterscheidung ist sinnvoll, da generische Dienste domänenübergreifend für die verschiedensten Anwendungen wieder verwendet werden können und somit zur Basissoftware einer Anwendung gehören. Ein Beispiel für eine solche (komplexe) generische Komponente ist ein Datenbank-Managementsystem, das als Basissoftware für eine Datenbankanwendung eingesetzt wird. Als domänenspezifische Komponente ist beispielsweise die schon

erwähnte Komponente „Bilanzierung“ im Rahmen einer Finanzbuchführung anzusehen. Im Folgenden werden generische Komponenten als „Systemkomponenten“ (System Components) bezeichnet, während domänenspezifische mit dem Begriff „Fachkomponenten“ (Expert Components) umschrieben werden.

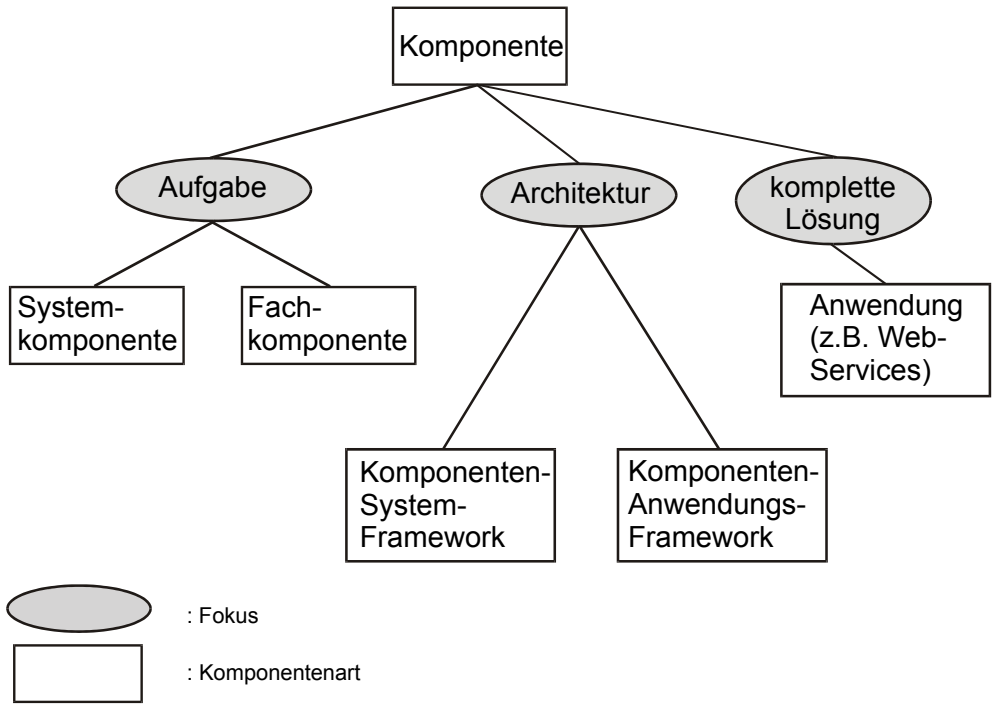


Abbildung 2 Komponentenarten

Unabhängig von diesem Kriterium gibt es noch eine zweite Unterscheidung im Hinblick auf Komponenten, die sich auf die Erfüllung einer Aufgabe konzentrieren, und solche, die eher ein Gerüst (also eine Architektur) für eine Anwendung zur Verfügung stellen. Im letzteren Falle spricht man statt von Komponenten auch von Frameworks. Ein solches Framework gibt also allgemeine Dienste und Regeln für das Zusammenwirken von Komponenten vor. Auch hier lassen sich generische (Komponenten-Systemframeworks bzw. System Frameworks) und domänenspezifische Frameworks (Komponenten-Anwendungsframeworks bzw. Application Frameworks) unterscheiden. Ein generisches Framework ist z.B. ein Betriebssystem oder eine Middleware wie die Technologie der Web-Services bzw. die eines EJB-Applikationsservers. Beispiele für ein anwendungsspezifisches Framework (aus der Domäne der betrieblichen Anwendungen) sind das San Francisco Framework von IBM (vgl. [IBM2000]) bzw. die entsprechenden CORBA Domain Interfaces.

Somit besteht eine komponentenorientierte Anwendung also ggf. aus folgenden Komponentenarten (in Anlehnung an [Raut2001]): einem Komponenten-Systemframework (das die Middleware-Schicht bildet), Systemkomponenten (die einzelne generische Dienste bereitstellen), einem Komponenten-Anwendungsframework (das allgemeine Dienste für eine Anwendungsdomäne bereitstellt und somit die Anwendungsplattform bildet) sowie aus Fachkomponenten (die spezialisierte Dienste der Anwendungsdomäne bereitstellen).

3 Ein einheitlicher Spezifikationsrahmen

Der Erfolg der komponentenorientierten Anwendungsentwicklung hängt entscheidend von einer einheitlichen, herstellerübergreifenden Spezifikation der Komponenten ab. Diese Spezifikation ist die Basis, auf der geeignete Komponenten ausgewählt und in die Anwendungsentwicklung integriert werden können. Im Rahmen eines Arbeitskreises der Gesellschaft für Informatik, in dem Vertreter aus der Industrie und der Universitäten zusammenkommen, wurde ein Vorschlag für die Vereinheitlichung der Spezifikation von Fachkomponenten entwickelt (vgl. [Turo2002]), der sich leicht zu einem Vorschlag für die einheitliche Spezifikation aller hier genannten Komponentenarten verallgemeinern lässt.

Kern dieses Vorschlags ist ein Spezifikationsrahmen, der neben wichtigen Aspekten einer Komponente auch Empfehlungen für den Einsatz konkreter Beschreibungssprachen (die am Markt bereits etabliert sind) vorgibt. Er dürfte daher (neben Universitäten) vor allem für die Abteilungen der betrieblichen Anwendungsentwicklung, die Standards und Methoden für die Anwendungsentwicklung vorgeben, sowie für Werkzeughersteller von Interesse sein.

Das Ziel eines Spezifikationsrahmens ist die möglichst vollständige Beschreibung der Außensicht einer Komponente, so dass ihre Eignung für einen Einsatz in der Anwendungsentwicklung möglichst allein auf Grund dieser Dokumentation (und ohne einen Testlauf) beurteilt werden kann. Dazu werden in der Praxis häufig Muster verwendet, welche entweder die Struktur oder den (fachlichen) Inhalt einer Komponente beschreiben. Eine verbreitete Darstellung von Mustern findet sich in abstrakten Datentypen, die sich auf die Spezifikation der Schnittstelle, der Vor- und Nachbedingungen für einzelne Dienste sowie die Angabe von Invarianten konzentrieren (vgl. [Grif1998]). Dort werden verschiedene Aspekte einer Komponente zusammen (und dadurch recht unübersichtlich) spezifiziert. Darüber hinaus reichen die gemachten Angaben für eine vollständige Beschreibung der Außensicht meist nicht aus. Der durch den Arbeitskreis standardisierte Spezifikationsrahmen greift diese Kritikpunkte auf und identifiziert zunächst verschiedene Aspekte einer Komponente, die sich den thematischen Bereichen „Administration“ (White bzw. Yellow Pages), „Konzeption (fachliche Semantik)“ (Blue Pages) und „Technik“ (Green Pages) zuordnen lassen. Durch die Gliederung in Aspekte wird zum einen eine Komplexitätsreduktion erreicht (da Spezifikationen jeweils thematisch zugeordnet werden können). Darüber hinaus ist dieser Spezifikationsrahmen ggf. um zusätzliche Aspekte erweiterbar. Auf diese Weise können zukünftige Versionen kompatibel gehalten werden. Abbildung 3 zeigt eine Übersicht, in der die einzelnen Aspekte zusammengefasst sind.

Die Administration umfasst zwei Aspekte, den Vermarktungs- und den Qualitätsaspekt. Der Vermarktungsaspekt spezifiziert zunächst allgemeine Informationen über eine Komponente. Dort werden z.B. der Hersteller, der Lieferumfang, die Version, Art der Wiederverwendung, die Komponentenart, die Systemanforderungen einer Komponente sowie die Plattform, für die sie entwickelt wurde, beschrieben. Darüber hinaus werden an dieser Stelle die Dienste einer Komponente klassifiziert, indem ihnen eine Anwendungsdomäne (Finanzbuchführung) bzw. eine generische Funktion (Datenverwaltung) zugeordnet wird, je nachdem, ob es sich um eine Fach- oder Systemkomponente handelt.

Der Qualitätsaspekt umfasst Aussagen über das Leistungsverhalten einer Komponente. Hierzu gehören Aussagen zur Antwortzeit, Lastverhalten, Quality of Service etc. Die Aspekte der Administration ermöglichen also zunächst einen Überblick über die Komponente. Sie dienen dem Entwickler, der nach einer passenden Komponente sucht, als ersten Anlaufpunkt. Von

dort ausgehend können weitergehende Aspekte für die Auswahl in Betracht gezogen werden. Dabei geht es zunächst um die Klärung fachlicher Aspekte, bevor schließlich technische Aspekte im Mittelpunkt seines Interesses stehen.

Hinter dem Schlagwort „Konzeption“ verbirgt sich dann auch die Beschreibung dieser fachlichen Semantik, also der thematischen Inhalte einer Komponente. Diese Inhalte sind im Falle einer Fachkomponente bzw. eines Komponenten-Anwendungsframeworks objektsprachliche Fachbegriffe (also Fachbegriffe eines Anwendungsbereichs), im Falle einer Systemkomponente bzw. eines Komponenten-Systemframeworks metasprachliche (generische) Termini. Die Beschreibung umfasst wiederum zwei Aspekte, den Aufgaben- und den Terminologieaspekt. Im Aufgabenaspekt wird vor allem die von der Komponente unterstützte Aufgabe definiert und ggf. in Teilaufgaben zerlegt. Bei Fachkomponenten steht an dieser Stelle eine Aufgabe aus der Anwendungsdomäne (z.B. Bilanzieren), bei Systemkomponenten eine generische Aufgabe (z.B. Daten persistent speichern).

Der Terminologieaspekt dient der zentralen Erfassung aller bei der Implementierung der Komponente verwendeten Fachbegriffe nebst Definition in einem Lexikon. Auf diese Weise kann nachvollzogen werden, ob eine Komponente im Rahmen einer Anwendungsentwicklung auch begrifflich für eine Auswahl in Frage kommt (vgl. [Ortn2000]). Insbesondere auf Grund regelmäßig fehlender fachlicher Standards ist die Spezifikation der fachlichen Semantik unablässig, wenn Entwickler sich vor möglichen (fachlichen) Überraschungen bei der Verwendung einer Komponente schützen wollen. Somit spielt sie bei der Auswahl einer Komponente eine zentrale Rolle.

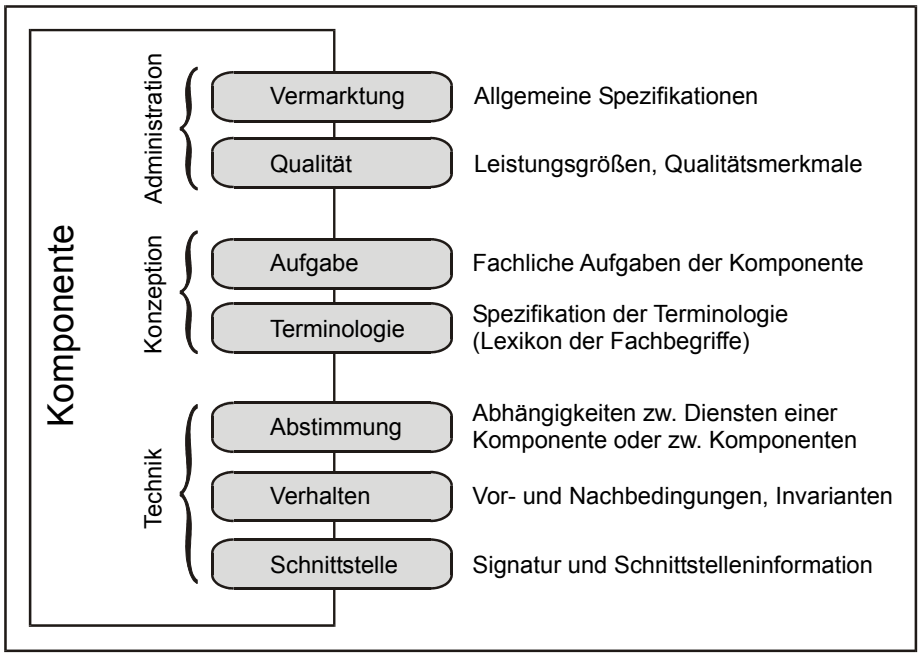


Abbildung 3 Einheitlicher Spezifikationsrahmen

Mit dem Schlagwort „Technik“ sind schließlich alle Aspekte zusammengefasst, die für die technisch korrekte Einbindung einer Komponente in eine Anwendung zu berücksichtigen sind. Dabei steht zunächst im Schnittstellenaspekt die Beschreibung der Schnittstelle einer

Komponente im Mittelpunkt. In diesem Aspekt wird dem Entwickler mitgeteilt, wie die einzelnen Dienste an der Schnittstelle benannt wurden (Methodennamen, Parameternamen und Datentypen), durch welche Dienste eine Aufgabe (aus dem Aufgabenaspekt) jeweils erfüllt wird, welche Ausnahmezustände dabei auftreten können etc. – es geht dabei also vornehmlich um die Beschreibung der Syntax. Hierfür werden vorzugsweise herstellerunabhängige Sprachen wie die Interface Description Language (IDL) bzw. die Web Services Description Language (WSDL) herangezogen (vgl. [Turo2002]).

Das Systemverhalten einer Komponente wird im gleichnamigen Verhaltensaspekt näher beschrieben. Hier geht es vor allem um die Formulierung von Invarianten und Vor- bzw. Nachbedingungen, die für einzelne Dienste der Schnittstelle gelten.

Vervollständigt wird die technische Spezifikation durch den Abstimmungsaspekt, der angibt, welche Abhängigkeiten es zwischen den einzelnen Diensten einer Komponente oder von den Diensten einer anderen Komponente gibt. Ein Beispiel für eine Abhängigkeit, von der die Komponente „Bilanzierung“ betroffen ist, wäre z.B. die (vereinfacht notierte) Aussage „…vor Aufruf der Bilanzierung muss eine Gewinn- und Verlustrechnung durchgeführt worden sein.“. Dies ist insbesondere dann hilfreich, wenn die Schnittstelle eine Vielzahl von Diensten anbietet, die teilweise voneinander abhängig sind (was in der Praxis nicht selten der Fall ist).

Wie gezeigt wurde, unterstützt der beschriebene Spezifikationsrahmen die ausführliche Dokumentation der Außensicht einer Komponente durch insgesamt sieben Aspekte. Diese stellen zugleich den Kern des Standards dar, der in jedem auf diesem basierenden Werkzeug unterstützt werden sollte. Darüber hinaus ist er für Erweiterungen auf Grund seines aspekteorientierten Aufbaus offen. Genauer ausgeführt sind die einzelnen Aspekte und die jeweils zum Einsatz empfohlenen Sprachen in [Turo2002]. Daher soll an dieser Stelle auf weitere Details verzichtet und stattdessen einige Anwendungen des Spezifikationsrahmens vorgestellt werden.

4 Vorgehensmodell und Werkzeugunterstützung

Der vorgenannte Spezifikationsrahmen dient nicht nur der Beschreibung von Komponenten, sondern ist zugleich auch die Basis für die Entwicklung von Vorgehensmodellen und die Implementierung von Werkzeugen. Im Folgenden zeigt ein einfaches, jedoch somit auch allgemein verwendbares Vorgehensmodell, wie unter Zuhilfenahme des Spezifikationsrahmens komponentenorientierte Anwendungen flexibel entwickelt werden können. Dabei zeigt sich die zentrale Bedeutung eines standardisierten Spezifikationsrahmens sowie einer entsprechenden Komponenten-Terminologie, die in das Vorgehensmodell aufgenommen wurden.

4.1 Das Multipfad-Vorgehensmodell

An der komponentenorientierten Anwendungsentwicklung sind IT-Spezialisten in den Rollen „Komponentenhersteller“, „Komponentenkonsument“ und „Komponentenmanager“ beteiligt. Der Komponentenhersteller implementiert und vertreibt Komponenten. Der Komponentenkonsument sucht für ihn passende Komponenten, um daraus Anwendungen zu bauen. Er wird daher im Folgenden auch als Anwendungsentwickler bezeichnet. Der Komponentenmanager verwaltet Komponenten in einer zentralen Ablage (Komponenten-Repository).

Im Falle eines unternehmensinternen Komponentenmarkts sind alle drei Parteien Angehörige des Unternehmens und können sogar in Personalunion auftreten. Darüber hinaus wäre es

denkbar, dass die Anwendungsentwickler den Fachbereichen des Unternehmens zugeordnet sind, die Komponentenhersteller hingegen eine zentrale (IT-) Abteilung bilden. Der Komponentenmanager administriert in diesem Fall ein Unternehmens-Repository. Im Falle eines offenen Marktes können die Rollen jedoch auseinander fallen. Der Komponentenmanager ist in diesem Fall der Betreiber eines offenen Komponenten-Verzeichnisses (Kataloges) oder eines Komponenten-Marktplatzes (Handelsplatzes). Zusätzlich setzen Produzent und Konsument idealerweise jeweils ein eigenes Unternehmens-Repository ein, um die entwickelten bzw. verwendeten Komponenten zu dokumentieren.

Ein einfaches Vorgehensmodell für die Anwendungsentwicklung gliedert sich in sechs Phasen: „Voruntersuchung“, „Fachentwurf mit Klassifikation“, „Systementwurf“, „Implementierung“, „Konfigurierung“ und „Stabilisierung“. Bei der Anwendungsentwicklung können diese auf unterschiedlichen Pfaden durchlaufen werden (vgl. Abbildung 4). So sind bei der Entwicklung von Individualsoftware alle diese Phasen zu durchlaufen. Für die komponentenorientierte Anwendungsentwicklung ergeben sich jedoch unter Umständen Abkürzungen, die zu einer Zeit- und Kostenersparnis führen.

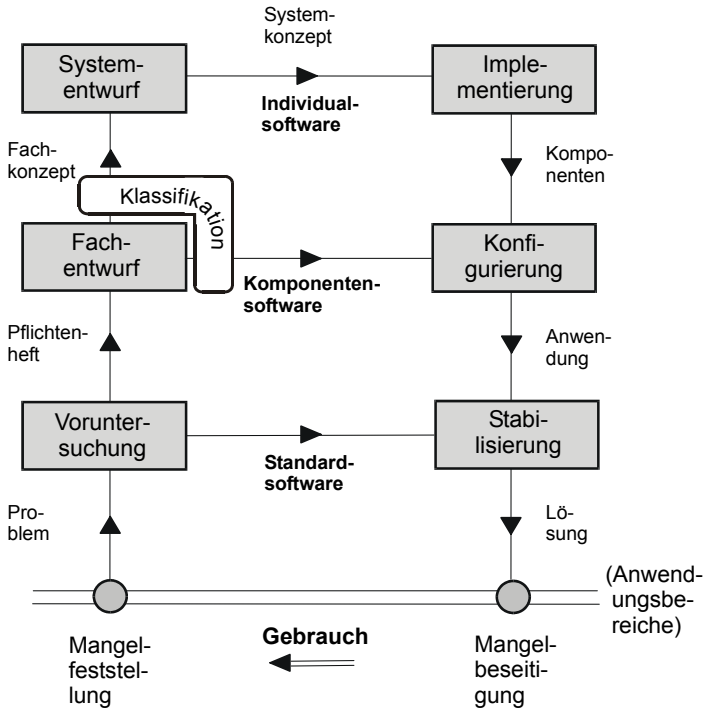


Abbildung 4 Flexibles Multipfad-Vorgehensmodell der komponentenorientierten Anwendungsentwicklung

Während der Voruntersuchung wird zunächst vom Anwendungsentwickler ein Pflichtenheft erstellt, das neben der Aufgabenstellung auch eine Machbarkeitsstudie enthält. Gelingt es ihm, zur Erfüllung der Aufgabenstellung eine passende Standardsoftware einzusetzen, kann er die Entwicklung bereits hier beenden und gelangt in die Stabilisierung. Anderenfalls begibt er sich in die Phase des fachlichen Entwurfs, in der er (im Dialog mit den Endanwendern) fachliche Aussagen über die zu schaffende Anwendung ermittelt. Die Phase endet mit der Erstellung des Fachkonzepts, das aus einer Aussagensammlung auf der Basis geklärter Fachbegriffe besteht. Mit dieser Aussagensammlung wird der Anwendungsentwickler nun in die Lage ver-

setzt, passende Komponenten für eine Anwendung zu suchen. Dazu muss er die Aussagen jedoch derart ordnen, dass sie ihm konkrete Vorgaben für die Suche nach möglichen Komponenten geben. Man nennt diesen Schritt auch Klassifikation.

Bei der Klassifikation spielt nun der verwendete Spezifikationsrahmen bzw. die verwendete Komponenten-Terminologie eine Rolle, da durch diese ein Klassifikationsschema determiniert werden kann. Durch dieses wird festgelegt, nach welchen Aspekten die Aussagen aus dem Fachkonzept für die Suche nach Komponenten zu gliedern sind. Abbildung 5 zeigt ein zweidimensionales Klassifikationsschema, das Aussagen einmal in solche über die (in dieser Arbeit genannten) verschiedenen Komponentenarten differenziert. Zum anderen werden die Aussagen den verschiedenen Aspekten des beschriebenen Spezifikationsrahmens zugeordnet.

Mit einem solchen Schema wendet sich der Anwendungsentwickler anschließend an einen Komponentenmanager, der ihn auf dessen Basis bei der Suche nach geeigneten Komponenten unterstützen kann. Gelingt es dem Anwendungsentwickler dabei, alle nötigen Komponenten zu finden (und zu erwerben), kann er aus diesen seine Anwendung konfigurieren und anschließend testen. Gelingt ihm dies nicht, muss er den Bau der fehlenden Komponenten bei einem Produzenten in Auftrag geben (oder selbst besorgen). Nur in dem Fall wird die Phase „Implementierung“ noch durchlaufen.

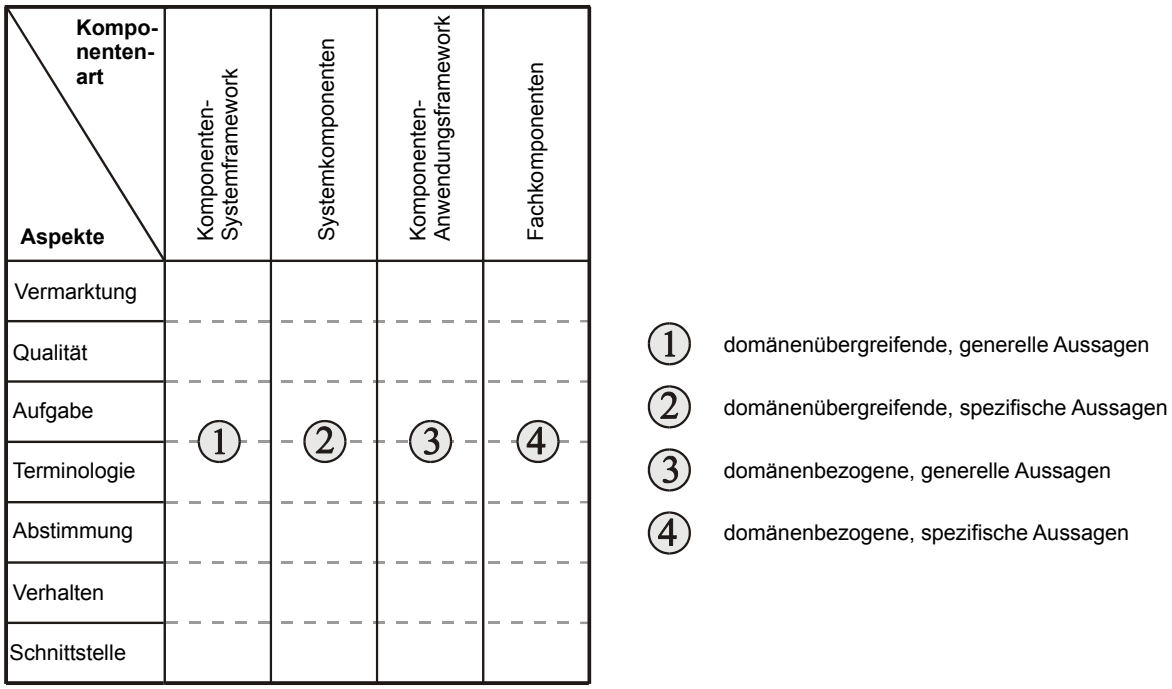


Abbildung 5 Klassifikationsschema für die Suche nach Komponenten

Schon aus diesem einfachen Vorgehensmodell wird ersichtlich, dass die komponentenorientierte Anwendungsentwicklung im Allgemeinen zu einem erheblich flexibleren Entwicklungsprozess führt. Detaillierte Organisations- und Vorgehensmodelle für die komponentenorientierte Anwendungsentwicklung können auf seiner Basis von den Unternehmen individuell erarbeitet werden (einen Ansatz hierfür liefert [Sahm2000]).

4.2 Werkzeugunterstützung durch Komponenten-Repositories

Die komponentenorientierte Anwendungsentwicklung lässt sich an vielen Stellen von Werkzeugen unterstützen. Von zentraler Bedeutung sind dabei diejenigen für die Verwaltung und Administration der Komponenten in einem Unternehmen. Diese Werkzeuge, die Komponenten-Repositories genannt werden, dokumentieren, welche Komponenten entwickelt bzw. in welchen Anwendungen verwendet wurden. Diese Funktionalität ist vor allem dann sinnvoll, wenn eine Komponente einen Versionswechsel erfährt und die betroffenen Systeme gewartet werden müssen. Schließlich dienen sie als Teilelager für die Anwendungsentwicklung und unterstützen Entwickler bei der Suche nach geeigneten Komponenten. Sowohl für den Komponentenproduzenten als auch den -konsumenten ist der Einsatz eines Komponenten-Repository daher sinnvoll. Ein Komponenten-Repository ist eine speziell zur Administration von Komponenten entwickelte Datenbankanwendung. Hierzu liefert sie ein entsprechendes Metaschema mit (vgl. [Ortn1999b]), das die Art der Dokumentation bestimmt. Der im Rahmen dieser Arbeit beschriebene Spezifikationsrahmen ist daher auch für die Entwicklung solcher Werkzeuge von Bedeutung. Er kann als Kern eines solchen Metaschemas dienen, das ggf. um weitere Aspekte (wie die Dokumentation der Anwendungen) zu ergänzen ist.

Über die beschriebene Funktionalität hinaus wäre vor allem für die Phasen „Klassifikation“ und „Konfiguration“ eine Unterstützung des Entwicklers durch das Komponenten-Repository wünschenswert. So ist es denkbar, dass die Auswahl einer optimalen Kombination von Komponenten für eine zu entwickelnde Anwendung auf Basis des Klassifikationsschemas automatisiert erfolgt. Hinter dieser Aufgabenstellung verbirgt sich ein komplexes Optimierungsproblem, das sich mit Methoden des Operations Research (zumindest annähernd) lösen lässt. Leider kann man zeigen, dass eine exakte optimale Lösung nur für Systeme, die aus relativ wenigen Komponenten zusammenzusetzen sind, mit vertretbarem Aufwand zu ermitteln ist (vgl. [Kauf2000]). Somit kommen hierfür bevorzugt heuristische Verfahren zum Einsatz, die nicht immer die beste Lösung liefern. Ein viel versprechender Ansatz, der zudem noch garantiert optimale Lösungen liefert, basiert auf dem Einsatz von Variantenstücklisten (vgl. [Wede1981]) und ist derzeit Gegenstand eines Forschungsprojektes. Die Konfigurierung eines optimalen Systems aus verschiedenen alternativen Komponenten lässt sich jedoch auch ohne Automatisierung durch Methoden wie den Morphologischen Kasten (vgl. Abbildung 6) hinreichend unterstützen. Dieser ermöglicht die Analyse verschiedener Lösungen und die Auswahl einer optimalen Kombination von Komponenten.

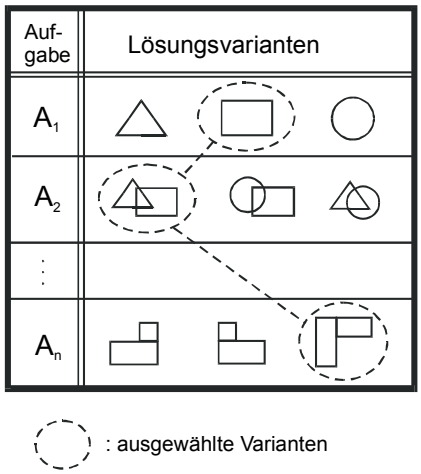


Abbildung 6 Morphologischer Kasten

5 CompoNex – ein Marktplatz für den Handel mit Software-Komponenten über das Internet

Eine weitere wesentliche Voraussetzung für den Erfolg der komponentenorientierten Anwendungsentwicklung ist das Vorhandensein eines Komponentenmarktes, der für einen reibungslosen Austausch von Software-Komponenten zwischen Anbietern und Nachfragern sorgt. Hier zeichnet sich insbesondere durch die sich etablierende Technologie der XML Web-Services ein spürbarer Fortschritt ab. Bestandteil dieser Technologie ist UDDI (Universal Description, Discovery and Integration), ein Standard für offene Verzeichnisse, in denen Web-Services registriert werden können (vgl. [UDDI2000]). Damit existiert für das Auffinden von Komponenten, die als XML Web-Service ansprechbar sind, bereits ein zentrales Komponenten-Repository. Ein Nachteil der Verzeichnisse, die auf dem Standard UDDI basieren, ist jedoch die Tatsache, dass sie sich auf Komponenten des Typs XML Web-Service beschränken und darüber hinaus lediglich die technische Schnittstelle spezifizieren (also nur den Schnittstellenaspekt bei der Spezifikation berücksichtigen). Hierdurch wird die Beurteilung einer Komponente durch den Entwickler erschwert.

Am Fachgebiet Wirtschaftsinformatik I der TU Darmstadt wurde im Rahmen eines Dissertationsprojektes (prototypisch) ein elektronischer Marktplatz für den Handel mit Software-Komponenten jeder Art entwickelt, der in einem Spin-Off vermarktet wird. Dieser Marktplatz trägt den Namen CompoNex (Component Exchange) und berücksichtigt die Tatsache, dass es sich bei Software-Komponenten um erklärungsbedürftige Güter handelt, deren Handel durch spezielle Dienste unterstützt werden sollte. So implementiert er als Basis des Komponenten-Kataloges den hier vorgestellten Spezifikationsrahmen und verfügt über eine geordnete (aspektorientierte) Darstellung der Außensicht von Komponenten. Er unterstützt darüber hinaus den Vertrieb sämtlicher Komponentenarten, die in Abschnitt 2 vorgestellt wurden und beschränkt sich dabei nicht auf Komponenten einer bestimmten Technologie. Durch die Implementierung des Spezifikationsrahmens integriert sich der Marktplatz in das geschilderte Vorgehensmodell und ermöglicht die Suche nach Komponenten auf Basis des Klassifikationsschemas. Darüber hinaus diente die Implementierung während der Standardisierungsarbeiten zugleich als „Proof of Concepts“.

5.1 Architektur und Dienste am Marktplatz

CompoNex basiert auf einem datenbankgestützten Komponenten-Repository, das den hier erwähnten Spezifikationsrahmen als Metaschema implementiert und für jede Komponente die Dokumentationsaspekte „Vermarktung“, „Qualität“, „Aufgabe“, „Terminologie“, „Abstimmung“, „Verhalten“ und „Schnittstelle“ zur Verfügung stellt. Es unterscheidet darüber hinaus zwischen Fach- und Systemkomponenten bzw. Komponenten-Anwendungsframeworks und Komponenten-Systemframeworks und gibt Auskunft darüber, ob eine Komponente logisch oder physisch wieder verwendet werden kann.

Der darauf aufbauende Komponentenkatalog kann von einem Marktplatzbesucher schrittweise durchstöbert werden, wobei er zunächst die gewünschte Komponentenart wählt und sich anschließend eine Anwendungsdomäne bzw. generische Funktionalität aussucht. Auf diese Weise kann er sich zu den gewünschten Komponenten vorarbeiten, wobei konzeptionelle (und nicht technologische) Überlegungen wie die Komponentenart oder die Funktionalität (Aufgabe) der Komponente im Vordergrund stehen.

Darüber hinaus bietet der Marktplatz zwei Suchfunktionen, um gezielt nach einzelnen Komponenten zu suchen. Die Volltextsuche ermöglicht die einfache Angabe von Schlüsselwörtern, die dann in den Spezifikationsaspekten „Vermarktung“, „Aufgabe“ und „Terminologie“ gesucht werden. Eine parametrisierbare Suche ermöglicht die Angabe detaillierter Suchwerte in den vorgenannten Aspekten, so dass die Suche stärker eingeschränkt werden kann. Auf diese Weise wird es zum Beispiel möglich, nach Komponenten, die die Aufgabe „Bilanzierung“ implementieren, als Enterprise Java Bean gefertigt sind und von SAP stammen, zu suchen.

Die Suchfunktion ist dabei mit semantischer Intelligenz ausgestattet, so dass sie in der Lage ist, semantisch verwandte Schlüsselwörter gewichtet in die Suche mit einzubeziehen. Sie wird Bezug nehmend auf das vorgenannte Beispiel erkennen, dass Komponenten der Domäne „Finanzbuchführung“ in die Suche aufzunehmen sind und somit auch Komponenten, welche die Aufgabe „Gewinn- und Verlustrechnung“ implementieren, einbeziehen (und mit einem niedrigeren Rang versehen).

Aufbauend auf dieser Suchfunktion bietet der Marktplatz einen Subskriptionsdienst (basierend auf einem Publish/Subscribe Algorithmus), über den Suchabfragen periodisch ausgeführt und deren Ergebnisse dem Besucher als Abonnement (per E-Mail, SMS etc.) zugestellt werden können. Auf diese Weise kann er automatisch auf dem aktuellen Stand gehalten werden.

Bei der Implementierung des Marktplatzes für den Komponentenhandel war ferner zu berücksichtigen, dass die zugrunde liegenden Geschäftsprozesse möglichst direkt unterstützt werden. So erfolgt die Benutzung des Marktplatzes vorwiegend während des Entwicklungsprozesses (in der Phase „Fachentwurf mit Klassifikation“), wodurch eine direkte Integration in die CASE Tools des Anwenders (beispielsweise sein Entwicklungswerkzeug oder sein Komponenten-Repository) erforderlich wurde. CompoNex wurde daher als XML Web-Service implementiert und stellt somit selbst eine Software-Komponente dar, die in andere Anwendungen eingebunden werden kann.

Im Rahmen der Implementierung wurde der Marktplatz sowohl auf die Benutzung über eine webbasierte Schnittstelle (die unter www.componex.biz verfügbar ist) ausgelegt sowie exemplarisch in das CASE Tool Microsoft Visual Studio .NET über ein Plug-In integriert. Abbildung 7 zeigt diese beiden Benutzungsoberflächen in der Zusammenschau. Anwendungen von

Drittanbietern steht zur Integration des Marktplatzes die (in ihrer Außensicht nach dem Spezifikationsrahmen) dokumentierte Schnittstelle des CompoNex Web-Service zu Verfügung. Auf diese Weise ist der Komponentenmarktplatz ohne Medienbruch zu dem Zeitpunkt ansprechbar, zu dem er auch tatsächlich nachgefragt wird.

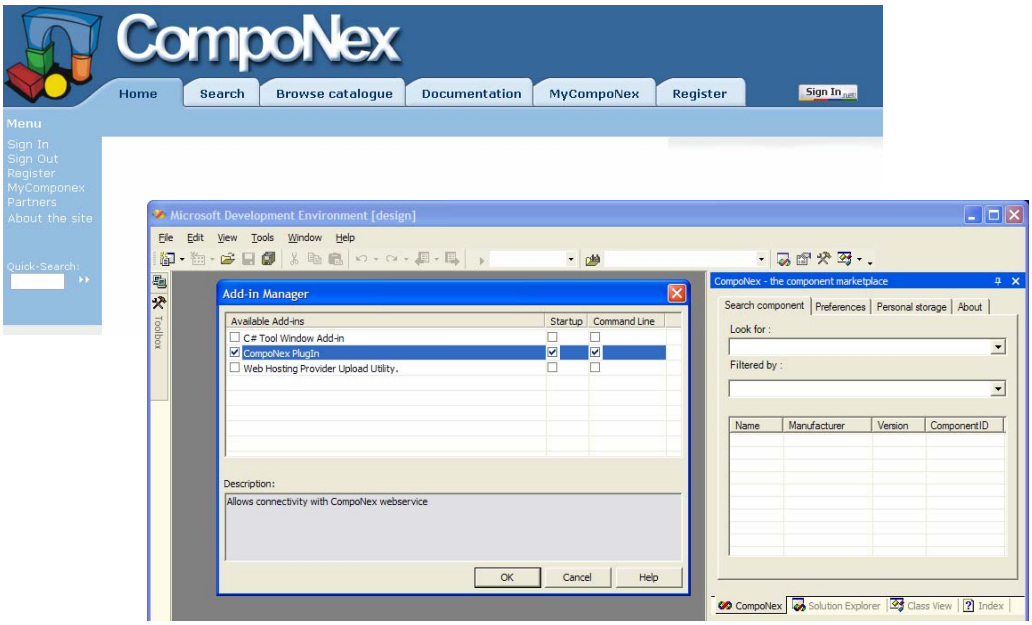


Abbildung 7 Web-Benutzungsschnittstelle und CASE Tool Integration

Aus den vorgenannten Anforderungen während der Systementwicklung ergibt sich für den Marktplatz eine komplexe Architektur, die (auszugsweise) in Abbildung 8 dargestellt ist. Die gezeigte Architektur setzt auf einem komponentenorientierten Middleware-Framework für die Entwicklung von E-Commerce-Anwendungen auf, das unter dem Namen E-NOgS (Electronic New Organon {Server, Service, Servant}) zuvor bereits am Fachgebiet entwickelt wurde.

Darüber hinaus verfügt der Marktplatz über ein differenziertes Geschäftsmodell (vgl. [Merz2002]), das Zusatzdienste wie den Subskriptionsdienst gegen eine Gebühr offeriert und dem Anbieter von Komponenten auf einer Provisionsbasis erfolgreich verkaufte Güter in Rechnung stellt. Darüber hinaus kann (auf Wunsch des Anbieters) die Speicherung von Komponenten in ein eigenes Komponentenlager des Marktplatzes vorgenommen werden. Somit werden insbesondere kleinere Komponentenanbieter vom Aufbau einer eigenen Vertriebsorganisation entlastet und können in den Komponentenmarkt eintreten.

Durch die detaillierte Spezifikation der Komponenten und die angebotenen Zusatzdienste hebt sich der Marktplatz von anderen am Markt befindlichen Komponentenkatalogen (wie beispielsweise UDDI) ab und besitzt somit auch die Chance, sich erfolgreich am entstehenden Komponentenmarkt zu etablieren. Für seine Konzeption und die Implementierung wurde er im Rahmen eines Industrie-Wettbewerbs der Firma Microsoft bereits mit einem Preis ausgezeichnet.

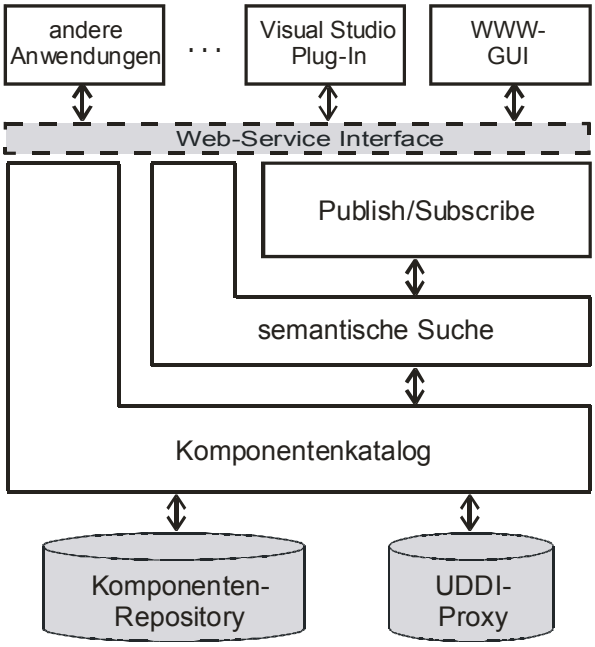


Abbildung 8 Architektur von CompoNex (in Auszügen)

6 Ausblick

Im Rahmen dieses Beitrags wurde gezeigt, dass die einheitliche Spezifikation von Komponenten eine Schlüsselbedeutung für die komponentenorientierte Anwendungsentwicklung besitzt. Mit der Etablierung von Standards in den Themenfeldern „Spezifikation“ und „Komponententechnologie“ (Web-Services als Kommunikationsplattform) lassen sich wesentliche Fortschritte erreichen. Lediglich auf dem Gebiet der fachlichen Verschiedenheiten (Terminologien) sind Standards bislang kaum in Sicht. Einen möglichen Ausweg aus dieser Situation zeigt indes der hier geschilderte Spezifikationsrahmen, der die verwendete Semantik explizit dokumentiert und dadurch ggf. den Bau von Übersetzern ermöglicht. Darüber hinaus zeichnen sich in einigen Branchen (wie dem deutschen Versicherungswesen) Bewegungen mit dem Ziel ab, fachliche Standards im Hinblick auf die Schaffung eines offenen Komponentenmarkts zu etablieren.

Die Komponentenorientierung benötigt für ihren Durchbruch eine umfassende Unterstützung durch Werkzeuge (einige Aufgaben wurden in Abschnitt 4 bereits angesprochen) und Methoden. Insbesondere die Schaffung einer Modellierungssprache, welche die hier genannten (konzeptionellen) Komponentenarten als Architekturbestandteile unterstützt, wäre (beispielsweise als Erweiterung der UML) wünschenswert. Darüber hinaus bleibt im Hinblick auf die Verbreitung einer eher konzeptionellen Komponenten-Terminologie eine gewisse Überzeugungsarbeit gegenüber den IT-Experten zu leisten, die bislang eine eher implementierungsnahe Terminologie verwenden. Diese ist (analog zu dem bekannten Beispiel der Datenverwaltungssysteme) jedoch für die Architekturbeschreibung und Modellierung unzureichend.

Literatur

[Bett2001] *Bettag, U.:* Web-Services. In: Informatik Spektrum 24 (2001) 5, S. 302 – 304.

[Grif1998] *Griffel, F.:* Componentware – Konzepte und Techniken eines Softwareparadigmas. dpunkt-Verlag, Heidelberg 1998.

[IBM2000] *IBM Corporation (Hrsg.):* IBM San Francisco Overview. IBM Corporation, 2000. http://www.ibm.com/software/ad/sanfrancisco, Abruf am 1.3.2002.

[Kauf2000] *Kaufmann, T.:* Entwurf eines Marktplatzes für heterogene Komponenten betrieblicher Anwendungssysteme. Berlin 2000.

[McKi2001] *McKie, S.:* Web Services – A Manager's Guide. Content Can, 2001.

[Merz2002] *Merz, M.:* E-Commerce und E-Business – Marktmodelle, Anwendungen und Technologien. dpunkt-Verlag, Heidelberg 2002.

[Orfa1996] *Orfali, R. et al:* The Essential Distributed Objects Survival Guide. John Wiley & Sons, New York 1996.

[Ortn1999a] *Ortner, E. et al:* Anwendungssystementwicklung mit Komponenten. In: IM Information Management & Consulting 14 (1999) 2, S. 35 – 45.

[Ortn1999b] *Ortner, E.:* Repository Systems. Teil 1: Mehrstufigkeit und Entwicklungsumgebung. In: Informatik Spektrum 22 (1999) 4, S. 235 – 251. Teil 2: Aufbau und Betrieb eines Entwicklungsrepositoriums. In: Informatik Spektrum 22 (1999) 5, S. 351 – 363.

[Ortn2000] *Ortner, E.:* Terminologiebasierte, komponentenorientierte Entwicklung von Anwendungssystemen. In: *Flatscher, R. et al (Hrsg.):* Tagungsband 2. Workshop Komponentenorientierte betriebliche Anwendungssysteme. Wien 2000, S. 1 – 17.

[Raut2001] *Rautenstrauch, C.; Turowski, K.:* Common Business Component Model (COBCOM): Generelles Modell komponentenbasierter Anwendungssysteme. In: *Buhl, H. U. et al (Hrsg.):* Information Age Economy. Physica Verlag, Heidelberg 2001.

[Sahm2000] *Sahm, S.:* Organisationsmodelle zur komponentenorientierten Anwendungsentwicklung. In: *Turowski, K. (Hrsg.):* Modellierung und Spezifikation von Fachkomponenten: Workshop im Rahmen der MobIS 2000 Modellierung betrieblicher Informationssysteme. Siegen 2000, S. 17 – 40.

[Souz1998] *d'Souza, D.; Wills, A.:* Objects, Components and Frameworks with UML – The Catalysis Approach. Addison-Wesley, Reading (Massachusetts) 1998.

[Szyp1998] *Szyperski, C.:* Component Software: Beyond Object-Oriented Programming. Addison-Wesley, Harlow 1998.

[Turo2001] *Turowski, K.:* Spezifikation und Standardisierung von Fachkomponenten. In: Wirtschaftsinformatik 43 (2001) 3.

[Turo2002] *Turowski, K. (Hrsg.):* Vereinheitlichte Spezifikation von Fachkomponenten. Gesellschaft für Informatik (GI), Arbeitskreis 5.10.3, Augsburg, 2002. http://www.fachkomponenten.de, Abruf am 20.4.2002.

[UDDI2000] *UDDI Organization (Hrsg.):* UDDI Technical White Paper. UDDI Standards Organization, September 2000. http://www.uddi.org, Abruf am 1.3.2002.

[Wede1981] *Wedekind, H.; Müller, T.:* Stücklistenorganisation bei einer großen Variantenzahl. In: Angewandte Informatik 23 (1981) 9, S. 377 – 383.

Komponentenorientierte Vorgehensmodelle im Vergleich

Peter Fettke, Iulian Intorsureanu, Peter Loos

Technische Universität Chemnitz, Fakultät für Wirtschaftswissenschaften, Information Systems & Management (Professur Wirtschaftsinformatik II), D-09107 Chemnitz, Germany, Tel.: +49/371/531-4375, Fax: -4376, E-Mail: peter.fettke@isym.tu-chemnitz.de, iintorsu@inforec.ase.ro, loos@isym.tu-chemnitz.de, WWW: http://www.isym.tu-chemnitz.de/

Zusammenfassung. Werden Softwaresysteme auf Basis eines komponentenorientierten Architekturparadigmas entwickelt, stellt sich die Frage, welches Vorgehensmodell zur Projektabwicklung herangezogen werden kann. In der Literatur werden unterschiedliche Vorgehensmodelle zur komponentenorientierten Softwareentwicklung vorgeschlagen. Aus diesen werden in der vorliegenden Untersuchung vier Modelle ausgewählt: Catalysis, Perspective, Rational Unified Process 2002 und V-Modell ’97. Die ausgewählten Vorgehensmodelle werden auf Basis eines allgemeinen Rahmens beschrieben und verglichen. Dabei werden die Aspekte Terminologie, Klassifizierung, Komponentenbegriff, Abdeckung des Lebenszyklus einer Komponente, Abdeckung der Tätigkeitsbereiche, Prozessarchitektur, Prozesssteuerung, Rollenabdeckung und Adaption untersucht. Komponentenorientierte Vorgehensmodelle sind sowohl Weiterentwicklungen bekannter konventioneller Vorgehensmodelle als auch ausschließlich auf die komponentenorientierte Entwicklung ausgerichtet. Obwohl die ausgewählten Vorgehensmodelle speziell auf eine komponentenorientierte Entwicklung ausgerichtet sind, zeigt sich, dass wesentliche Lebenszyklen einer Komponente nur rudimentär behandelt werden. Eine Ausnahme bildet hier Catalysis.

Schlüsselworte: Catalysis, Perspective, Rational Unified Process 2002, V-Modell ’97, Methodenvergleich, Komponente, Component Engineering, Prozessmodell

1 Ausgangssituation und Problemstellung

Die Idee, gesamte Softwaresysteme aus mehr oder weniger vorgefertigten Bausteinen zu entwickeln, ist bereits mehrere Jahrzehnte alt und wurde in der Vergangenheit immer wieder aufgegriffen.[Same1997; Grif1998; Szyp1999] Dabei zeigte sich, dass verschiedene Autoren unterschiedliche Auffassungen an den Begriff der Komponente legten. Neben den Aspekten der Granularität einer Komponente wurden weitere Eigenschaften wie Wiederverwendbarkeit, Abgeschlossenheit, Vermarktbarkeit, Kombinierbarkeit usw. in der Literatur diskutiert.[Turo2001, S. 15-19] In jüngster Zeit zeigte sich, dass gewisse Übereinkünfte in der Literatur erzielt werden konnten, was unter einer Komponente zu verstehen ist. Dies ermöglichte u. a. im Arbeitskreis 5.10.3 „Komponentenorientierte betriebliche Anwendungssysteme“ der Gesellschaft für Informatik eine einheitliche Definition des Begriffes Komponente zu formulieren:

„Eine Komponente besteht aus verschiedenartigen (Software-)Artefakten. Sie ist wiederverwendbar, abgeschlossen und vermarktbar, stellt Dienste über wohldefinierte Schnittstellen zur Verfügung, verbirgt ihre Realisierung und kann in Kombination mit anderen Komponenten

eingesetzt werden, die zur Zeit der Entwicklung nicht unbedingt vorhersehbar ist. [im Original z. T. kursiv gesetzt, die Autoren]“[Acke+2002, S. 1]

In Theorie und Praxis sind seit Beginn des Software Engineering unterschiedliche Vorgehensmodelle bekannt.[Brem1998] Ein Vorgehensmodell kann verstanden werden als „ein Muster zur Beschreibung eines Entwicklungsprozesses [im Original z. T. kursiv gesetzt, die Autoren]“[FBM+1998, S. 20] für ein Softwaresystem. Die zunehmende Verwendung von Komponenten-Strategien zur Entwicklung von Softwaresystemen wirft die Frage auf, wie Vorgehensmodelle zur Entwicklung derselben ausgestaltet sind bzw. sein sollten. Die Durchsicht der Literatur zeigt, dass verschiedene Vorgehensmodelle zur Entwicklung von Komponenten vorgeschlagen werden.

Ziel des vorliegenden Beitrages ist es, einen vergleichenden Überblick über ausgewählte Vorgehensmodelle zur komponentenorientierten Entwicklung von Softwaresystemen zu geben. Dabei sollen insbesondere diejenigen Aspekte berücksichtigt werden, die im Hinblick auf die komponentenorientierte Entwicklung eine besondere Rolle einnehmen. Es sollen die vorgestellten Vorgehensmodelle nicht im Detail behandelt werden. Dazu sei auf die Originalquellen verwiesen. Der Vergleich der Vorgehensmodelle ist systematisch, da er auf Basis eines einheitlichen Vergleichsrahmens durchgeführt wird. (Ein nicht-systematischer Vergleich, sondern allgemeiner Überblick über komponentenorientierte Vorgehensmodellen findet sich bei [Turo2001, S. 110-121]).

Die Arbeit ist wie folgt aufgebaut: Kapitel 2 erläutert kurz die Auswahlentscheidung der betrachteten Vorgehensmodelle und gibt dann einen kurzen Überblick über diese. Im Kapitel 3 wird der Vergleichsrahmen für die Untersuchung der Vorgehensmodelle beschrieben. Kapitel 4 stellt die Vorgehensmodelle vergleichend gegenüber. Im abschließenden Kapitel 5 wird ein Resümee der Untersuchung gezogen und ein Ausblick auf weitere Fragestellungen gegeben.

2 Überblick über die betrachteten Vorgehensmodelle

2.1 Auswahl der Vorgehensmodelle

Grundlage der Auswahl der untersuchten Vorgehensmodelle war eine Literaturrecherche. Um in das Untersuchungsfeld aufgenommen zu werden, wurden folgende Kriterien angelegt:

- Das Vorgehensmodell sollte explizit eine komponentenorientierte Vorgehensweise unterstützen. Dabei sollte der in Kapitel 1 eingeführte Komponentenbegriff als Grundlage dienen. Dieser musste mit dem jeweiligen Komponentenbegriff der Autoren der betrachteten Vorgehensmodelle zumindest in grundsätzlichen Merkmalen übereinstimmen. Diese Einschränkung führt dazu, dass innerhalb dieser Untersuchung keine Aussagen getroffen werden können, wie nicht-komponentenorientierte Vorgehensmodelle eine komponentenorientierte Entwicklung unterstützen. Durch dieses Kriterium werden bspw. Vorgehensmodelle wie Extreme Programming [Beck2000; Beck1999; BeFo2000], Crystal [o.V.2001] oder Personal Software Process [Hump1995] ausgeschlossen, deren Protagonisten zwar zum Teil feststellen, dass auf Basis dieser Modelle eine komponentenorientierte Vorgehensweise möglich ist, allerdings keine speziellen Konzepte und Methoden dafür einführen.
- Das Vorgehensmodell sollte von vornherein nicht ausschließlich ausgewählte Aktivitäten in der Entwicklung unterstützen, sondern einen breiten Rahmen von Entwick-

lungsphasen abdecken. Ferner sollten die beschriebenen Aktivitäten in einer hinreichenden Tiefe dargelegt werden, um deren unmittelbare Anwendung zu erlauben. Diese Einschränkung führt dazu, dass ausschließlich Modelle betrachtet werden, die einen hohen Reifegrad erreicht haben, um realistischerweise als Grundlage zur Entwicklung in der industriellen Praxis herangezogen werden zu können, ohne zunächst eine tiefgreifende Ausarbeitung und Detaillierung notwendig werden zu lassen. Aufgrund dieser Einschränkung sind folgende Arbeiten nicht mit in die Untersuchung einbezogen worden: [STR2000; Burg+2000; McIn1999] beschreiben erste grundlegende Ideen für komponentenorientierte Vorgehensmodelle, [Schr2001] behandelt primär die Einführung von komponentenorientierten Vorgehensmodellen.

- Die Ausführungen sollten den Charakter eines Vorgehensmodells haben. Aufgrund dieses Kriteriums werden die Arbeiten von [Same1997; Szyp1999; HeCo2001; Grif1998; Turo2001] nicht mit in die Untersuchung einbezogen. Diese Arbeiten geben zwar einen umfassenden Überblick über Aufgaben, Techniken und Konzepte bei der komponentenorientierten Softwareentwicklung. Letztlich liefern diese Autoren allerdings keine zusammenhängende Beschreibung eines Entwicklungsprozesses für komponentenorientierte Softwaresysteme.

Bei der Literaturrecherche gefundene Vorgehensmodelle, die den definierten Kriterien genügten, sind in Bild 1 überblicksartig zusammengestellt und werden in den folgenden Abschnitten 2.2 bis 2.5 zunächst knapp erläutert.

Bezeichnung	Autor(en)	Erscheinungsjahr	Quelle(n)
Catalysis	D'Souza; Wills	1998	[DSWi1998], www.catalysis.org
Perspective	Allen; Frost	1998	[AlFr1998]
Rational Unified Process 2000	Rational Software	2001	[Rati2001]
V-Modell '97	Öffentliche Hand	1997	[o.V.1997a]

Bild 1: Betrachtete Vorgehensmodelle

2.2 Catalysis

Catalysis wurde von Desmond D'Souza und Allan Wills entwickelt. Das Vorgehensmodell ist im Jahre 1998 in [DSWi1998] beschrieben, verschiedene Informationen sind ebenso unter www.catalysis.org zu finden. Das Modell stützt sich weitgehend auf die Unified Modeling Language (UML). Catalysis beschreibt nicht einen Entwicklungsprozess, der für alle Entwicklungsprojekte gültig ist, sondern besteht vielmehr aus einer Reihe von sogenannten Prozess-Mustern (Process Patterns). Ein Prozess-Muster kann jeweils in besondern Kontexten bzw. unter bestimmten Randbedingungen angewendet werden. Darüber hinaus zeichnet sich Catalysis durch folgende Eigenschaften aus:

- Das Modell unterstützt ausschließlich eine komponentenorientierte Entwicklung. Um dies zu erreichen wird ein sogenannter Produktfamilien-Ansatz gewählt. Die Autoren

verstehen unter einer Produktfamilie mehrere Softwaresysteme, die auf einer Menge gleicher Komponenten beruhen.

- Das Modell basiert auf einem stark iterativen und inkrementellen Vorgehen, das zu sehr kurzen Entwicklungszeiten führt.
- Auch wenn der Ansatz keine durchgängige formale Spezifikation von Softwaresystemen erfordert, wird dennoch Wert auf eine hohe Qualität der Spezifikation eines Systems gelegt.

2.3 Perspective

Das Vorgehensmodell Perspective wurde ursprünglich 1994-1995 von der Firma Select (jetzt: Aonix) entwickelt. Im Jahre 1998 haben die Autoren Allen und Frost in einer Publikation ein überarbeitetes Modell vorgelegt.[AlFr1998] Gleichzeitig bietet die Firma Aonix ein zusammenhängendes Softwarepaket zur Unterstützung des Vorgehensmodells an. Perspective ist vollständig auf die Entwicklungsprozesse einer komponentenorientierten Entwicklung ausgerichtet. Das Vorgehensmodell verwendet zur Modellierung von Softwaresystemen die UML. Einen besonderen Fokus bekommt die Arbeit durch die explizite Betrachtung und Integration der Geschäftsprozessmodellierung in das Vorgehensmodell.

Das Vorgehensmodell versteht sich als eine konsolidierte Zusammenfassung einer Menge von Erfahrungen und Techniken, die sich beim Einsatz in der Praxis bewährt haben. Ein charakteristisches Merkmal des Prozesses ist die Trennung des Vorgehens in einen Prozess der Entwicklung unternehmensspezifischer Softwaresysteme einerseits, sowie der Entwicklung von Komponenten andererseits. Koordiniert werden beide Prozesse durch ein Repositorium für Komponenten, in dem diese archiviert und recherchiert werden können. Hierbei wird zwischen Benutzer-, Geschäfts- sowie Daten-Diensten unterschieden. Sowohl der Solution als auch der Component Process werden iterativ und inkrementell durchlaufen. Die Autoren weisen darauf hin, dass das Vorgehensmodell die parallele Entwicklung verschiedener Systeme bzw. Systemteile fördert.

2.4 Rational Unified Process 2000

Der Rational Unified Process (RUP) wird von der Firma Rational Software als Hypertext angeboten.[Rati2001] Dieser Untersuchung lag die Version 2002.05.00 aus dem Jahre 2001 zu Grunde. Der RUP kann als eine konkrete und detaillierte Variante des Unified Software Development Process (USDP) verstanden werden, der von [JBR1998] entwickelt wurde. Der USDP wiederum ist auf das Vorgehensmodell Objectory zurückzuführen, dass in einer frühen Version in [JCJÖ1992] beschrieben wurde. Der RUP basiert auf einer objektorientierten Modellierung mit der UML und erlaubt ein inkrementelles und iteratives Vorgehen. Neben der Hypertext-Dokumentation werden von Rational Software eine Reihe von Werkzeugen für verschiedene Aufgaben im Entwicklungsprozess angeboten. Die Integration der Beschreibung ist allerdings nicht soweit fortgeschritten, dass der RUP nur mit den angebotenen kommerziellen Werkzeugen verwendet werden kann. Vielmehr kann das Vorgehensmodell auch mit anderen Werkzeugen genutzt werden.

Der RUP ist weniger als ein konkretes Vorgehensmodell zu verstehen, sondern vielmehr als ein Prozessrahmenwerk, dass für verschiedene Einsatzgebiete angepasst werden kann. Hierbei werden von Rational unterschiedliche Möglichkeiten, sogenannte Roadmaps, vorgestellt. Un-

ter anderem existiert eine Adaption für die komponentenorientierte Vorgehensweise, auf die sich die folgende Beschreibung bezieht. Das Vorgehensmodell hat zwei Dimensionen: Zum einen wird hinsichtlich der Zeit unterschieden, in welcher Entwicklungsphase sich ein Projekt befindet. Orthogonal zu der zeitlichen Untergliederung wird hinsichtlich verschiedener Arbeitsbereiche unterschieden, die in jedem Iterationsschritt auszuführen sind.

2.5 V-Modell ’97

„Das V-Modell [Vorgehensmodell, die Autoren] ist ein anerkannter Entwicklungsstandard für IT-Systeme, der einheitlich und verbindlich festlegt, was zu tun ist, wie die Aufgaben durchzuführen sind und womit dies zu geschehen hat. Es umfasst

- das Vorgehensmodell,
- die Methodenzuordnung und
- die funktionalen Werkzeuganforderungen.

Damit ist klar umrissen, in welchen Schritten und mit welchen Methoden die Entwicklungsarbeiten auszuführen sind.“[o.V.1997b, S. 2.] Ursprünglich entwickelt wurde das V-Modell im Auftrag des Bundesministeriums für Verteidigung in Zusammenarbeit mit dem Bundesamt für Wehrtechnik und Beschaffung in Koblenz von der Industrieanlagen-Betriebsgesellschaft mbH in Ottobrunn bei München. Der aktuelle Standard ist das V-Modell ’97 (kurz: V-Modell), in dem eine Reihe von Ergänzungen und Verbesserungen vorgenommen worden sind (bspw. Straffung der Prozessdokumentation sowie Einführung von modernen Entwicklungsszenarien). Das V-Modell wird inzwischen nicht nur von Behörden verwendet, sondern hat auch in der Industrie an Verbreitung gewonnen.

Das V-Modell kennt verschiedene Szenarien, für die der Ablauf der definierten Aktivitäten in den verschiedenen Submodellen exemplarisch dargestellt wird. Von den sechs beschriebenen Szenarien wie bspw. „Inkrementelle Entwicklung“, „Grand Design“, „Objektorientierte Entwicklung“ soll im Folgenden das Szenario „Einsatz von Fertigprodukten“ herausgegriffen werden. Dieses Szenario basiert auf der Idee, den „Entwicklungsaufwand durch Nutzung bereits verfügbarer Bausteine (SW und HW) zu reduzieren“[o.V.1997a, Teil 3: Handbuchsammlung - Szenarien]

3 Vergleichsrahmen

3.1 Überblick

Die Untersuchung der Vorgehensmodelle basiert auf einem einheitlichen Vergleichsrahmen. Der Vergleichsrahmen wurde auf Grundlage von [NoSc1999] entwickelt, die einen Vergleich objektorientierter Vorgehensmodelle durchgeführt haben. Der hier verwendete Vergleichsrahmen wurde im Hinblick auf komponentenorientierte Vorgehensmodelle angepasst. Der Vergleichsrahmen besteht aus folgenden Aspekten:

- Terminologie
- Klassifizierung
- Komponentenbegriff

- Abdeckung des Lebenszyklus einer Komponente
- Abdeckung der Tätigkeitsbereiche
- Prozessarchitektur
- Prozesssteuerung
- Rollenabdeckung
- Adaption.

Der Vergleichsrahmen wird in den folgenden Abschnitten 3.2 bis 3.10 vorgestellt.

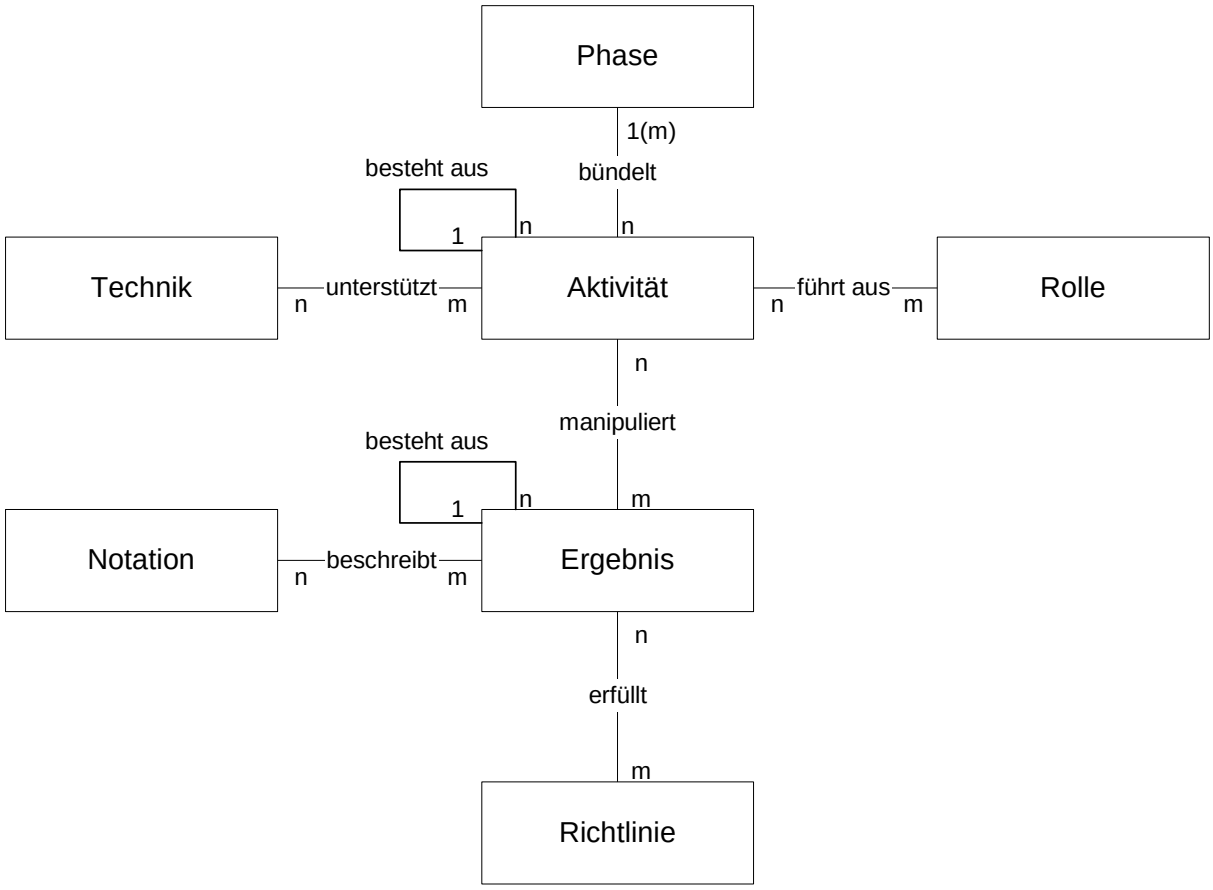


Bild 2: Grundlegende Terminologie (Quelle: [NoSc1999, S. 169])

3.2 Terminologie

Es wird folgende neutrale Terminologie für diese Untersuchung eingeführt:[NoSc1999, S. 168-170]

- Eine Phase bezeichnet eine Gruppe von Aktivitäten. Phasen sind grundlegende Einheiten zur Planung und Steuerung eines Projektes.
- Eine Aktivität ist eine Beschreibung der durchzuführenden Arbeitsschritte. Aktivitäten erhalten bestimmte Ergebnisse als Eingabe und erzeugen Ergebnisse als Ausgabe. Ferner ist es möglich, Vor- und Nachbedingungen von Aktivitäten zu bestimmen. Aktivitäten können ebenso in Unteraktivitäten aufgeteilt werden.

- Bei Ergebnissen handelt es sich um Dokumente, die bei der Durchführung von Aktivitäten erzeugt werden. Ergebnisse können sich wiederum aus Teilergebnissen zusammensetzen.
- Eine Rolle definiert die notwendigen Fähigkeiten, Kenntnisse und Erfahrungen von Personen, die am Entwicklungsprozess beteiligt sind.
- Eine Technik beschreibt, wie eine Aktivität durchzuführen ist, wohingegen eine Aktivität beschreibt, was auszuführen ist.
- Eine Richtlinie beschreibt einen Standard, der es erlaubt, Ergebnisse vergleichbarer und messbarer Qualität zu erzeugen sowie die Ergebnisse verschiedener Projekte austauschbar zu gestalten.
- Eine Notation definiert, mit welchen Zeichen die Ergebnisse in Dokumenten festgehalten werden.

In Bild 2 werden die eingeführten Begriffe im Zusammenhang graphisch dargestellt.

3.3 Klassifizierung

Komponentenorientierte Vorgehensmodelle können folgendermaßen klassifiziert werden:

- Art: Es wird hinsichtlich evolutionären und revolutionären Vorgehensmodellen unterschieden. Vorgehensmodelle werden in diesem Kontext als evolutionär bezeichnet, wenn sie durch Anpassung vorhandener Vorgehensmodelle für die komponentenorientierte Entwicklung angepasst und tauglich gemacht wurden. Dahingegen sind revolutionäre Vorgehensmodelle vollständig neu konzipiert und betrachten ausschließlich die komponentenorientierte Softwareentwicklung.
- Komponentenbildung: Es wird zwischen Top-Down- und Bottom-Up-basierten Vorgehensmodellen unterschieden.[Schr2001, S. 82] Beim Top-Down-Ansatz wird die Analyse und der Entwurf eines Softwaresystems unabhängig von bereits vorhandenen Komponenten durchgeführt. Dahingehen werden bei Bottom-Up-basierten Ansätzen schon zu Beginn des Entwicklungsprozesses vorhandene Komponenten berücksichtigt. In einem Entwicklungsprozess können beide Prinzipien der Komponentenbildung angewendet werden.
- Systemgranularität: Es wird unterschieden zwischen der Entwicklung einer einzelnen Komponente und eines vollständigen Softwaresystems, das aus mehreren Komponenten besteht. Ferner ist es denkbar, dass das Vorgehensmodell sowohl die Entwicklung einer einzelnen Komponente als auch eines vollständigen Softwaresystems unterstützen kann.

3.4 Komponentenbegriff

Der in Kapitel 1 eingeführte Komponentenbegriff kann auf verschiedene Merkmale zurückgeführt werden. Es wird untersucht, inwieweit die Vorgehensmodelle die Merkmale ebenso beschreiben:

- Software-Artefakte: Eine Komponente setzt sich aus mehreren Software-Artefakten wie Spezifikation, Implementierung, Benutzerhandbuch etc. zusammen.

- Wiederverwendbarkeit: Eine Komponente kann in verschiedenen Projekten verwendet werden.
- Abgeschlossenheit: Die Software-Artefakte, die zu einer Komponente gehören, können eindeutig abgegrenzt werden.
- Vermarktbarkeit: Eine Komponente kann prinzipiell auf Komponentenmärkten (Commercial off-the-shelf (COTS) Komponente) gehandelt werden. Alternativ ist ebenso eine Vermarktung in einem unternehmensinternen Komponenten-Repositorium denkbar.
- Dienstangebote: Eine Komponente bietet unterschiedliche Dienste wie bspw. Benutzer-, Daten- oder Geschäftsdienste an.
- Wohldefinierte Schnittstelle: Die Schnittstelle der Komponente zur Nutzung der Dienste ist explizit definiert. Dies schließt ebenso die explizite Definition der Dienste ein, die von der Komponente nachgefragt werden.
- Verbirgt ihre Realisierung: Die programmtechnische Implementierung einer Komponente ist nicht sichtbar.
- Kombinierbarkeit: Eine Komponente kann mit anderen Komponenten benutzt werden, ohne dass dies zur Entwicklungs- oder Übersetzungszeit der Komponenten geplant wurde. Vielmehr können Komponenten ohne Änderungen ihrer Implementierungen, quasi zur Laufzeit, zusammengesetzt werden.

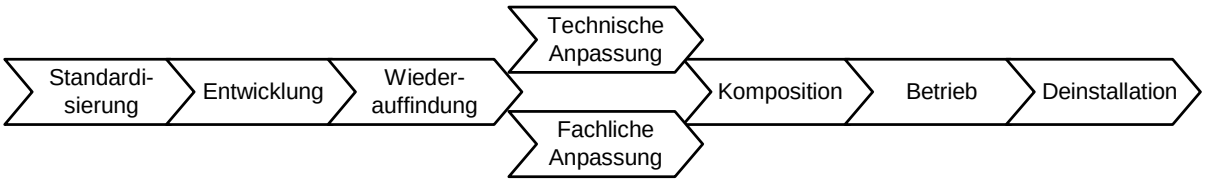


Bild 3: Lebenszyklus einer Komponente (in Anlehnung an [Turo1999, S. 13])

3.5 Abdeckung des Lebenszyklus einer Komponente

Einzelne Komponenten bzw. komponentenorientierte Softwaresysteme können hinsichtlich ihres Lebenszyklus strukturiert werden. Grundlage dieser Arbeit ist das Lebenszyklus-Konzept in [Turo2001, S. 41-48]. Dieses unterscheidet folgende Phasen:(vgl. Bild 3)

- Standardisierung: Diese Phase umfasst die Vereinheitlichung und Spezifikation von Komponenten.
- Entwicklung: Diese Phase umfasst die vollständige Realisierung einer Komponente mit Ausnahme ihrer Standardisierung.
- Technische Anpassung: Diese Phase umfasst die Behebung implementierungsbedingter, technischer Inkompatibilitäten.
- Fachliche Anpassung: Diese Phase umfasst die fachliche Parametrisierung einer Komponente.
- Komposition: Diese Phase umfasst die fachliche und technische Integration einer Komponente in ein Softwaresystem.

- Betrieb: Diese Phase umfasst alle Aufgaben zur Anpassung und zum Austausch einer Komponente nach ihrer erstmaligen Installation.
- Deinstallation: Diese Phase umfasst alle Aufgaben, die mit der Entfernung einer Komponente aus einem Softwaresystem zusammenhängen.

Dieses Lebenszyklus-Konzept wird ergänzt um die Phase der Wiederauffindung, der im Rahmen der komponentenorientierten Softwareentwicklung eine besondere Bedeutung beigemessen wird.[FeLo2000; FeLo2001] Diese Phase umfasst sowohl die Suche nach Komponenten, die Selektion grundsätzlich geeigneter Komponenten, die Auswahl einer Komponente und die Beschaffung der ausgewählten Komponente. Diese Phase wird nach der Phase Entwicklung in das Lebenszyklus-Konzept eingeführt.

3.6 Abdeckung der Tätigkeitsbereiche

Die Aktivitäten, die im Rahmen von Vorgehensmodellen definiert werden, können auf einer allgemeinen Ebene vier Tätigkeitsbereichen zugeordnet werden:[FBM+1998, S. 16-26]

- Projektmanagement: Dieser Tätigkeitsbereich umfasst die Planung, Steuerung und Kontrolle eines Softwareentwicklungsprojektes.
- Konfigurationsmanagement: Dieser Tätigkeitsbereich umfasst die „Identifikation der Konfigurationen eines Systems zu diskreten Zeitpunkten zum Zwecke der systematischen Steuerung von Konfigurationsänderungen und der Aufrechterhaltung der Vollständigkeit und Verfolgbarkeit der Konfigurationen während des gesamten SW-Lebenszyklus.“[FBM+1998, S. 24]
- Qualitätsmanagement: Dieser Tätigkeitsbereich umfasst die Planung, Steuerung und Kontrolle der Qualität eines Softwaresystems (Produkt) sowie des Prozesses seiner Erstellung.
- Systementwicklung: Dieser Tätigkeitsbereich umfasst die unmittelbare Erstellung des Softwaresystems. Dagegen nehmen die anderen drei Tätigkeitsbereiche nur eine mittelbare Rolle bei der Systementwicklung ein.

3.7 Prozessarchitektur

Einen groben Überblick über die Anordnung und den Ablauf der Phasen und der zugehörigen Aktivitäten wird Prozessarchitektur genannt.[NoSc1999, S. 171] Traditionelle Modelle wie das Wasserfall-Modell beruhen bspw. auf einem sequentiellen Modell, in dem alle Phasen linear durchlaufen werden. Neuere Modelle besitzen i. d. R. eine ausgefeiltere Ablaufreihenfolge.

3.8 Prozesssteuerung

Die Prozesssteuerung beschreibt, welches Prinzip dem Vorgehensmodell zur Definition der Ablaufreihenfolgen zugrunde liegt.[NoSc1999, S. 175] Hierbei werden drei verschiedene Prinzipien unterschieden:

- Aktivitätsorientierte Vorgehensmodelle: Diese Modelle beschreiben, welche Schritte in welcher Reihenfolge durchzuführen sind, um bestimmte Aktivitäten bzw. Phasen erfolgreich zu durchlaufen.

- Ergebnisorientierte Vorgehensmodelle: Werden die zu erstellenden Ergebnisse, deren Status und ihre Transformation in den Vordergrund gestellt, handelt es sich um ergebnisorientierte Vorgehensmodelle.
- Entscheidungsorientierte Vorgehensmodelle: Bei dieser Prozesssteuerung werden bestimmte Bedingungen definiert, die beschreiben, wann welche Aktivitäten durchzuführen sind. Damit werden bestimmte Entwicklungsmuster situationsbedingt festgehalten und beschrieben.

Es sei darauf hingewiesen, dass die beschriebenen Prinzipien sich nicht gegenseitig ausschließen, sondern komplementär verwendet werden können. Konkrete Vorgehensmodelle verwenden i. d. R. mehrere Prinzipien. Dabei steht meist ein Prinzip im Vordergrund bei der Beschreibung des Vorgehensmodells.

3.9 Rollenabdeckung

Unter diesem Aspekt werden die im Vorgehensmodell definierten Rollen näher beschrieben. Dabei wird ein besonderer Fokus auf diejenigen Rollen gelegt, die im Kontext der komponentenorientierten Entwicklung eine besondere Bedeutung haben.

3.10 Adaption

Vorgehensmodelle streben nach einem bestimmten Grad an Allgemeingültigkeit, um unabhängig von bestimmten Anwendungsbedingungen einsetzbar zu sein. Auch wenn Vorgehensmodelle eine Standardisierung der Prozesse der Systementwicklung anstreben, ist es meist unumgänglich, bei der Einführung eines Vorgehensmodells bestimmte Anpassungen an diesem vorzunehmen.[NoSc1999, S. 177] Die Möglichkeiten, die ein Vorgehensmodell dazu erlaubt, werden unter dem Aspekt Adaption näher beschrieben.

4 Gegenüberstellung und Vergleich

4.1 Terminologie

In Bild 4 werden die in den ausgewählten Vorgehensmodellen verwendeten Begriffe der hier eingeführten Terminologie gegenübergestellt, um so ein besseres Verständnis und eine höhere Transparenz über die verwendete Sprache zu erhalten.

	Catalysis	Perspective	RUP	V-Modell
Phase	-	Stage	Phase	Phase
Aktivität	Activity	Task	Workflow, Activity, Step	Aktivität, Teilaktivität
Ergebnis	Deliverable	Deliverables, Inputs, Outputs	Artifact	Produkt, Teilprodukt
Rolle	-	Team Role	Worker	Rolle
Technik	Technique	Technique	-	Elementar-methode
Richtlinie	-	Practical Guidelines	Work Guidelines	Handbuchsamm-lung
Notation	Notation	Notation	Language	-

Bild 4: Terminologische Einordnung

4.2 Klassifizierung

Eine Klassifizierung der ausgewählten Vorgehensmodelle erfolgt in Bild 5.

	Catalysis	Perspective	RUP	V-Modell
Art	revolutionär	revolutionär	evolutionär	evolutionär
Komponenten-bildung	Top-Down und Bottom-Up	Bottom-Up	Top-Down	Top-Down
System-granularität	Anwendung	Komponente und Anwendung	Anwendung	Anwendung

Bild 5: Klassifizierung der Vorgehensmodelle

4.3 Komponentenbegriff

In Bild 6 werden die von den untersuchten Vorgehensmodellen eingeführten Komponenten-begriffe gegenübergestellt. Hierbei wurde differenziert, ob Merkmale explizit angeführt oder implizit unterstellt worden sind. Falls keine Aussagen möglich waren, wurde dies ebenso vermerkt. Ergänzend sei darauf hingewiesen, dass bei der Auswertung der Literatur es ebenso vorgesehen war, dass bestimmte Merkmale von den Autoren der Vorgehensmodelle explizit ausgeschlossen werden. Von dieser prinzipiellen Möglichkeit machte allerdings keiner der Autoren Gebrauch.

	Catalysis	Perspective	RUP	V-Modell
Bezeichnung	Component	Component	Component	Fertigprodukt
Software-Artefakte	(+)	(+)	(+)	(+)
Wiederverwendbarkeit	(+)	+	+	+
Abgeschlossenheit	+	+	(+)	(+)
Vermarktbarkeit	?	?	?	(+)
Dienstangebote	+	+	+	?
Wohldefinierte Schnittstelle	+	+	+	?
Verbirgt ihre Realisierung	(+)	+	?	?
Kombinierbarkeit	(+)	(+)	?	?

Legende:

+	Merkmal explizit angeführt
(+)	Merkmal implizit unterstellt
?	keine Aussage möglich

Bild 6: Gegenüberstellung des Komponentenbegriffs

Das V-Modell unterschiedet hinsichtlich der Herkunft der Fertigprodukte in kommerzielle Fertigprodukte, die auf dem freien Markt angeboten werden, und Fertigprodukten, die innerhalb der eigenen Organisation vorliegen. Das V-Modell weist darauf hin, dass Fertigprodukte auf allen Ebenen der Erzeugnisstruktur des V-Modells, also nicht nur auf Quellcode-Ebene, sondern bspw. auch auf der Architektur- oder Anforderungs-Ebene verwendet werden können. Dieser weite Komponentenbegriff wird insofern von den Autoren wieder relativiert, indem bei der Entwicklungsprozessbeschreibung stets davon ausgegangen wird, dass ausschließlich Fertigprodukte für die Realisierung eingekauft werden.[o.V.1997a, Teil 3: Handbuchsammlung - Szenarien]

Ferner soll auf zwei weitere sprachliche Schwierigkeiten hingewiesen werden. Erstens verwendet der RUP das Wort „component“ sowohl in einer allgemeinen Bedeutung für bspw. Datenbank-Tabellen, Quell-Code-Dateien u. ä. Darüber hinaus wird das Wort ebenso in dem hier unterstellten Komponentenbegriff verwendet (vgl. Definition „component“ im Glossar des Vorgehensmodells). Zweitens finden sich auch bei Catalysis sprachliche Unschärfen. Catalysis führt zunächst einen Komponentenbegriff im Allgemeinen und im Besonderen ein.[DSWi1998, S. 386f.] Davon abweichend wird indes ebenso ein unreflektierter Kompo-

nentenbegriff verwendet, der bspw. auch organisatorische Einheiten umfasst.[DSWi1998, S. 414]

4.4 Abdeckung des Lebenszyklus einer Komponente

4.4.1 Überblick

Bild 7 gibt einen Überblick darüber, welche Phasen im Lebenszyklus einer Komponente von den betrachteten Vorgehensmodellen (gut) unterstützt werden. Dabei wird jeweils der Name der Phase bzw. Aktivität angegeben, wie er von dem entsprechenden Vorgehensmodell verwendet wird. In den folgenden Unterabschnitten werden verschiedene Aspekte aufgegriffen, die explizit bei einer komponentenorientierten Bedeutung von Interesse sind.

	Catalysis	**Perspective**	**RUP**	**V-Modell**
Standardisierung	How to Specify a Component	-	-	-
Entwicklung	How to Implement a Component	Component Process	Process Workflows	Systemerstellung
Wiederauffindung	-	(Search for possible areas of reuse)	(Project Management)	Realisierbarkeitsuntersuchung, Marktsichtung
Technische Anpassung	Reuse and Pluggable Design Frameworks in Code	-	-	(Systemkonfektion)
Fachliche Anpassung	-	-	-	(Systemkonfektion)
Komposition	Components and Connectors	(Roll Out)	Deployment	Integration
Betrieb	-	-	-	-
Deinstallation	-	-	-	-

Legende:

(...) Phase im Lebenszyklus einer Komponente wird zwar genannt, inhaltliche Ausgestaltung bleibt (teilweise) unklar

\- Phase im Lebenszyklus einer Komponente wird nicht abgedeckt

Bild 7: Abdeckung des Lebenszyklus einer Komponente

4.4.2 Catalysis

Catalysis beschreibt Aktivitäten zur Spezifikation von Komponenten, die von diesem Vorgehensmodell als eine Voraussetzung zur erfolgreichen Standardisierung von Komponenten angesehen werden. Hierbei wird insbesondere auf eine formale Spezifikation Wert gelegt. Komponenten sollen ohne weitere Änderungen, sondern nur durch Anpassung mit Parametri-

sierung an eine Systemumgebung angepasst werden können. Um die Komposition zu ermöglichen, wird eine sogenannte Kit-Architektur vorgeschlagen, die standardisierte Techniken bereithält. Ebenso werden Techniken zur Verknüpfung heterogener Komponenten dargestellt. Um die verschiedenen Techniken nutzen zu können, werden entsprechende Muster angegeben.[DSWi1998]

4.4.3 Perspective

Von Perspective wird ein expliziter Entwicklungsprozess für die Entwicklung einer Komponente vorgestellt. Aktivitäten, welche die anderen Phasen des Lebenszyklus einer Komponente betreffen, werden zwar genannt. Die inhaltliche Ausgestaltung dieser Phasen bleibt allerdings unklar.

4.4.4 RUP

RUP erläutert explizit für jede Phase und den jeweiligen Workflow die Auswirkungen bei einer komponentenorientierten Vorgehensweise. Dabei handelt es sich allerdings meist nicht um eine Darstellung konkreter Aktivitäten. Vielmehr werden entsprechende zusätzliche zu bedenkende Rahmenparameter beschrieben. Beispielsweise wird darauf hingewiesen, dass bei der Beschaffung von Komponenten bei Fremdanbietern zwar neue Risikofaktoren entstehen, gleichzeitig allerdings Entwicklungszeiten verkürzt werden können.

4.4.5 V-Modell

Bei dem V-Modell handelt es sich um ein Top-Down-basiertes Modell. Nachdem die Anforderungen an ein System beschrieben sind, wird im Rahmen einer Realisierbarkeitsuntersuchung eine Marktsichtung vorgenommen.[o.V.1997a, Teil 3: Handbuchsammlung – Szenarien, S. SA-13] Grundlage der Auswahl einer Komponente bildet ein auf Basis der Anforderungen erstellter Kriterienkatalog. Ergebnis der Realisierbarkeitsuntersuchung kann sein, dass die Anforderungen von einer am Markt erhältlichen Komponente gedeckt werden. Ist dies nicht der Fall, so wird ein sogenanntes Forderungscontrolling eingeleitet, indem überprüft wird, ob die definierten Anforderungen ohne Projektgefährdung in der Art angepasst werden können, dass evtl. zum Einsatz kommende Komponenten doch noch verwendet werden können. Der Einsatz von Komponenten erfordert nach ihrer Beschaffung sogenannte Konfektionierungsarbeiten, um die verwendeten Komponenten so anzupassen, dass sie den definierten Anforderungen genügen. Darüber hinaus werden ebenso notwendige Integrationsaktivitäten definiert, um die vorhandenen Komponenten lauffähig zu gestalten. Die inhaltliche Ausgestaltung der Konfektionierungs- sowie Integrations-Aktivitäten werden inhaltlich nicht weiter erläutert. Die Beschaffung der Komponente obliegt dem Projektmanagement. Dieses stößt evtl. das der Softwareentwicklung unterliegende Forderungscontrolling an. Eine explizite Durchführung von Prüfungsaktivitäten bei der Beschaffung von Komponenten wird im Rahmen des Qualitätsmanagements nicht als erforderlich erachtet. Vielmehr obliegt es dem Projektmanagement, die Qualität der eingekauften Komponenten zu gewährleisten. Anders dagegen bei den durchzuführenden Konfektionierungsaktivitäten. Hier werden diese durch Maßnahmen des Qualitätsmanagements überwacht. Das V-Modell schreibt vor, dass fremd- und eigenentwickelte Komponenten demselben Konfigurationsmanagement zu unterliegen haben. Dabei wird insbesondere darauf hingewiesen, dass die Weiterentwicklung der Komponenten nicht von neuen Anforderungen an das System ausgehen, sondern von dem Anbieter der

Komponente. Die Weiterentwicklung der Komponente ist demnach hauptsächlich vom Markt getrieben und nicht von den Anwenderanforderungen.

4.5 Abdeckung der Tätigkeitsbereiche

Bild 8 beschreibt, welche Tätigkeitsbereiche von den untersuchten Vorgehensmodellen aufgegriffen werden.

	Catalysis	**Perspective**	**RUP**	**V-Modell**
Projekt-management	-	-	Project Management	Projekt-management
Qualitäts-management	-	-	-	Qualitätssicherung
Konfigurations-anagement	-	-	Configuration & Chanage Mgmt	Konfigurations-management
Entwicklung	Catalysis Process	Perspective Process	Core Process Workflows	Systemerstellung

Legende:

- Tätigkeitsbereich wird nicht abgedeckt

Bild 8: Abdeckung der Tätigkeitsbereiche

4.6 Prozessarchitektur

4.6.1 Catalysis

Das Vorgehensmodell Catalysis nimmt eine Sonderstellung unter den Vorgehensmodellen ein, da das Modell keine Vorschriften bezüglich zu durchlaufender Phasen und Aktivitäten kennt. Statt dessen definiert das Vorgehensmodell eine Reihe von sogenannten Prozess-Mustern (Process Patterns), die beschreiben, unter welchen Voraussetzungen und Randbedingungen bestimmte Aktivitäten durchzuführen sind. Die Prozess-Muster sind in folgende Kategorien gegliedert:

- Main Process Patterns (4 Patterns)
- (Business) Modeling Patterns (13 Patterns)
- Patterns for Specifying Components (13 Patterns)
- Patterns for Designing to a Meet a Specification (13 Patterns)
- Detailed Design Patterns (5 Patterns).

Ein Beispiel für ein Main Process Pattern ist das Pattern Short-Cycle Development. Dieses Pattern beschreibt, dass kurze Entwicklungszyklen gewählt werden sollen, um Projekte unter unsicheren Bedingungen durchführen zu können. Andere Muster wie bspw. Object Development from Scratch übernehmen primär die Funktion andere Muster zu bündeln. Ferner existie-

ren ebenso Mustern, die für den Entwurf von Softwaressystemen geeignet sind. Beispielsweise sind die Detailed Design Patterns mit den Mustern in [GHJV1995] vergleichbar.

Nichtsdestotrotz weisen die Autoren darauf hin, dass der Ablauf sämtlicher Aktivitäten bei Catalysis einem Grundgerüst folgt. Dieses besitzt eine Form aus Modellierung, Entwurf, Implementierung und Test.[DSWi1998] Die Aktivitäten sind allerdings nicht umfassend definiert und erläutert, sondern haben nur eine allgemeine Bedeutung und können sich auf unterschiedliche Bezugsobjekte beziehen. Dabei werden die Bezugsobjekte allerdings nicht festgeschrieben. Beispielsweise nennen die Autoren die Bezugsobjekte: Business (Domain) Model, System Context, System Specification, Component Models, System Architecture, System Detailed Design.[DSWi1998, S. 528-530] Die exemplarisch genannten Bezugsobjekte können allerdings variieren und werden im Projektkontext jeweils neu festgelegt.

Aufgrund der großen Freiheitsgrade im Entwicklungsprozess charakterisieren die Autoren ihn als nicht-linear, iterativ und parallel. Nicht-linear bedeutet, dass alternative Pfade im Entwicklungsprozess möglich sind. Iterativ bedeutet, dass verschiedene Aktivitäten wiederholt durchgeführt werden. Darüber hinaus ist es möglich, dass bestimmte Aktivitäten parallel durchgeführt werden können. [DSWi1998, S. 512-514]

Aufgrund der genannten Eigenschaften ist es nicht möglich, eine allgemeine Prozessarchitektur zu beschreiben. Eine exemplarische Darstellung eines Entwicklungsprozesses wird in [DSWi1998, S. 522-526] gegeben. Diese soll hier aus Gründen des beschränken Umfanges der Untersuchung nicht rekapituliert werden.

4.6.2 Perspective

Auf einer abstrakten Ebene besteht das Vorgehensmodell Perspective aus zwei Prozessen: Ein sogenannter Solution Process erstellt ausgehend von spezifischen Anforderungen ein unternehmensindividuelles Softwaresystem. Dagegen werden im Component Process auf Basis allgemeingültiger Anforderungen spezifischer Domänen einzelne Komponenten entwickelt. Beide Prozesse werden in unterschiedliche Phasen aufgeteilt, die jeweils nicht strikt sequentiell, sondern iterativ und inkrementell durchlaufen werden können (vgl. Bild 9).

Der Solution Process besteht aus sieben Phasen:[AlFr1998, S. 271-301] Im Rahmen der Phase Feasibility wird eine Durchführbarkeitsstudie erstellt. Die Anforderungen an ein einzelnes Systeminkrement werden im Rahmen der Phase Analysis erhoben, die dabei durch gezieltes Entwickeln von benutzungsfähigen Prototypen unterstützt wird (Phase Prototype). Innerhalb der Phase Plan Increment wird ein Plan erstellt, der beschreibt, welche Systeminkremente, welche Funktionalitäten erfüllen sollen und in welcher Reihenfolge diese ausgeliefert werden. Der Entwurf und die Implementierung eines Inkrements erfolgen in der Phase Design and Build. Diese Phase wird überlagert von der Phase User Acceptance, um die Erfüllung der Benutzer-Anforderungen sicherzustellen. Abgeschlossen wird die Entwicklung mit der Einführung des Systems in der Phase Roll Out.

Der Component Process besteht aus acht Phasen:[AlFr1998, S. 303-327] Ein allgemeiner Überblick über die Anforderungen an eine Komponente werden innerhalb der Phase Architectural Scoping erstellt. Innerhalb der Phase Assessment wird überprüft, welche Nachfrage nach bestimmten Diensten von Projekten im Solution Process bestehen. Die Phase Plan Services erstellt einen Plan aus dem hervorgeht, welche Dienste in welchen Komponenteninkrementen bereitgestellt werden. Die Phasen Design and Build, Acceptance sowie Roll Out sind analog

zu den entsprechenden Phasen im Solution Process, beziehen sich allerdings hier auf die Entwicklung einer Komponente. Zwei zusätzliche Phasen werden durchlaufen, wenn im Rahmen des Assessment festgestellt wird, eine neue Komponente zu entwickeln und nicht eine vorhandene Komponente zu erweitern. Dann werden ebenso die Phasen Feasibility und Analysis abgearbeitet.

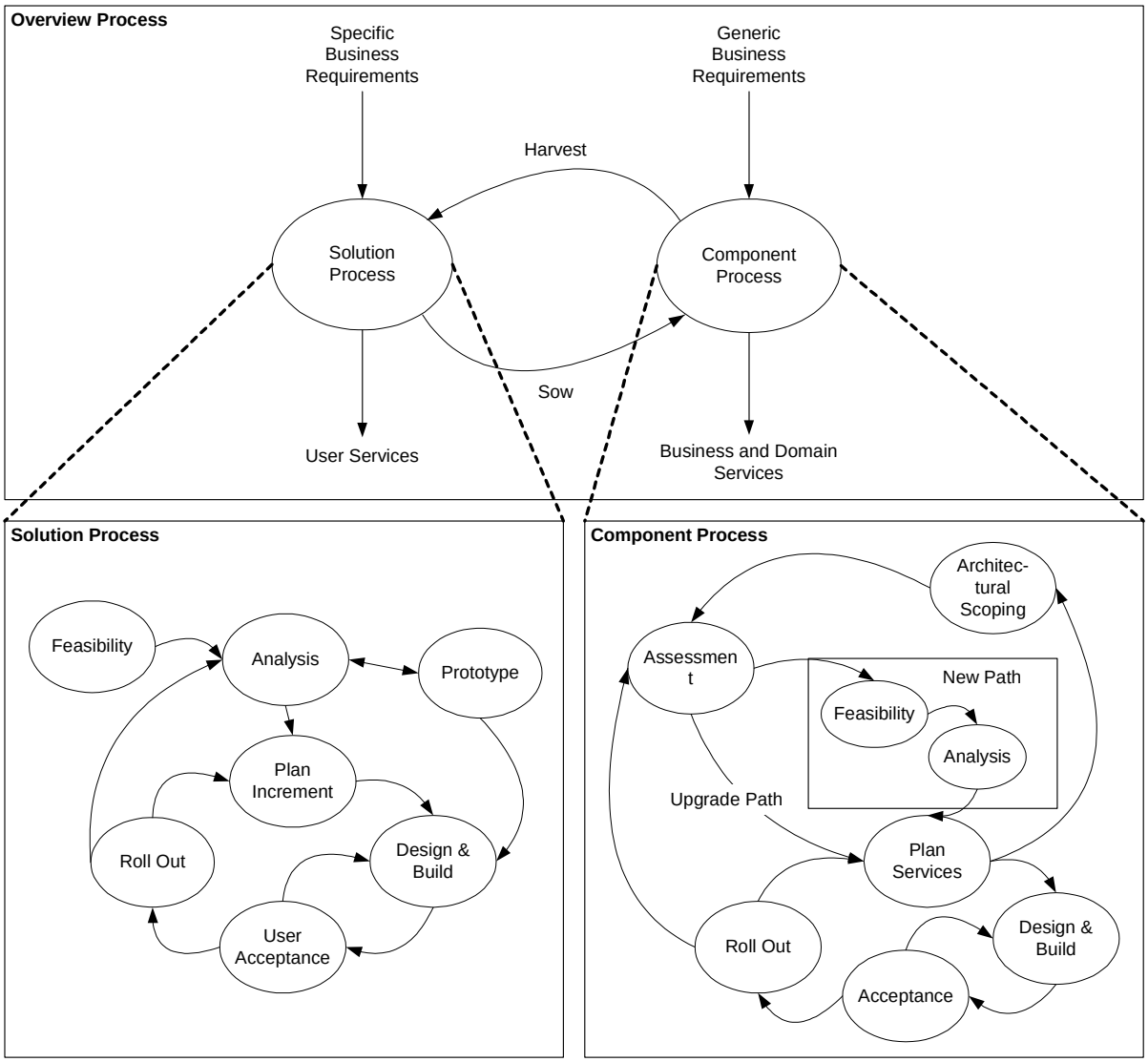


Bild 9: Prozessarchitektur von Perspective (in Anlehnung an: [AlFr1998, S. 264, 275, 309])

4.6.3 RUP

Ein Charakteristikum des RUP ist die konzeptionelle Trennung des Entwicklungsprozesses in Phasen und Arbeitsabläufe (Workflows). Die Trennung in Phasen spiegelt dabei eine zeitliche Unterscheidung wider. Die Trennung in Arbeitsabläufe orientiert sich an sachlichen Kriterien. Es werden vier Phasen unterschieden: In der Phase Inception wird eine Kernidee für ein Softwaresystem entwickelt und eine Grundvorstellung über die Leistungsfähigkeiten des Produktes hervorgebracht. Im Rahmen der Phase Elaboration wird die Architektur des Systems defi-

niert. Die Implementierung der Architektur geschieht im Rahmen der Phase Construction. Die Phase Transition umfasst die Auslieferung des Softwaresystems bei einem Anwender.

Innerhalb jeder Phase können mehrere Iterationen eingeführt werden, um eine inkrementelle Systementwicklung zu ermöglichen. Innerhalb jeder Iteration werden prinzipiell sämtliche Arbeitsabläufe (Business Modeling, Requirements, Analysis & Design, Implementation, Test, Deployment, Configuration Management, Project Management und Environment) durchgeführt. Dabei verschiebt sich allerdings der Schwerpunkt der Arbeitsabläufe: In frühen Iterationen liegt der Schwerpunkt eher bei konzeptionellen Tätigkeiten, während sich dieser zu späteren Projektphasen eher bei implementierungstechnischen Tätigkeiten befindet. Vgl. hierzu die entsprechenden Kurven in Bild 10, die als exemplarische Belastungsprofile in einem typischen Projekt interpretiert werden können. Jeder Arbeitsablauf wird durch ein UML Activity Diagramm weiter verfeinert. Diese Darstellung ist allerdings nicht explizit abhängig von der entsprechenden Phase, sondern für alle Phasen identisch. Eine weitere Differenzierung findet sich erst wieder in den verfeinerten, rein textuellen Darstellungen.

Prinzipiell kann der RUP als eine Variante des Spiralmodells nach Boehm [Somm1996, S. 13-16] verstanden werden, die inhaltlich detaillierter ausformuliert wurde und vornehmlich optisch anders dargestellt ist.

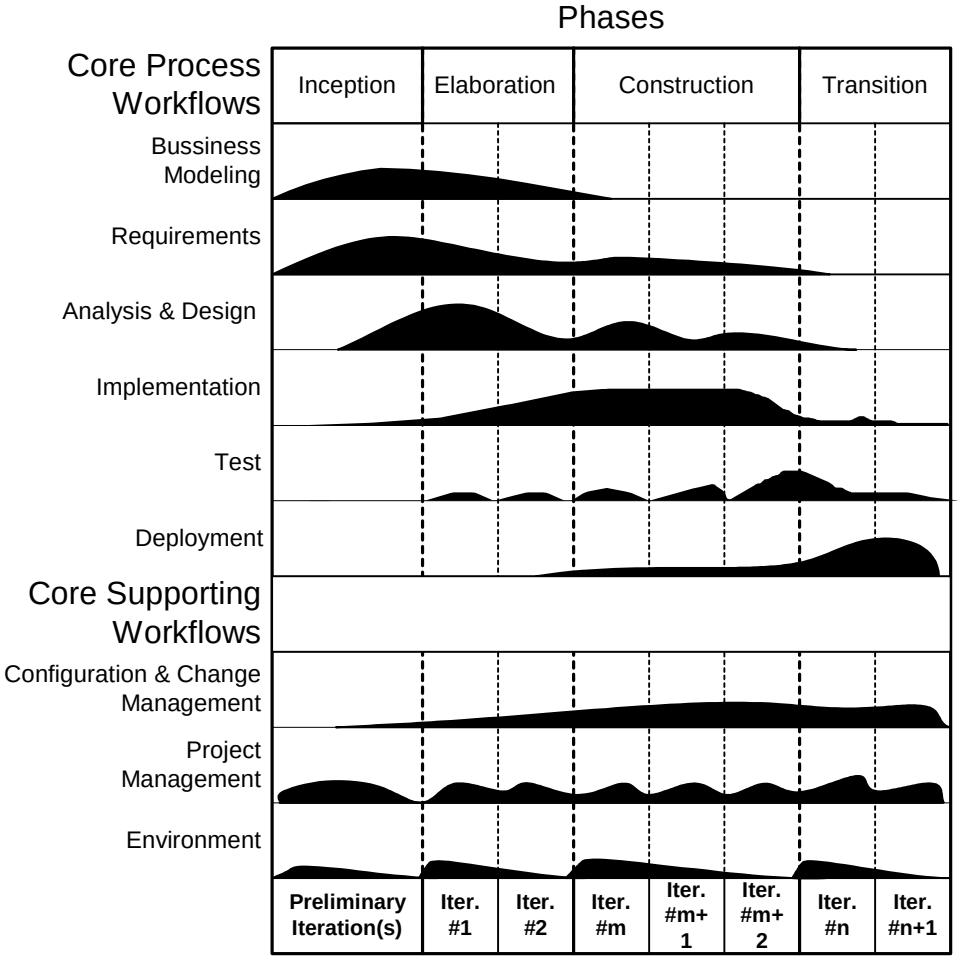


Bild 10: Prozessarchitektur von RUP (Quelle: [Rati2001])

4.6.4 V-Modell

Die gesamten Aktivitäten des V-Modell sind in vier Submodelle untergliedert:

- Das Projektmanagement (PM) plant, steuert, kontrolliert und informiert die drei Submodelle Systemerstellung (SE), Qualitätssicherung (QS) und Konfigurationsmanagement (KM).
- Das KM verwaltet die Produkte, die im Rahmen der übrigen Aktivitäten erstellt werden. Ein Produkt wird als das Ergebnis einer Aktivität bzw. als Gegenstand einer Aktivität des V-Modells verstanden. Dieses Submodell stellt sicher, dass alle Produkte eindeutig identifizierbar, dass verschiedene Versionen eines Produktes unterscheidbar und dass Änderungen an Produkten kontrolliert durchführbar sind.
- Die QS erarbeitet einerseits entsprechende Qualitätskriterien, Prüfpläne, Prüfkriterien und Standards. Andererseits werden die in den übrigen Modellen entwickelten Produkte den entsprechenden Prüfungen unterzogen.
- Die SE umfasst alle Aktivitäten, die unmittelbar dem Prozess der Softwareentwicklung zuzurechnen sind: System-Anforderungsanalyse, System-Entwurf, Grobentwurf, Feinentwurf, Implementierung, Integration, System-Integration und Überleitung in die Nutzung. Diese Aktivitäten können als Phasen verstanden werden und werden graphisch in Form des Buchstabens V angeordnet. Dabei werden die Aktivitäten auf der linken Seite der Anordnung der Entwicklung von Systemteilen, die auf der rechten Seite der Integration der Systemteile zugeordnet.

Trotz der Definition neuer Konzepte und verschiedener Entwicklungsszenarien ist festzuhalten, dass das V-Modell im Kern als ein um die Qualitätssicherung erweitertes Wasserfall-Modell mit Rückkopplung verstanden werden kann.[Balz1998, S. 101.] Das Zusammenspiel der verschiedenen Modelle wird in Bild 11 dargestellt.

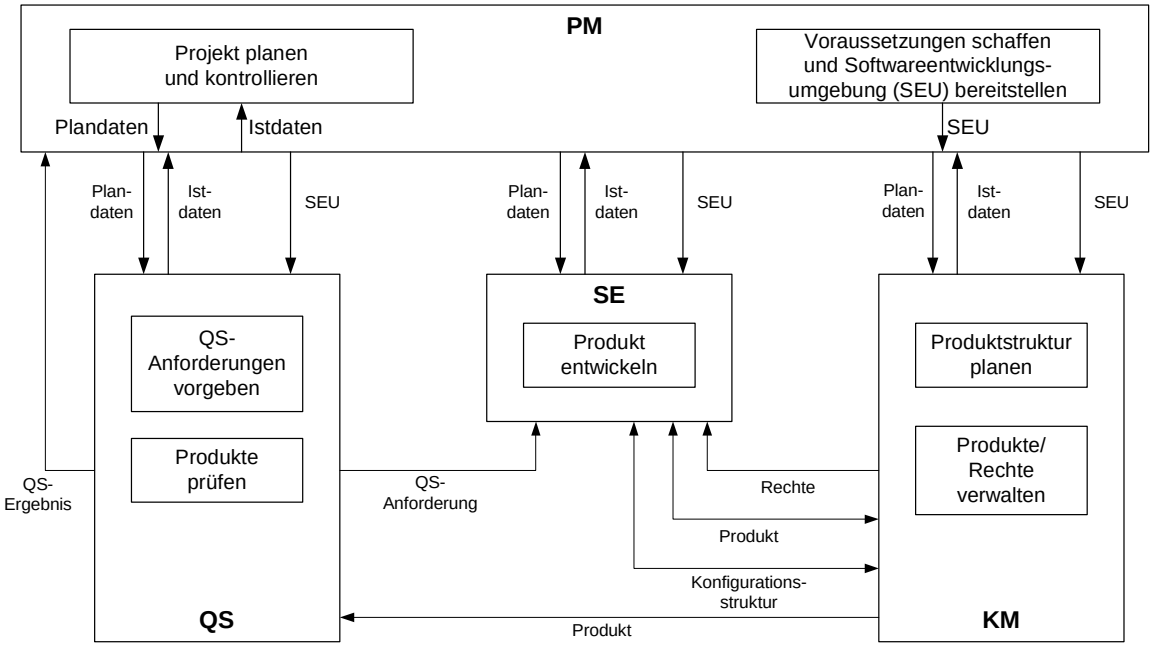


Bild 11: Prozessarchitektur vom V-Modell (Quelle: [o.V.1997b])

4.7 Prozesssteuerung

4.7.1 Catalysis

Catalysis beschreibt keine geschlossene Menge von durchzuführenden Aktivitäten oder zu erstellenden Ergebnissen. Vielmehr wird der Entwicklungsprozess durch eine Menge von Prozess-Mustern beschrieben, aus denen hervorgeht, in welcher Situation bzw. unter welchen Bedingungen bestimmte Arbeitsschritte durchzuführen sind. Es handelt sich demnach um eine entscheidungsorientierte Prozesssteuerung.

4.7.2 Perspective

Perspective basiert auf einer aktivitätsorientierten Prozesssteuerung. Jede der Phasen wird durch Angabe von durchzuführenden Aktivitäten und zugehörigen Unteraktivitäten beschrieben. Hierbei werden auf der Ebene der Aktivitäten ebenso die notwendigen Ergebnisse zur Bearbeitung der Aktivität sowie die zu erzeugenden Ergebnisse formuliert. Ergänzend enthält das Vorgehensmodell auch Aspekte einer ergebnisorientierten Prozesssteuerung. Beispielsweise wird in [AlFr1998, S. 278f., 311] beschrieben, welche Ergebnisse zu welchem Zeitpunkt in welchem Zustand vorliegen sollten. Diese nehmen allerdings im Vergleich zur aktivitätsorientierten Beschreibung nur eine untergeordnete Rolle ein.(anderer Meinung [NoSc1999, S. 177])

4.7.3 RUP

RUP ist primär aktivitätsorientiert definiert. Zu jedem Arbeitsablauf werden detaillierte Darstellungen vorgenommen, die eine schrittweise Abarbeitung ermöglichen. Ergänzend werden im Modell auch Aussagen über die benötigten und erzeugten Ergebnisse zusammengestellt. Dabei ist teilweise ersichtlich, welche Ergebnisse welche Zustände im Entwicklungsprozess zu durchlaufen haben.

4.7.4 V-Modell

Das V-Modell basiert vorwiegend auf einer aktivitätsorientierten Prozesssteuerung. Zu jedem der vier Submodelle werden jeweils die Hauptaktivitäten und deren Beziehungen bzw. Ablaufreihenfolgen genannt. Diese werden in weiterführenden Beschreibungen in Unteraktivitäten gegliedert. Darüber hinaus wird zu jeder durchzuführenden Aktivität beschrieben, welche Ergebnisse sich in welchen Zuständen befinden, wann die Aktivität gestartet werden kann bzw. durchgeführt worden ist. Mit anderen Worten wird ebenso z. T. eine ergebnisorientierte Prozesssteuerung unterstellt.

Bild 12 beschreibt zusammenfassend die verschiedenen Prinzipien der Prozesssteuerung der betrachteten Vorgehensmodelle.

	Catalysis	**Perspective**	**RUP**	**V-Modell**
Prozesssteuerung	entscheidungs-orientiert	aktivitäts-orientiert	aktivitäts-orientiert	aktivitäts-orientiert

Bild 12: Gegenüberstellung der Prozesssteuerung

4.8 Rollenabdeckung

4.8.1 Catalysis

In Catalysis werden keine Rollen definiert.

4.8.2 Perspective

Perspective legt hohen Wert darauf, dass die Entwicklungsarbeit in Teams durchgeführt wird, die jeweils aus nicht mehr als sechs Personen bestehen.[AlFr1998, S. 329-344] Innerhalb dieser Teams soll eine ausgewogenes Verhältnis zwischen fachlich und technisch ausgerichteten Rollen existieren. Prinzipiell wird zwischen Rollen im Solution- und im Component-Process sowie Rollen für die technische Infrastruktur unterschieden. Dabei werden insgesamt 21 Rollen definiert. Im Hinblick auf die komponentenorientierte Entwicklung sind folgende Rollen herauszuheben:

- Reuse Identifier: Aufgabe dieser Rolle ist es, im Rahmen des Solution Process Möglichkeiten zur Gestaltung wiederverwendbarer Dienste zu identifizieren sowie die Wiederverwendung existierender Dienste sicherzustellen.
- Reuse Librarian: Diese Rolle ist verantwortlich für den Betrieb eines Komponenten-Repositoriums, indem Komponenten archiviert und recherchiert werden.
- Reuse Assessor: Im Rahmen dieser Rolle werden neue Möglichkeiten der Wiederverwendung identifiziert, bewertet und vorhandene Möglichkeiten werden in die verschiedene Solution Process Projekte getragen.
- Reuse Architect: Diese Rolle entwickelt eine Gesamtvision, wie wiederverwendbare Komponenten zu entwickeln sind und wie diese sich in eine Gesamtarchitektur einfügen.

4.8.3 RUP

Insgesamt werden im RUP 30 Rollen definiert, die in fünf Gruppen zusammengestellt werden: Analysts, Developer, Testing Professionals, Manager und Other Workers. Dabei werden im Vorgehensmodell keine im Hinblick auf eine komponentenorientierte Entwicklung besonderen Rollen eingeführt. Rollen, die einen vermutlichen engen Bezug zur Komponentenerstellung haben, sind Architect, Designer, Implementer und Integrator. Bei der Beschreibung der Fähigkeiten und Kompetenzen dieser Rollen zeigt sich allerdings, dass zwar der Begriff Component verwendet wird. Dort allerdings in einer unspezifischen Bedeutung gebraucht wird. Daher wird die Beschreibung der eingeführten Rollen hier nicht weiter vertieft.

4.8.4 V-Modell

Insgesamt werden im V-Modell 23 Rollen definiert.[o.V.1997a, Teil 3: Handbuchsammlung – Rollenkonzept, S. A-1] Spezifische Rollen für die komponentenorientierte Entwicklung werden nicht definiert. Daher wird an dieser Stelle davon abgesehen, dass Rollenverständnis im V-Modell weiter zu vertiefen.

4.9 Adaption

4.9.1 Catalysis

Catalysis kann flexibel durch Einsatz und Anpassung von Prozess-Mustern (Process Patterns) gestaltet werden, ohne dass eine feste Aktivitätenreihenfolge vorgeschrieben wird. Es werden Erscheinungsformen des Prozesses für typische Projekttypen wie die Entwicklung ohne Wiederverwendung (Build Route) oder die Komposition von Komponenten (Assembly Route) dargestellt.[DSWi1998]. Es muss allerdings kritisch eingeräumt werden, dass diese Darstellungen eher exemplarischen Charakter haben. Damit geben sie nur verhältnismäßig wenig systematische Unterstützung.

4.9.2 Perspective

Im Vorgehensmodell Perspective wird darauf hingewiesen, dass das Modell an spezifische Belange angepasst werden kann und auch sollte.[AlFr1998, S. 260f.] Konkrete Aktivitäten werden allerdings nicht beschrieben.

4.9.3 RUP

Um den RUP an unternehmensspezifische Besonderheiten anzupassen, werden im Rahmen des Arbeitsablaufes Environment verschiedene Aktivitäten definiert, die beschreiben, wie eine Einführung des Prozesses in eine Organisation abgewickelt werden kann. Die Anpassungsmaßnahmen beruhen auf verschiedenen Assessment-Aktivitäten, die ermitteln, welche Kompetenzen, Schwachpunkte etc. eine Organisation bei der Softwareentwicklung besitzt. Ergänzend werden darüber hinaus konkrete sogenannte Roadmaps definiert, die eine bestimmte Adaption des Prozesses im Kontext eines speziellen Entwicklungsumfeldes beschreiben. Es werden neben der Adaption für eine komponentenorientierte Entwicklung ebenso Adaptionen für die Entwicklung von E-Business-Anwendungen sowie für die Entwicklung kleiner Projekte angeboten.

4.9.4 V-Modell

Das V-Modell gibt umfassende Hilfestellung für eine Adaption der Vorgehensweise. Zum einen beschreibt das Vorgehensmodell verschiedene Entwicklungsszenarien, die einen prinzipiellen Ablauf der Entwicklung demonstrieren (bspw. „Inkrementelle Entwicklung“, „Grand Design“, „Objektorientierte Entwicklung“, „Einsatz von Fertigprodukten“). Diese Szenarien geben einen Einblick in den Ablauf der Aktivitäten in einem bestimmten Entwicklungsrahmen.[o.V.1997a, Teil 3: Handbuchsammlung - Szenarien] Darüber hinaus wird ein umfassender Leitfaden beschrieben, aus dem hervorgeht, unter welchen Umständen bzw. Randbedingungen bestimmte Aktivitäten des Modells (bedingt) entfallen können bzw. zwingend durchgeführt werden müssen. Zur Typisierung von Anforderungen werden Merkmale wie Projektgrößeneinstufung, Komplexitätseinstufung von Daten und Funktionen, Quantifizierung der Wartbarkeitsanforderungen u. a. genannt.[o.V.1997a, Teil 3: Handbuchsammlung – Tailoring und projektspezifisches V-Modell]

5 Resümee und Ausblick

Bei den untersuchten komponentenorientierten Vorgehensmodellen handelt es sich sowohl um revolutionäre (Catalysis, Perspective) als auch um evolutionäre Modelle (RUP, V-Modell). Während RUP und V-Modell ebenso Projekt-, Qualitäts- und Konfigurationsmanagement unterstützen, behandeln Catalysis und Perspective ausschließlich die Entwicklung eines Softwaresystems. Einschränkend ist allerdings darauf hinzuweisen, dass V-Modell und RUP keine komponentenspezifischen Aspekte des Projekt-, Qualitäts- und Konfigurationsmanagements aufweisen. Auf die Unterscheidung zwischen der Entwicklung einer einzelnen Komponente und eines Komponentensystems wird insbesondere von Perspective hingewiesen, das für beide Fälle verschiedene Vorgehensmodelle beschreibt. Obwohl die ausgewählten Vorgehensmodelle speziell auf eine komponentenorientierte Entwicklung ausgerichtet sind, zeigt sich, dass wesentliche Phasen im Lebenszyklus einer Komponente von den Vorgehensmodellen nur rudimentär behandelt werden. Eine Ausnahme bildet hier Catalysis, wo umfangreiche Aktivitäten zur Standardisierung, zur technischen Anpassung sowie zur Komposition von Komponenten beschrieben werden. Spezifische Rollen für eine komponentenorientierte Entwicklung wie bspw. ein Komponentenbibliothekar werden nur von Perspective eingeführt. Daher ist zu vermuten, das speziell die Wiederauffindung von Komponenten von den anderen Vorgehensmodellen nur unbefriedigend ermöglicht wird. Die Adaption des Vorgehensmodells wird vom RUP und V-Modell umfassend, von Catalysis und Perspective nur unzureichend dargelegt.

Es verbleibt die Frage, welches Vorgehensmodell ein Softwarehersteller für die komponentenorientierte Entwicklung verwenden soll. Um eine fundierte Entscheidungsempfehlung aussprechen zu können, sind weitere Untersuchungen notwendig, die insbesondere auf empirischen oder experimentellen Methoden fußen sollten. Abgesehen von diversen Parametern wie bspw. Unternehmensgröße, Erfahrungshintergrund der Organisation, Projektgröße, Anwendungsdomäne etc. sollten vertiefende Untersuchungen insbesondere drei strategische Handlungsoption eines Softwareherstellers berücksichtigen:

- Komponentenentwickler: Der Softwarehersteller konzentriert sich ausschließlich auf die Entwicklung einzelner Komponenten und fungiert als Zulieferer für Komponenten.
- Komponentenassemblierer: Der Softwarehersteller konzentriert sich auf die Wiederauffindung, technische und fachliche Anpassung sowie auf die Komposition von Komponenten zu vollständigen Softwaresystemen.
- Komponentengeneralist: Der Softwarehersteller entwickelt auf Basis eines komponentenorientierten Architektur-Paradigmas und beherrscht alle Phasen des Lebenszyklus einer Komponente. Dabei ist allerdings die Menge der zugekauften bzw. fremdentwickelten Komponenten verhältnismäßig gering.

Es wird hier die Hypothese formuliert, dass für die unterschiedlichen Handlungsoptionen jeweils speziell angepasste Vorgehensmodelle benötigt werden, welche die jeweiligen Schwerpunkte der Tätigkeitsbereiche optimal unterstützen.

Literatur

[o.V.1997a] Das V-Modell - Allgemeiner Umdruck Nr. 250: Vorgehensmodell - Planung und Durchführung von IT-Vorhaben - Entwicklungsstandard für IT-Systeme des Bundes. http://www.v-modell.iabg.de/, Abruf am 2002-02-28. 1997a.

[o.V.1997b] Das V-Modell - Entwicklungsstandard für IT-Systeme des Bundes - Kurzbeschreibung. http://www.v-modell.iabg.de/, Abruf am 2002-02-28. 1997b.

[o.V.2001] Crystal software development methodologies. http://www.crystalmethodologies.org/, Abruf am 2002-02-28.

[Acke+2002] *Ackermann, J.; Brinkop, F.; Conrad, S.; Fettke, P.; Frick, A.; Glistau, E.; Jaekel, H.; Klein, U.; Kotlar, O.; Loos, P.; Mrech, H.; Ortner, E.; Overhage, S.; Sahm, S.; Schmietendorf, A.; Teschke, T.; Turowski, K.*: Vereinheitlichte Spezifikation von Fachkomponenten - Memorandum des Arbeitskreises 5.10.3 Komponentenorientierte betriebliche Anwendungssysteme. http://wi2.wiso.uni-augsburg.de/gi-memorandum.php.htm, Abruf am: 2002-03-01. Augsburg 2002.

[AlFr1998] *Allen, P.; Frost, S.*: Component-Based Development for Enterprise Systems - Applying the Select Perspective. Cambridge University Press, Cambridge 1998.

[Balz1998] *Balzert, H.*: Lehrbuch der Software-Technik - Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Heidelberg, Berlin 1998.

[Beck1999] *Beck, K.*: Embracing Change with Extreme Programming. In: IEEE Computer 32 (1999) 10, S. 70-77.

[Beck2000] *Beck, K.*: Extreme Programming Explained - Embrace Change. Addison-Wesley, Boston et al. 2000.

[BeFo2000] *Beck, K.; Fowler, M.*: Planning Extreme Programming. Boston et al. 2000.

[Brem1998] *Bremer, G.*: Genealogie von Entwicklungsschemata. In: *R. Kneuper; G. Müller-Luschnat; A. Oberweis (Hrsg.)*: Vorgehensmodelle für die betriebliche Anwendungsentwicklung. Teubner, Stuttgart, Leipzig 1998, S. 32-59.

[Burg+2000] *Burg, W. D.; Hawker, S.; Hale, D. P.; McInnis, K.; Parrish, A.; Sharpe, S.; Woolridge, R.*: Exploring a Comprehensive CBD Method - Use of CBD/e in Practice. Third International Workshop on Component-Based Software Engineering. 2000.

[DSWi1998] *D'Souza, D. F.; Wills, A. C.*: Objects, Components, and Frameworks with UML - The Catalysis Approach. Addison-Wesley, Reading, MA, et al. 1998.

[FeLo2000] *Fettke, P.; Loos, P.*: Komponentendokumentationen - Eine systematische Bewertung von Ordnungssystemen aus formaler Sicht. In: *K. Turowski (Hrsg.)*: Modellierung und Spezifikation von Fachkomponenten: Workshop im Rahmen der MobIS 2000 Modellierung betrieblicher Informationssysteme, Siegen, Deutschland, 12. Oktober 2000, Tagungsband. Siegen 2000, S. 51-70.

[FeLo2001] *Fettke, P.; Loos, P.* Fachkonzeptionelle Standardisierung von Fachkomponenten mit Ordnungssystemen - Ein Beitrag zur Lösung der Problematik der Wiederauffindbarkeit von Fachkomponenten. Working Papers of the Research Group Information Systems & Management, Paper 3. Chemnitz 2001.

[FBM+1998] *Fischer, T.; Biskup, H.; Müller-Luschnat, G.*: Begriffliche Grundlagen für Vorgehensmodelle. In: *R. Kneuper; G. Müller-Luschnat; A. Oberweis (Hrsg.)*: Vorgehensmodelle für die betriebliche Anwendungsentwicklung. Teubner, Stuttgart, Leipzig 1998, S. 13-31.

[GHJV1995] *Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.*: Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, et al. 1995.

[Grif1998] *Griffel, F.*: Componentware - Konzepte und Techniken eines Softwareparadigmas. Heidelberg 1998.

[HeCo2001] *Heineman, G. T.; Councill, W. T.*: Component-Based Software Engineering. Addison-Wesley, Boston et al. 2001.

[Hump1995] *Humphrey, W. S.*: A Discipline for Software Engineering. Addison-Wesley, Reading, MA, et al. 1995.

[JBR1998] *Jacobson, I.; Booch, G.; Rumbaugh, J.*: The Unified Software Development Process. Addison-Wesley, Reading, MA, et al. 1998.

[JCJÖ1992] *Jacobson, I.; Christerson, M.; Jonsson, P.; Övergaard, G.*: Object-Oriented Software Engineering. Addison-Wesley, Reading, MA, et al. 1992.

[McIn1999] *McInnis, K.* An Overview of CBD/e. http://www.cbd-hq.com/articles/1999/991115km_overviewcbde.asp, Abruf am 2002-02-28.

[NoSc1999] *Noack, J.; Schienmann, B.*: Objektorientierte Vorgehensmodelle im Vergleich. In: Informatik-Spektrum 22 (1999), S. 166-180.

[Rati2001] *Rational Software Corporation* Rational Unified Process. http://www.rational.com/tryit/rup/index.jsp, Abruf am 2002-03-01.

[Same1997] *Sametinger, J.*: Software Engineering with Reusable Components. Berlin et al. 1997.

[STR2000] *Sandmann, C.; Teschke, T.; Ritter, J.*: Ein Vorgehensmodell für die komponentenbasierte Anwendungsentwicklung. In: *B. Britzelmaier; S. Geberl (Hrsg.)*: Information als Erfolgsfaktor. Teubner Verlag, 2000.

[Schr2001] *Schryen, G.*: Komponentenorientierte Softwareentwicklung in Softwareunternehmen: Konzeption eines Vorgehenmodells zur Einführung und Etablierung. Deutscher Universitäts-Verlag, Wiesbaden 2001.

[Somm1996] *Sommerville, I.*: Software Engineering. 5. Aufl., Addison-Wesley, Harlow et al. 1996.

[Szyp1999] *Szyperski, C.*: Component Software - Beyond Object-Oriented Programming. Harlow, England, et al. 1999.

[Turo1999] *Turowski, K.*: Ordnungsrahmen für komponentenbasierte betriebliche Anwendungssysteme. In: *K. Turowski (Hrsg.)*: Tagungsband des 1. Workshops Komponentenorientierte betriebliche Anwendungssysteme (WKBA 1). Magdeburg 1999, S. 3-14.

[Turo2001] *Turowski, K.*: Fachkomponenten - Komponentenbasierte betriebliche Anwendungssysteme. Habil.-Schr. Magdeburg 2001.

Formularbasierte Benutzerinteraktion mit Fachkomponenten

Martin Gaedke, Martin Nussbaumer

Universität Karlsruhe (TH), Institut für Telematik, IT-Management und Web Engineering, Postfach 6980 76128 Karlsruhe, Deutschland, Tel: +49 (721) 608 – 8076, Email: {gaedke | nussbaumer}@tm.uni-karlsruhe.de, URL: http:// mw.tm.uni-karlsruhe.de

Zusammenfassung: Das World Wide Web hat sich zu der Plattform für verteilte Anwendungen gewandelt. Eine große Menge dieser Web Anwendungen dient der Realisierung betrieblicher Zielsetzungen. Für die Umsetzung der betrieblichen Abläufe liegt allen Web-Anwendungen ein einfaches, formularbasiertes Implementierungsmodell zugrunde, das es Benutzern ermöglicht mit den Fachkomponenten einer modernen Web Anwendung zu interagieren. Betriebliche Web Anwendungen sind daher durch komplexe Formularstrukturen geprägt. Durch den Fortschritt im Bereich der Darstellungsmedien sowie aus Gründen der Aktualität müssen Web Anwendungen jedoch häufig geändert und erweitert werden. Während Fachkomponenten hiervon zumeist nur selten betroffen sind, erschwert das zugrundeliegende Implementierungsmodell des Web die Wiederverwendung und Evolution der Formulare erheblich, da auf ihre strukturelle Information nach ihrer Abbildung in das grobgranulare Implementierungsmodell nicht mehr zugegriffen werden kann. Ohne einen disziplinierten Ansatz können die Anforderungen an betriebliche Web Anwendungen hinsichtlich der Interaktion mit Fachkomponenten somit nicht erfüllt werden. In diesem Beitrag wird der objektorientierte PetrIX Ansatz als durchgängige Unterstützung für die Modellierung und Realisierung von formularbasierten Web Anwendungen vorgestellt.

1 Einleitung

Mit der Standardisierung von HTML 2.0 [1995] für Dokumente im World Wide Web (Web) wurde die Auszeichnungssprache HTML 1995 um Formular-Elemente ergänzt. Die Auszeichnungssprache HTML beschrieb somit nicht nur die Darstellung von Dokumenten und ihrer Verknüpfung mit lokationsunabhängigen Ressourcen im Web [1990b], sondern ermöglichte erstmals die Beschreibung von Interaktion zwischen Benutzer und Web Anwendung mittels Formularen. Mit dem Fortschritt des Formulars als integraler Bestandteil der Dokumentbeschreibungssprache, wurde einem wachsenden Bedarf an betrieblichen Web Anwendungen Rechnung getragen [1999e].

Diesen Web Anwendungen ist gemeinsam, dass Interaktionstechnologien zum Einsatz kommen, die über bekannte Textein-/ausgabe-Verfahren und Mensch-Maschine-Schnittstellen herkömmlicher Software hinausgehen und nicht durch einfache softwaretechnische Maßnahmen am Anwendungscode handhabbar sind. Unter dem Einfluss des zunehmenden Wettbewerbs unterliegen diese Anwendungen einem ständigen Wandel; sei es bezüglich Funktionalität, Anwendungsschnittstellen oder der verfügbaren Informationen [1999a]. Infolgedessen werden die Änderungszyklen von Web Anwendungen immer kürzer. Die Anwendungen befinden sich ständig in einem Prozess der Wartung und Weiterentwicklung, welcher als *Evolution* bezeichnet wird [2000b].

Die kurzen Änderungszyklen an diesen komplexen Strukturen werden insbesondere durch die grobgranularen Beschreibungsmechanismen der Interaktionseinheiten und den damit verknüpften Programmkonstrukten erheblich beeinträchtigt. *Interaktionseinheiten* bezeichnen eine abgeschlossene Einheit, innerhalb der Interaktion zwischen Benutzer und Web Anwendung

ermöglicht wird. Solche Einheiten werden in HTML beispielsweise durch Formulare realisiert, die grobgranulare Beschreibungseinheiten darstellen, da sie nur als Ganzes wiederverwendet werden können. Eine *Interaktionsstruktur* bildet einen Kontext innerhalb einer solchen Interaktionsenheit, in der interaktive Vorgänge zueinander in Beziehung gesetzt werden. So können komplexe Interaktionseinheiten in einzelne, für den Benutzer überschaubare Gruppen von *Interaktionselementen*, wie z.B. Eingabefelder, Auswahllisten, etc., partitioniert werden. So können beispielsweise Gruppen von *Interaktionselementen*, wie z.B. Eingabefelder, Auswahllisten, etc., der Reihe nach navigiert werden, um komplexere Interaktionseinheiten in einzelne, für den Benutzer überschaubare zu partitionieren.

Eine feingranulare Betrachtung von Interaktionseinheiten mit Interaktionsstrukturen ist aber im WWW nicht möglich, da die aktuellen Auszeichnungssprachen HTML und XHTML [2000c] auf der Basis von Formularen eine solche Trennung nicht unterstützen. Durch die steigende Zahl der darstellenden Medien, wie z.B. Web-Browser, Personal Digital Assistant oder Mobiltelefon, muss jede einzelne Interaktionseinheit einer Web Anwendung zumeist in mehreren Ausprägungen realisiert werden, da sich diese Medien in ihren technologischen Möglichkeiten erheblich unterscheiden [1998b]. Eine Änderung am Interaktionsentwurf oder an der Fachkomponente, welche die Formulareingaben verarbeitet, hat mehrere Änderungen an den Implementierungen zur Folge. Somit wird die Evolution von Interaktionseinheiten insgesamt zu einer aufwändigen und fehleranfälligen Aufgabe, da eine entworfene Interaktionseinheit mehrere unterschiedliche, plattformabhängige Realisierungen ihrer Interaktionsstrukturen impliziert.

Trotzdem setzt sich nur langsam die Erkenntnis durch, dass eine disziplinierte Anwendung von Softwaretechnik für Entwicklung und Evolution von Benutzerinteraktion in Web Anwendungen erforderlich ist, um hierdurch die Qualität der Benutzschnittstelle zu steigern und die Kosten der Evolution zu verringern. In den folgenden Abschnitten werden zunächst die Benutzerinteraktion mit Fachkomponenten im Web untersucht sowie Ansätze zur Modellierung und Realisierung von Benutzerinteraktion für Web Anwendungen analysiert. Anschließend wird das in der Forschungsgruppe IT-Management und Web Engineering entwickelte Vorgehensmodell PetrIX vorgestellt, das die Benutzerinteraktion mit Fachkomponenten durchgängig objektorientiert modelliert. Am Beispiel der Web Anwendung webengineering.org Community wird der Ansatz exemplarisch für eine komplexe Fachkomponente verdeutlicht. Der Beitrag schließt mit einer Zusammenfassung und einem Ausblick auf aktuelle Arbeiten der Forschungsgruppe.

2 Interaktion mit Fachkomponenten in Web Anwendungen

Der Begriff der Interaktivität ist in der Literatur weit gestreut. So gibt es viele unterschiedliche Auffassungen darüber, was Interaktion eigentlich bedeuten soll. Oftmals wird das bloße Vorhandensein von Navigationsstrukturen in einem Dokument als „interaktiv“ bezeichnet. In [1990a] werden für die Interaktion zwischen zwei und mehr Personen, *zustandsoriente Aktionen* und *Kommunikation* gefordert. Während die Kommunikation zwischen Mensch und Web Anwendung durch das darstellende Medium implizit unterstützt wird, muss der Forderung nach zustandsorientierterten Aktionen softwaretechnisch Rechnung getragen werden. Hierauf aufbauend wird in [2002b] folgende Definition abgeleitet:

> *In einer formularbasierten Web Anwendung hat ein Benutzer die Möglichkeit durch eine Interaktionseinheit zu navigieren (Navigation), Daten zu modifizieren (Modifkation) und diese Änderungen innerhalb der Interaktionseinheit wieder sichtbar zu machen (Rückkopplung).*

Die in dieser Definition angeführten Ausprägungen von Interaktion, im folgenden *Interaktionsprimitive* genannt, sind Navigation, Modifikation und Rückkopplung. Diese Interaktionsprimitive, insbesondere Navigation, beziehen sich lediglich auf den formularbasierten Teil der Web Anwendung, also auf die Generierung der Parameter des durch die Fachkomponente

zur Verfügung gestellten Dienstes. Aufbauend auf dieser Definition bildet eine *Interaktionsstruktur* somit einen Kontext innerhalb dessen Interaktion auf Basis von Interaktionsprimitiven (Navigation, Modifikation und Rückkopplung) gebildet wird. Diese Interaktionsstrukturen werden von den Schnittstellen der Fachkomponente impliziert, welche die Daten und den Kontext zur Verfügung stellen. Interaktion wird hier nicht als die Interaktion zwischen Fachkomponenten verstanden. Vielmehr handelt es sich bei Interaktionsstrukturen um die abstrakte Definition einer Benutzerschnittstelle, welche die Parametrisierung eines von einer Fachkomponente zur Verfügung gestellten Dienstes vornimmt.

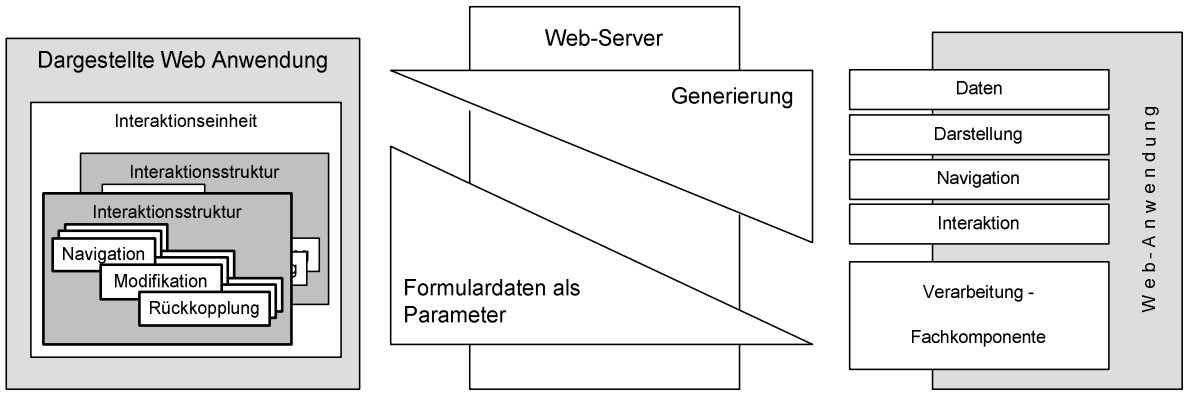


Bild 1: Zusammenhang von formularbasierter Benutzinteraktion und Fachkomponenten

Die Interaktion von Benutzern einer Web Anwendung mit verarbeitender Fachkomponente wird basierend auf dieser Definition in Bild 1 dargestellt. Das Interaktionsprimitiv *Rückkopplung* symbolisiert insbesondere die Möglichkeit, ein vom Benutzer *modifiziertes* Datum, sofort, bei allen Verwendungen innerhalb der Interaktionseinheit zu aktualisieren.

Die *dargestellte Web Anwendung* führt Daten sowie Darstellungs-, Navigations- und Interaktionscode in Web-Dokumenten zusammen und stellt die generierte Benutzerschnittstelle dem darstellenden Medium zur Verfügung. *Navigation, Modifikation* und *Rückkopplung* wird in der dargestellten Web Anwendung durchgeführt bzw. durch den ausführenden Browser ermöglicht.

Hierbei kann jede einzelne Modifikation innerhalb einer Modifikationssequenz bzw. Formularbearbeitung zu Rückkopplungen führen. Die Daten der finalen *Formularmodifikation* werden abschließend an die *Fachkomponente* der Web Anwendung zur Verarbeitung des implementierten Betriebsprozesses gesendet.

Die Komposition der fünf in Bild 1 dargestellten Komponenten werden als Service bezeichnet und stellen die Web-Fachkomponente einer betrieblichen Web Anwendung dar [1999b]. Ein Service beinhaltet einmal die Informationen, die eine Organisation mit der entsprechenden betrieblichen Dienstleistung einer definierten Anwendungsdomäne assoziiert. Ferner sind Darstellung und Navigation, mit denen diese Informationen zur Verfügung gestellt werden sollen, sowie Anweisungen darüber, welche Interaktionsmöglichkeiten die Benutzerschnittstelle anbieten soll, Bestandteil eines Services. Die Beschreibung der Umsetzung von (Geschäfts-)Prozessen durch eine Fachkomponente ist ein weiterer wichtiger Bestandteil. Er trennt die logische Verarbeitung von der konkreten Ausprägung der dargestellten Web Anwendung. Durch den Einsatz von Fachkomponenten zur Verarbeitung der Datenströme lassen sich Web Anwendungen somit in unterschiedlichen Ausprägungen darstellen ohne die eigentliche betriebliche Verarbeitungslogik ändern zu müssen.

Darstellung, Navigation und Interaktion eines Services sorgen dafür, dass die Dienstleistung als Ressource über das WWW zugänglich gemacht werden kann. Dabei braucht es sich nicht notwendigerweise um eine HTML-Ressource zu handeln. So lassen sich durch die Trennung von

Modell und Darstellung unterschiedliche Mensch-Maschine-Schnittstellen konkret Mensch-Fachkomponenten-Schnittstellen realisieren. Die Komposition solcher Services zu komplexen Web Anwendungen können durch den in [2000a] beschriebenen WebComposition-Ansatz diszipliniert entworfen und wiederverwendungsorientiert mit dem Evolutionsbus-Framework realisiert werden.

Für die abstrakte Betrachtung von Design und Implementierung von Benutzerinteraktion mit Fachkomponenten stellt die effektive Umsetzung auf eine Zielplattform somit die zentrale Problemstellung dar. Eine wichtige Anforderung, die an ein abstraktes Modell zur Beschreibung von solchen Interaktionseinheiten gestellt werden muss, ist daher die Möglichkeit der Abbildung auf ein vom Web unterstütztes Laufzeitsystem. Ferner muss eine abstrakte Beschreibung von Interaktionseinheiten folglich durch erweiterbare und partiell wiederverwendbare Interaktionsstrukturen unterstützt werden.

3 Ansätze zur Beschreibung von Interaktionsstrukturen

In der *Softwaretechnik* kommen zur Modellierung von Interaktionsstrukturen hauptsächlich zustandsorientierte Modelle zum Einsatz. Die Modellierung von Interaktion wird häufig als die Modellierung von interagierenden Objekten, wie im Interaktionsdiagramm der UML [1997a], aufgefasst. Benutzerspezifische Modelle wie die UML Use Cases berücksichtigen zwar den Benutzer, allerdings fokussieren sie eher auf die geforderte Funktionalität (Anforderungsanalyse), als dass sie interaktive Vorgänge als konkrete Abläufe beschreiben [1997b]. Daher eignen sich diese Modelle weniger, um die hier vorliegenden Interaktionsstrukturen zu spezifizieren, da der Benutzer hierbei explizit mit in Betracht gezogen werden muss. Für die Beschreibung von Interaktionsstrukturen im *Web Engineering* [1999c] muss somit einerseits der Benutzer explizit berücksichtigt werden, andererseits die Benutzerschnittstelle *abstrakt* bzw. darstellungsunabhängig modelliert werden können. Darüber hinaus muss Parallelität im Sinne von Benutzerinteraktion explizit in der Modellierung Berücksichtigung finden. Das bedeutet die Modellierung von gleichzeitig aktiven Benutzer-Optionen innerhalb eines Formulars muss auch als parallel zueinander erkennbar sein.

Weitergefasstere Modelle wie *Endliche Automaten* und Petri Netze bieten sich daher für die Modellierung der Interaktionsstrukturen eher an, da sie keine starre Auffassung über die zu modellierenden Elemente und deren Zusammenwirken haben. Vielmehr lassen sie sich aufgrund ihrer generischen Struktur besser auf die Bedürfnisse zur Modellierung von Benutzerinteraktionen anpassen.

Endliche Automaten sind prinzipiell gut geeignet, allerdings erfüllen sie nicht die Voraussetzung für das gewünschte Maß an Parallelität. Daher eignen sich Endliche Automaten nur begrenzt zur Modellierung von Benutzerinteraktionen.

Petri Netze [1962] hingegen sind für die Modellierung von Parallelität ausgelegt. Sie sind übersichtlich und lassen sich einfach benutzen und bedienen. Im Gegensatz zu endlichen Automaten ermöglichen sie auch das grafische Simulieren von interaktiven Abläufen. Allerdings erfüllen sie nicht die Anforderungen, die an ein erweiterbares Modell gestellt werden müssen. So können Teile eines Petri Netzes nur schwer wiederverwendet werden, da das Zusammenfassen von Stellen zu Gruppen bei Petri Netzen standardmäßig nicht unterstützt wird, und so ein Netz als atomare Einheit aufgefasst werden muss. Jedoch ermöglichen Petri Netze aufgrund der Objekte Stelle und Transition im Gegensatz zu endlichen Automaten einen einfachen Mechanismus für Assoziationen. Eine Verbindung zwischen Stellen und Transitionen hat lediglich eine Richtung und ist ansonsten unabhängig von einem Typ oder Wert, so dass semantische Informationen auf die Objekte Stelle und Transition verlagert werden können. Dadurch können Erweiterungen jeweils an diesen Objekten vorgenommen werden, und nicht, wie bei endlichen Automaten, an den Übergängen zwischen ihnen.

Um die Interaktionsstrukturen auf eine geeignete *Zielplattform* abzubilden, bedarf es darüberhinaus noch einer unterstützenden Implementierungstechnologie wie beispielsweise HTML [1999d], XHTML [2000c] oder WML [2000d]. Die in HTML wie auch XHTML und WML enthaltenen Mechanismen, um Formulare zu definieren sind nicht geeignet, da sie keine Entkopplung von ihrer Darstellung im Browser und ihrer Semantik zulassen. Vielmehr werden die Formularelemente nahtlos in die grobgranulare Dokumentstruktur des World Wide Webs eingefügt und vermischen sich so mit Teilen des Inhalts und der Darstellung. Dedizierte Implementierungen von Formularen unter Einsatz der Technologie DHTML ermöglichen zwar eine teilweise Entkopplung zwischen den Interaktionselementen und deren Semantik, erhöhen allerdings den Aufwand, um Änderungen an den Formularen durchzuführen.

Eine feingranulare Betrachtung von Interaktionselementen ist mit Hilfe von XForms [2002a] möglich. Aktuell befindet sich XForms noch im Standardisierungsprozess des W3C, der die Lücke zwischen der eXtensible Markup Language (XML) [1998a] und den bisherigen HTML Formularen schließen soll. XForms bildet durch die Trennung zwischen Inhalt (Model) und Layout (Form Control) eine geeignete Ausgangsbasis für die Implementierung von Interaktionsstrukturen. Ferner verfügt XForms über die Möglichkeit dynamische Teile zu definieren, die über zustandsorientierte Benutzeraktionen aktiviert werden können. Als Anwendung der XML erbt XForms zugleich die Vorteile von XML, wie beispielsweise die Möglichkeit der einfachen Überprüfung auf Korrektheit und Validität. Damit lassen sich Gruppen von Interaktionselementen Web-konform zusammenfassen und es wird eine Strukturierung von Formularen, also den Interaktionseinheiten, ermöglicht. Durch die Unterstützung von XML als Datencontainer (*XForms instance*) lassen sich Web Anwendungen mit XForms bis auf die Ebene der Interaktion in XML spezifizieren. Dedizierte Programme, die nur dem Zweck der Abbildung oder des Auslesens von Formulardaten in ihrer XML-Repräsentation dienen, werden damit obsolet. Das XForms-Model ist plattformunabhängig und kann für unterschiedliche Darstellungsmedien wie PDA oder Mobiltelefon wiederverwendet werden.

4 PetrIX – Design und Abbildung von Interaktionsstrukturen

Um eine Trennung von abstraktem Design und konkreter Implementierung zu erhalten, muss eine Trennung zwischen einem *abstrakten Modell* und einem *konkreten Modell* durchgeführt werden. Durch eine solche Trennung werden die abstrakten Modelleigenschaften der Interaktionsstrukturen bezüglich ihrer Semantik hervorgehoben und explizit von einer konkreten Implementierung getrennt. Dadurch ergibt sich unter anderem auch der Vorteil der plattformunabhängigen Betrachtung von Interaktionsstrukturen.

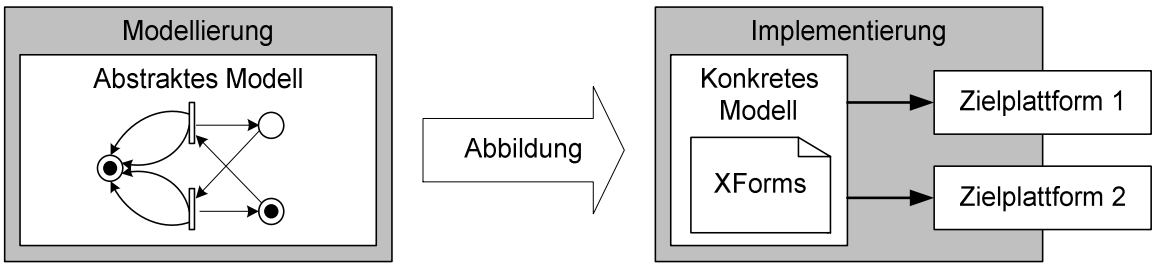


Bild 2: Zusammenhang zwischen abstraktem und konkretem Modell

Das PetrIX Vorgehensmodell unterstützt durchgängig die Benutzerinteraktion mit Fachkomponenten von der Modellierung bis hin zur Abbildung auf eine Zielplattform. Es stellt hierfür ein konkretes und abstraktes Modell zur Verfügung. Die abstrakte Modellierung von Interaktionsstrukturen basiert hierbei auf erweiterten Petri Netzen. Für das konkrete Modell wird

XForms verwendet. Bild 2 stellt die Abbildung des abstrakten auf das konkrete Modell dar. Der Abbildungsmechanismus nutzt die Kontextinformationen der Interaktionsstruktur, um die Generierung für die entsprechende Zielplattform zu ermöglichen.

Ein wesentlicher Bestandteil dieser konkreten Implementierung ist dabei die Umsetzung der Interaktionssemantik auf darstellende Formularelemente und die Bindung von Formularelementen an die Attribute des zu manipulierenden Datenobjekts einer Fachkomponente.

5 Objektorientierte Modellierung und Termersetzung mit PetrIX

Im ersten Schritt von PetrIX werden die Interaktionsstrukturen mit Hilfe von Petri Netzen dargestellt. Aufgrund der Komplexität dieser Strukturen werden die Petri Netze in einen objektorientierten Kontext gebettet, der es ermöglicht Strukturen verschachtelt darzustellen. Mit Hilfe dieser Objektstruktur lassen sich die Petri Netze serialisieren. Das bedeutet, dass die instantiierten Objekte als lineare Sequenz aufgefasst werden, die in einen korrespondierenden Term einer Termalgebra übersetzt werden. Ein wesentlicher Vorteil einer Termalgebra ist neben der Möglichkeit der algorithmischen Optimierung auch der Einsatz eines Termersetzungssystems. Mit Hilfe eines Termersetzungssystems lassen sich zuvor generierte Terme, die serialisierten Petri Netze repräsentieren, leicht in eine konkrete Implementierungssprache abbilden.

Bild 3 zeigt zwei Petri Netze, die Interaktionsstrukturen darstellen. Die üblichen Konventionen eines Petri Netzes werden beibehalten: eine Stelle repräsentiert einen Zustand, eine Transition einen Zustandsübergang. In PetrIX werden Zustände als Werte und Zustandsübergänge als Übergänge zwischen diesen Werten behandelt. Eine Stelle wird durch einen eindeutigen Namen und Typ dargestellt. Ist die Stelle vom Typ *InteractionStructure*, so kann sie wieder weitere Stellen aufnehmen. In Bild 3 sind zwei Interaktionsstrukturen dargestellt. Links dargestellt ist die Stelle reviewStep vom Typ *ISConcat*, einer Interaktionsstruktur, verbunden mit einer Transition Submit, welche die Beendigung einer Interaktionseinheit beschreibt. Rechts davon dargestellt ist die Verfeinerung von reviewStep mit den darin enthaltenen Stellen rankPaper (Typ *InteractionElement*) und *makeComment* (Typ *ISToggle*). Da es sich bei rankPaper um ein Interaktionselement handelt, kann diese Stelle nicht weiter verfeinert werden. Die Interaktionsstruktur *ISToggle* wird an späterer Stelle vertieft.

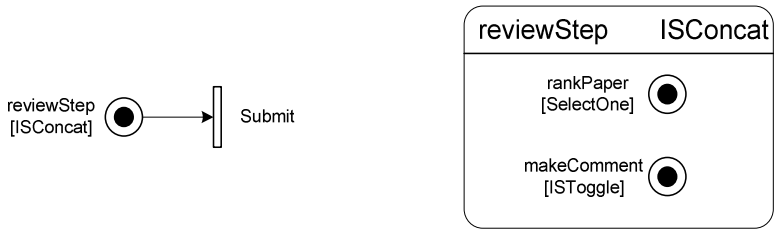


Bild 3: Abstrakte Darstellung einer Interaktionsstruktur als Petri Netz

Aufgrund des objektorientierten Kontextes, in den die Petri Netze gebettet werden, können sie besser an die sich ständig ändernden Bedingungen angepasst werden. Dadurch können änderungsrelevante Teile leichter identifiziert und modifiziert werden. Ferner wird in PetrIX durch die objektorientierte Modellierung von Petri Netzen eine „Netz-im-Netz“ Struktur, also die Schachtelung von Interaktionsstrukturen, ermöglicht. Die so definierten hierarchischen Petri Netze gewährleisten den Entwurf von Interaktionsstrukturen mittels schrittweiser Verfeinerung. Der Grad an Wiederverwendung kann bis auf die Granularität einer Stelle reichen. Zudem können Stellen gruppiert werden und solche Gruppen dynamisch aktiviert werden.

Bild 4 zeigt exemplarisch wie das Objekt Stelle als Basisklasse für den objektorientierten Entwurf benutzt werden muss. Auf diese Weise lassen sich beliebige Interaktionsstrukturen definieren und hinzufügen.

Für die von *InteractionStructure* und *InteractionElement* abgeleiteten konkreten Klassen existieren jeweils korrespondierende Transitionsklassen. Im Zusammenspiel zwischen diesen Objekten können die Interaktionsprimitive in den von der Interaktionsstruktur vorgegebenen Kontext gesetzt werden.

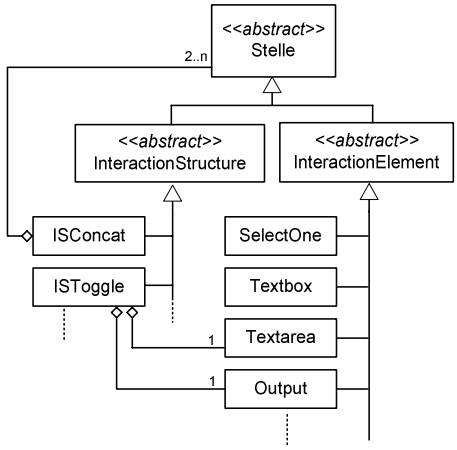


Bild 4: Das Objektmodell Stelle eines Petri Netzes

Zusätzlich müssen neben diesen strukturellen Beziehungen zwischen den Objekten auch die semantischen Beziehungen betrachtet werden. Die semantischen Informationen können beispielsweise zur Fehlerüberprüfung genutzt werden. Aufgrund der „Netz-im-Netz"-Struktur muss zudem noch die Erreichbarkeit von verschachtelten Stellen berechnet werden.

Bild 5 zeigt die Verfeinerung der Stelle makeComment aus dem Petri Netz in Bild 3. Die Interaktionsstruktur *ISToggle* ermöglicht das Hin- und Herschalten zwischen den Zuständen Editieren eines Kommentars *(*Interaktionselement *Textarea)* und Ansicht des Kommentars *(*Interaktinoselement *Output).*

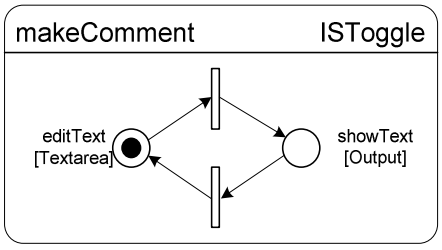
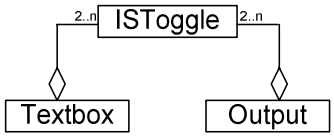


Bild 5: Verfeinerung der Stelle *makeComment*

Tabelle 1 verdeutlicht das Vorgehen, wie das in Bild 5 dargestellte Petri Netz (abstraktes Modell), das dem Objektmodell in Bild 4 genügt, in ein XForms-Programm (konkretes Modell) überführt werden kann. Aufgrund des objektorientierten Kontextes, in den die Petri Netze eingebettet sind, lassen sich die Petri Netze als Objekthierarchien auffassen. Mit Hilfe einer kontextfreien Grammatik läßt sich die Struktur dieser Objekthierarchien beschreiben. Dabei werden die objektorientierten Strukturen sukzessive auf die Regeln einer kontextfreien Grammatik abgebildet (Tabelle 1, Schritt 1 und Schritt 2). Durch Erweitern der Regeln dieser Grammatik um die Operationen einer Termalgebra können sie als formale Beschreibung eines Übersetzers zwischen

Petri Netzen und einer konkreten Zielplattform verwendet werden. In Tabelle 1 wird das Hinzufügen der termalgebraischen Operation (toggle) als Erweiterung für die Regel der kontextfreien Grammatik *ISToggle* verdeutlicht (vgl. Tabelle 1, Schritt 3) .

Für die Abbildung auf die konkrete Plattform muss abschließend noch ein regelbasiertes Termersetzungssystem für diese Plattform spezifiziert werden. Interaktionsstrukturen können auf diese Weise leicht an das Implementierungsmodell einer Fachkomponente angepasst werden. In Tabelle 1 (Schritt 4) wird das Implementierungsmodell von XForms benutzt, um den produzierten Term zu ersetzen.

1	Objektorientiertes Muster	
2	Regel der kontextfreien Grammatik	ISToggle := Textbox Output
3	um termalgebraische Operation erweiterte Regel	ISToggle := toggle (Textbox, Output)

4	Termersetzungsregel	`toggle(t1,t2) → <switch>`
		`<case id="t1"> eval(t1)</case>`
		`<case id="t2"> eval(t2)</case>`
		`</switch>`

Tabelle 1: Abbilden der Objektstruktur in eine Regel einer kontextfreien Grammatik

6 Beispiel – WebEngineering.org

Im Folgenden wird der Einsatz von PetrIX an dem realen Beispiel der Web Anwendung WebEngineering.org Community aufgezeigt. WebEngineering.org bietet Dienste zur Unterstützung der Web Engineering Community. Diese Dienste sind als Fachkomponenten realisiert und werden als Services über die Web Anwendung zugänglich gemacht. Eine solche komplexe und interaktionsreiche Fachkomponente ist ein *Peer-Review-System*, das zur Unterstützung der Begutachtung und Verwaltung von Konferenzbeiträgen im Rahmen von WebEngineering.org entwickelt wurde. Die Fachkomponente kann in unterschiedlichen Szenarien eingesetzt werden, wobei im Wesentlichen der Begutachtungsprozess bzw. die Interaktionsmöglichkeiten für den Benutzer konfiguriert werden müssen.

Die in Bild 6 dargestellte Benutzerschnittstelle zeigt einen Ausschnitt eines solchen konfigurierten Konferenz Systems. Die abstrakten Strukturen der dargestellten Benutzerinteraktion sind das *Bestimmen der Relevanz eines Beitrages* sowie dessen *Kommentierung*. Die Petri Netze in Bild 3 und Bild 5 entsprechen diesen Strukturen. Während das *Bestimmen der Relevanz* ein Interaktionselement darstellt, lässt sich die Struktur der Beitragskommentierung noch weiter verfeinern. Wenn der Text für den Kommentar eingegeben worden ist, soll das dafür vorgesehene Interaktionselement durch den Text des Kommentars in der Ausprägung als Hyperlink ersetzt werden (vgl. Bild 6: Schritt 1 und 2). Durch Aktivieren dieses Links soll wieder das Interaktionselement angezeigt werden, usw. Auf diese Weise wird ein Hin- und Herschalten (*engl. to toggle*) zwischen den beiden Interaktionselementen realisiert, das sich in der grafischen Entsprechung als Petri Netz wiederfindet (Bild 5). Der Vorteil einer solchen Struktur ist, dass schon

bearbeitete Teile eines (umfangreichen) Formulars als solche sofort erkannt werden und die Aufmerksamkeit des Benutzers auf die noch zu bearbeitenden Teile des Formulars gelenkt wird.

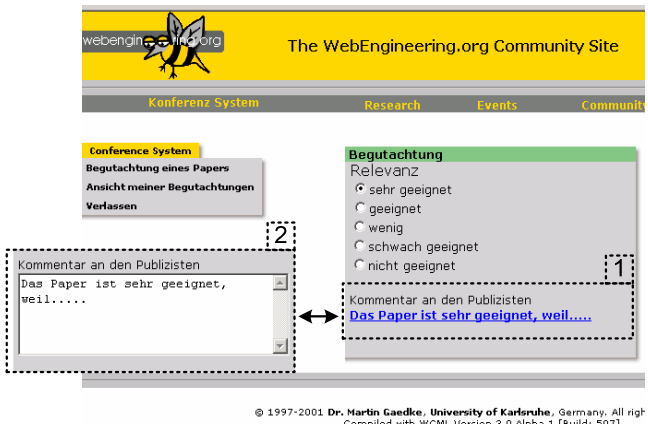


Bild 6: Benutzerschnittstelle des Peer-Review-Systems in WebEngineering.org

Aus der grafischen Darstellung der Interaktionsstrukturen lässt sich der Interaktions-Term, wie in Tabelle 1 beschrieben, bilden:

$\mathtt{ISConcat(IE_{SelectOne},ISToggle(IE_{textarea},IE_{output}))}.$

Durch die regelbasierte Termersetzung kann dieser Term nun weiter in die Zielsprache, im vorliegenden Fall Xforms, übersetzt werden.

```
<xforms_program>
  <xform id="model">
    <instance>
      <ranking>1</ranking>
      <comment>please write a comment</comment>
    </instance>
  </xform>
  <selectOne xform="model" ref="ranking" selectUI="radioGroup">
    <item value="1">sehr geeignet</item>
    ...
  </selectOne>
  <switch xform="model">
    <case id="show">
      <output xform="model" ref="comment"/> ...
    </case>
    <case id="edit">
      <textarea xform="model" ref="comment"/> ...
    </case>
  </switch>
</xforms_program>
```

Das Beispiel zeigt den übersetzten Term als XForms-Programm. Aufgrund der Aktualität von XForms existieren augenblicklich nur wenige Implementierungen von XForms-Prozessoren. Im Rahmen der Arbeiten wurde ein prototypischer XForms-Prozessor entwickelt, der zur Validierung für den erzeugten XForms-Programmcode benutzt wurde.

7 Zusammenfassung und Ausblick

In diesem Beitrag wurde das objektorientierte Vorgehensmodell PetrIX zur Unterstützung von Entwurf und Realisierung von formularbasierter Benutzerinteraktion mit Fachkomponenten vorgestellt. Durch die Abstraktion mittels darstellungsunabhängiger Interaktionsstrukturen, können diese für unterschiedliche Formulardarstellungen wiederverwendet werden. Durch den objektorientierten Ansatz lassen sich auch Teile von Interaktionsstrukturen gezielt wiederverwenden und an die durch Evolution geprägten Web Anwendungen und Fachkomponenten anpassen. Dadurch steigt sowohl die Qualität der Benutzerschnittstelle als auch die Produktivität im Softwareprozess. Durch die Abbildung auf XForms des World Wide Web Consortiums kann der Entwurfsprozess durchgängig mit Einsatz von XML gestaltet werden.

Zur Zeit wird in der Forschungsgruppe IT-Management und Web Engineering der Universität Karlsruhe ein Unterstützungswerkzeug für die grafische Modellierung von abstrakten Interaktionsstrukturen mit Petri Netzen entwickelt. Auf diese Weise können Interaktionsstrukturen grafisch entworfen, automatisch validiert und übersetzt werden. Aufgrund der semantischen Beziehungen zwischen Stellen und Transitionen eines Petri Netzes in PetrIX können Fehler schon während des Entwurfs lokalisiert werden. Dadurch soll die Qualität des Entwurfprozesses weiter verbessert und die Produktivität erhöht werden.

8 PetrIX-Werkzeuge

Der PetrIX XForms-Compiler sowie die WebComposition Laufzeitumgebung und weitere Informationen zum Evolutionsbus stehen im Web zur Verfügung:

http://webe.tm.uni-karlsruhe.de/

9 Literatur

[1990a] Bahrdt, H. P.: Schlüsselbegriffe der Soziologie. Eine Einführung mit Lehrbeispielen. C.H. Beck, München 1990a.

[1990b] Berners-Lee, T. Information Management: A Proposal. http://www.w3.org/Proposal.html, Abruf am 1998-10.10.1998.

[1995] Berners-Lee, T.; Connolly, D. Hypertext Markup Language (HTML) 2.0. RFC Nr. 1866, 1995.

[1997a] Booch, G.; Jacobson, I.; Rumbaugh, J. The Unified Modeling Language for Object-Oriented Development. http://www.rational.com/uml, Abruf.

[1998a] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M. Extensible Markup Language (XML) 1.0. Recommendation Nr. REC-xml-1998021, World Wide Web Consortium (W3C) 1998a.

[1999a] Cusumano, M. A.; Yoffie, D. B.: Software Development on Internet Time. In: IEEE Computer 32 (1999a) 10, S. 60-69.

[2002a] Dubinko, M.; Dietl, J.; Merrick, R.; Raggett, D.; Raman, T. V.; Welsh, L. B. XForms 1.0: W3C Working Draft 08 June 2001. Draft Nr. WD-xforms-20010608, World Wide Web Consortium (W3C) 2002a.

[1997b] Fowler, M.; Scott, K.: UML distilled: applying the standard object modeling language. Addison Wesley Longman, Reading, Mass. 1997b.

[2000a] Gaedke, M.: Komponententechnik für Entwicklung und Evolution von Anwendungen im World Wide Web. Shaker Verlag, Aachen 2000a.

[1998b] Gaedke, M.; Beigl, M.; Gellersen, H.-W.; Segor, C.: Web Content Delivery to Heterogeneous Mobile Platforms. In: Lecture Notes in Computer Science (LNCS), Springer Verlag 1552 (1998b) Advances in Database Technologies, S. 205-217.

[2000b] Gaedke, M.; Graef, G.: WebComposition Process Model: Ein Vorgehensmodell zur Entwicklung und Evolution von Web-Anwendungen. In: R. G. Flatscher; K. Turowski (Hrsg.): 2. Workshop Komponentenorientierte betriebliche Anwendungssysteme (WKBA 2). Wien, Austria 2000b, S. 21-38.

[1999b] Gaedke, M.; Turowski, K.: Framework for Maintaining Evolution of E-Commerce Applications in the Web. In: J.-C. Rault (Hrsg.): 12th International Conference - Software and Systems Engineering and their Applications (ICSSEA '99). Bd. 5, Paris, France 1999b, S. 18.14.11-18.14.10.

[1999c] Muragesan, S.: Web Engineering. In: SIGWEB Newsletter 8 (1999c) 3, S. 28-32.

[2002b] Nussbaumer, M.: Einsatz von Petri Netzen und Xforms zur Modellierung und Unterstützung von interaktiven Web-Anwendungen. Diplomarbeit, Universität Karlsruhe. Karlsruhe 2002b.

[2000c] Pemberton, S.; Altheim, M.; Austin, D.; Boumphrey, F.; Burger, J.; Donoho, A. W.; Dooley, S.; Hofrichter, K.; Hoschka, P.; Ishikawa, M.; ten Kate, W.; King, P.; Klante, P.; Matsui, S. i.; McCarron, S.; Navarro, A.; Nies, Z.; Raggett, D.; Schmitz, P.; Schnitzenbaumer, S.; Stark, P.; Wilson, C.; Wugofski, T.; Zigmond, D. XHTML 1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4 in XML 1.0. Recommendation Nr. REC-xhtml1-20000126, World Wide Web Consortium (W3C) 2000c.

[1962] Petri, C. A.: Kommunikation mit Automaten. Dissertation, Technischen Universität Darmstadt. Darmstadt 1962.

[1999d] Raggett, D.; Le Hors, A. HTML 4.0 Specification. http://www.w3.org/TR/html4, Abruf am 1999-24. December.

[1999e] Turowski, K. (Hrsg.): Tagungsband des 1. Workshops Komponentenorientierte betriebliche Anwendungssysteme (WKBA 1). Otto-von-Guericke-Universität Magdeburg, Magdeburg 1999e.

[2000d] WAP Forum. Wireless Application Protocol Wireless Markup Language Specification Version 1.3. Specification Nr. SPEC-WML-20000219, 2000d.

Konzeption eines Repositories zur Unterstützung der Wiederverwendung von Software-Komponenten

Oliver Höß[+], Anette Weisbecker[+]

[+] *Fraunhofer-Institut für Arbeitswirtschaft und Organisation (IAO) und Institut für Arbeitswissenschaft und Technologiemanagement (IAT) der Universität Stuttgart, Competence Center Software-Management, Nobelstr. 12, 70569 Stuttgart, Deutschland, Tel.: +49 (711) 970 - 2409 bzw. - 2400, Fax: +49 (711) 970 - 2401, E-Mail:{Oliver.Hoess|Anette.Weisbecker}@iao.fhg.de, URL: http://www.sw-management.iao.fhg.de*

Zusammenfassung. Der Ansatz der komponentenbasierten Softwareentwicklung kann nur erfolgreich umgesetzt werden, wenn ein geeignet großer Vorrat an wiederverwendbaren Komponenten existiert und diese auch benutzt werden können. In der Praxis ist dies oft nicht der Fall. Der vorliegende Beitrag zeigt in einem Überblick Hemmnisse auf, die die Wiederverwendung von Software-Komponenten be- oder verhindern. Ein Komponenten-Repository, d.h. ein Werkzeug, mit dem Komponenten innerhalb eines Unternehmens gespeichert, verwaltet und wiedergefunden werden können, stellt einen wesentlichen Beitrag zur Lösung eines Teils der Problematik dar. Daher werden die unterschiedlichen fachlichen und technischen Anforderungen an ein derartiges Repository sowie ein möglicher Realisierungsansatz dargestellt.

Schlüsselworte: Wiederverwendung; Komponenten-Repository; Komponentenbasierte Softwareentwicklung; Wissensmanagement in der Softwareentwicklung

1 Einleitung

Sowohl in der Wissenschaft als auch in der Praxis hat sich die Erkenntnis durchgesetzt, dass die komponentenbasierte Softwareentwicklung das derzeit geeignete Paradigma ist, die gestiegenen Anforderungen an moderne Software hinsichtlich

- Entwicklungsdauer,
- Entwicklungskosten und
- Software-Qualität

erfüllen zu können (siehe u.a. auch [FWH+2000] oder [HöWe2001]).

Die Wiederverwendung von Software-Komponenten ist dabei einer der wichtigsten Aspekte bei diesem Ansatz [Same1997]. Nur wenn Komponenten entsprechend häufig wiederverwendet werden, lohnt sich der zusätzliche Aufwand, der benötigt wird, um eine Komponente wiederverwendbar zu machen. Dieser Zusatz-Aufwand kann bis auf das 3-fache des normalerweise üblichen Aufwands steigen [JaGJ1997]. Eine genaue wirtschaftliche Berechnung ist hierbei schwierig, da eine Vielzahl von unterschiedlichen, teilweise nicht genau quantifizierbaren Faktoren eine Rolle spielt.

Die Umsetzung der Wiederverwendung und damit des Komponenten-Ansatzes wird derzeit

durch mehrere Faktoren begünstigt. Zum einen wurden standardisierte Komponenten-Technologien entwickelt, zum anderen hat die Verbreitung des Internets die Entstehung von Komponenten-Märkten ermöglicht [Behl2000].

Derzeit stehen eine Reihe von Komponenten-Technologien und Architekturmodellen zur Verfügung [WAD+2000]. Die derzeit wichtigsten sind Sun's Java 2 Enterprise Edition (J2EE) mit den Komponenten-Technologien JavaBeans und Enterprise JavaBeans, Microsoft's .NET mit den Komponenten-Technologien ActiveX und COM+ sowie der CORBA 3.0-Standard mit dem CORBA Component Model[1]. Durch diese Komponenten-Technologien wird die Wiederverwendung zumindest auf technischer Ebene vereinfacht.

Im Umfeld der Komponenten-Märkte[2] fällt auf, dass dort vorwiegend technische, domänenunabhängig einsetzbare Komponenten angeboten werden. Fachliche, domänenspezifische Komponenten sind dort selten zu finden. Um jedoch eine Steigerung der Effizienz in der Entwicklung von betrieblichen Anwendungssystemen zu erreichen, ist ein ausreichendes Angebot an wiederverwendbaren Fachkomponenten von ausschlaggebender Bedeutung [WeHS2001].

Insgesamt ist trotz dieser begünstigenden Faktoren die Wiederverwendung von Software-Komponenten in der betrieblichen Praxis von Unternehmen im industriellen Bereich und im Dienstleistungssektor bei weitem noch nicht optimal umgesetzt. Dies wird auch durch aktuelle Studien belegt (z.B. [DiEs2001] und [GfK+2000]).

Das Fraunhofer IAO hat in einer Reihe von Forschungsprojekten mit Industriebeteiligung (z.B. PROMPT [WeGr1998]) sowie in Projekten im Auftrag von Software-Herstellern und software-produzierenden Einheiten in Unternehmen ähnliche Erfahrungen gemacht. Wenn man den Durchschnitt aller Unternehmen betrachtet, wird nach dem in Bild 1 dargestellten Klassifikationsschema von [JaGJ1997] nach diesen Erfahrungen die Stufe 4 nicht erreicht.

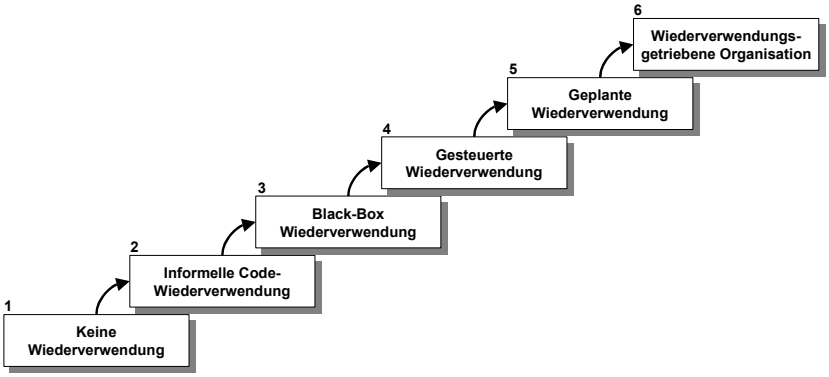


Bild 1: Reifestufen der Wiederverwendung (nach [JaGJ1997])

Dieser Beitrag zeigt nun in Abschnitt 2 eine Reihe von Hemmnissen auf, die die Wiederverwendung von Software-Komponenten be- oder verhindern. Ein wesentlicher Bestandteil einer Lösung zur Überwindung dieser Hemmnisse stellt dabei die Einführung und Nutzung eines

[1] Derzeit steht noch keine Implementierung des CORBA Component Models als Produkt zur Verfügung.

[2] Beispielsweise ComponentSource (www.componentsource.com) oder Flashline (www.flashline.com).

Komponenten-Repositories dar, das die Speicherung, Verwaltung und das Wiederfinden von gespeicherten Komponenten unterstützt. Basierend auf den Hauptprozessen der komponentenbasierten Softwareentwicklung (Abschnitt 3) werden in Abschnitt 4 Anforderungen an ein Komponenten-Repository formuliert sowie entsprechende Dienste definiert. Für die Umsetzung dieser Anforderungen und Dienste wird in Abschnitt 5 eine mögliche Systemarchitektur dargestellt. Der Beitrag wird dann mit einem Ausblick auf zukünftige Entwicklungen abgeschlossen.

2 Hemmnisse der Wiederverwendung von Software-Komponenten

Die Hemmnisse, die bei der Wiederverwendung von Software-Komponenten auftreten, sind von vielschichtiger Natur. In Bild 2 ist eine Übersicht der unterschiedlichen Kategorien dargestellt, die anschließend kurz erläutert werden. Dadurch soll die Vielfalt der Hemmnisse deutlich werden. Eine ausführlichere Darstellung ist in [WeHS2001] oder in [Reif1997] enthalten.

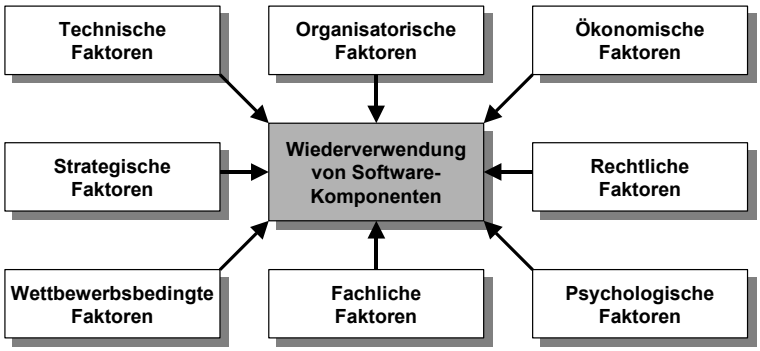


Bild 2: Hemmnisse bei der Wiederverwendung von Software-Komponenten

2.1 Technische Faktoren

Durch die in der Einleitung beschriebenen standardisierten Komponenten-Technologien (siehe auch [WAD+2000]) können viele Probleme auf technischer Ebene gelöst werden. Auch für den Fall, dass Komponenten unterschiedlicher Technologien miteinander kombiniert werden müssen, können durch Anwendung von „Wrapping"-Techniken Lösungen herbeigeführt werden.

Ebenfalls in den technischen Bereich fällt jedoch die meist nicht vorhandene Möglichkeit zur unternehmensweiten Speicherung, Verwaltung und Recherche von Komponenten in einem Komponenten-Repository. Dies wird auch in der Studie [DiEs2001] deutlich, bei der Punkte wie z.B.

- unzureichende Werkzeugunterstützung,
- eine unzureichende Verwaltung von Komponenten sowie
- Probleme beim Finden von Komponenten

als Hauptkritikpunkte angegeben werden.

2.2 Organisatorische Faktoren

Der wesentliche Faktor im Bereich der Organisation ist das häufige Fehlen entsprechender Stellen, die für die Wiederverwendung von Software-Komponenten verantwortlich sind. Die Hauptaufgabe eines derartigen „Reuse-Teams“ [WeGr1998] ist dabei sicherlich die Installation sowie die Pflege und Wartung des in Abschnitt 2.1 beschriebenen Komponenten-Repositories. Dies schließt auch die Evaluation und die Integration von extern erworbenen Komponenten mit ein.

Um auch die Integration in die Projekt-Struktur des jeweiligen Unternehmens sicherzustellen, sollten die Mitglieder dieses Teams auch in einer beratenden Funktion in den Projekten mitarbeiten um dort insbesondere in den Bereichen Wiederverwendung und Software-Architektur hohe Standards zu setzen. Außerdem können auf diese Weise in den Projekten wiederverwendbare Komponenten identifiziert werden und zur Wiederverwendung und Einstellung in das Repository vorbereitet werden.

2.3 Ökonomische Faktoren

Da das Management eines Unternehmens i.d.R. nach betriebswirtschaftlichen Grundsätzen denkt und handelt, ist es von entscheidender Bedeutung, dass der Nutzen der Wiederverwendung anhand von konkreten Zahlen deutlich wird. Diese Ermittlung von Kennzahlen im Bereich der Software-Entwicklung wird in den meisten Unternehmen nicht durchgeführt, was dazu führt, dass das Management nicht bereit ist, dauerhaft Mittel für den Betrieb eines Repositories sowie der entsprechenden Stellen für ein „Reuse-Team“ zur Verfügung zu stellen [Reif1997]. Daher sollte eine Ermittlung der entsprechenden Kennzahlen ebenfalls zu den Aufgaben dieses Teams gehören.

2.4 Rechtliche Faktoren

Die teilweise unklare Rechtslage, beispielsweise bei der Fehlfunktion eines Produkts durch die Fehlfunktion einer extern eingekauften Komponente, führt teilweise dazu, dass bereits bestehende Komponenten nochmals implementiert werden (siehe [SoSo1998]), um den rechtlichen Risiken in diesem Bereich aus dem Wege zu gehen.

Auch die in [Same1997] erwähnte Möglichkeit, Komponenten zur Verwendung auf eigenes Risiko freizugeben, mag in manchen Bereichen eine geeignete Lösung sein, ist jedoch nicht überall anwendbar.

2.5 Psychologische Faktoren

Auch psychologische Faktoren stellen einen häufig unterschätzten Hinderungsgrund bei der Wiederverwendung von Software-Komponenten dar. Häufig werden Komponenten aus Konkurrenzdenken den Kollegen nicht zur Verfügung gestellt, um zu verhindern, dass diese Erfolge auf Basis der eigenen Arbeiten erreichen.

In vielen Fällen ist es auch der Forscher-Drang, der Entwickler dazu verleitet, Funktionalitäten zu implementieren, ohne ausreichend zu untersuchen, ob diese evtl. bereits an anderer Stelle entwickelt wurden [Reif1997].

2.6 Fachliche Faktoren

Der in der Einleitung beschriebene Mangel an wiederverwendbaren fachlichen Komponenten ist nicht zuletzt darauf zurückzuführen, dass es äußerst schwierig ist, fachliche Funktionalitäten derart zu verallgemeinern, dass sie als „Fachkomponenten“ wiederverwendbar werden.

Erfolgversprechend sind in diesem Bereich unternehmensübergreifende, branchenspezifische Initiativen, die vereinheitlichte fachliche Objektmodelle entwickeln und diese für Software-Hersteller und –Anwender in der Branche zur Verfügung stellen. Ein Vorreiter auf diesem Gebiet ist der Gesamtverband der deutschen Versicherungswirtschaft, der ein vereinheitlichtes Fachkonzept auf UML-Basis für die gesamte Versicherungsbranche definiert hat [GdV1999].

2.7 Wettbewerbsbedingte Faktoren

In vielen modernen Unternehmen, wie z.B. bei Online-Finanzdienstleistern, stellt die verwendete Software das wesentliche Produktionsmittel dar. Derartige Unternehmen sind daher aus Gründen des Wettbewerbs in vielen Fällen nicht dazu bereit, standardisierte Komponenten einzusetzen, da sie ansonsten Alleinstellungsmerkmale aufgeben müssen, die sie von ihren Konkurrenten unterscheiden. Für die Überwindung dieses Hemmnisses ist daher eine sehr hohe Flexibilität und Konfigurierbarkeit der Komponenten notwendig.

2.8 Strategische Faktoren

Neben wettbewerbsbedingten Faktoren spielen in vielen Fällen auch strategische Überlegungen eine Rolle. Beispielsweise wird sich ein Unternehmen bei wichtigen Komponenten nur sehr ungern auf Komponenten eines kleinen Herstellers verlassen, da dessen Zukunft und somit die Pflege und Weiterentwicklung dieser Komponenten evtl. nicht gesichert ist. Dies wurde insbesondere in den letzten Jahren deutlich, als viele Startup-Unternehmen, darunter auch Komponenten-Hersteller, ihre Geschäftstätigkeit einstellen mussten.

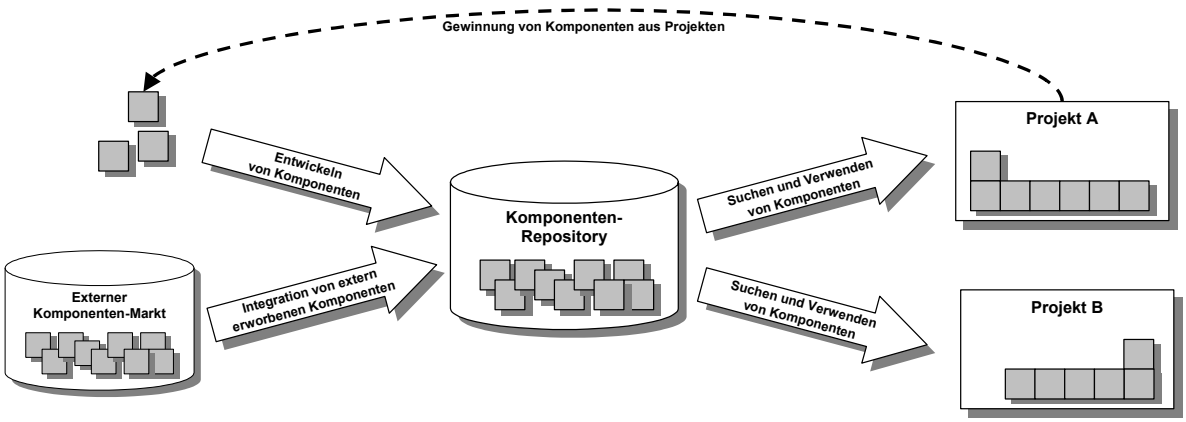


Bild 3: Hauptprozesse bei der komponentenbasierten Softwareentwicklung

3 Durch das Repository zu unterstützende Prozesse

Innerhalb der komponentenbasierten Softwareentwicklung existieren eine Reihe von Prozes-

sen, die durch ein Komponenten-Repository (vgl. Abschnitt 2.1) geeignet unterstützt werden müssen, um einen Nutzen zu erzielen. Die wichtigsten Teilprozesse, die auch eng mit den im Projekt REBOOT definierten „development *for* reuse“ und „development *with* reuse“ korrespondieren [Karl1995], sowie die Beteiligung des Reuse-Teams (vgl. Abschnitt 2.2) sind in Bild 3 dargestellt und werden in den folgenden Abschnitten näher erläutert (vgl. auch [Bull2000]). Die dargestellten Prozesse sind aus Gründen der Übersichtlichkeit stark vereinfacht und sollen nur die wesentlichen Aspekte verdeutlichen. Die Notation, die bei den graphischen Prozessdarstellungen verwendet wird, ist dabei an die Methode der ereignisgesteuerten Prozessketten angelehnt (siehe auch [KeNS1992]).

3.1 Entwicklung von neuen Komponenten bzw. Versionen

Neue Komponenten entstehen im Unternehmen i.d.R. auf zwei unterschiedliche Arten. Der einfache Fall ist, dass sie durch das Reuse-Team in Hinblick auf die spätere Wiederverwendung entwickelt werden. Die auf diese Art und Weise entstehenden Komponenten werden daher meistens die Anforderungen an eine spätere Wiederverwendung erfüllen.

Der weitaus häufigere Fall ist jedoch der, dass Software innerhalb von aktuellen Projekten entsteht und dabei von den Projektmitarbeiten oder durch das in beratender Funktion beteiligte Reuse-Team erkannt wird, dass sich die Software zur Wiederverwendung eignet. In diesem Fall ist jedoch i.d.R. ein weiterer Schritt für die Verallgemeinerung durch das Reuse-Team der meist an die konkreten Projektbelange angepasste Software notwendig, bevor man sie wirklich „Komponente“ nennen kann.

In beiden Fällen wird die Komponente anschließend durch das Reuse-Team in das Repository eingestellt. Dieser Teilprozess wird in Abschnitt 3.2 beschrieben.

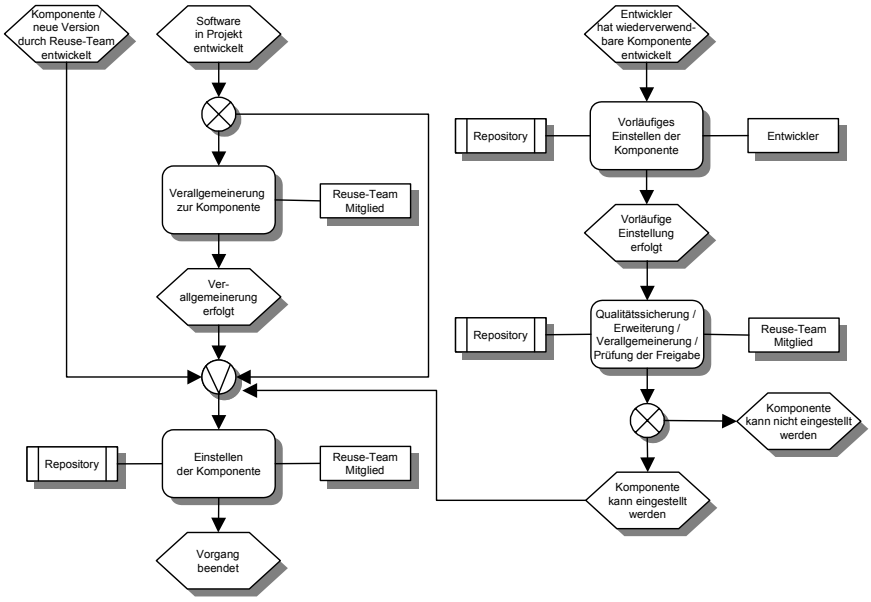


Bild 4: Entwicklung von neuen Komponenten bzw. Versionen

Es soll den Entwicklern auch selbst die Gelegenheit gegeben werden, Komponenten in das Repository einzustellen. Dies soll auf eine möglichst einfache Art und Weise möglich sein. Es

muss jedoch danach noch eine Qualitätssicherung mit einer Prüfung der Freigabe durch das Reuse-Team erfolgen. Nur wenn diese Freigabe erfolgt ist, kann die Komponente endgültig klassifiziert und eingestellt werden.

3.2 Einstellen von Komponenten bzw. Versionen

Vor der eigentlichen Einstellung einer Komponente in das Repository muss die dazugehörige Dokumentation bzw. Spezifikation ergänzt werden. Dabei bietet es sich an, sich an eine standardisierte Dokumentations- bzw. Spezifikationsmethodik anzulehnen. Ein Beispiel dafür ist der Vorschlag des GI-Arbeitskreises „Komponentenorientierte betriebliche Anwendungssysteme“ (siehe [ABC+2002]).

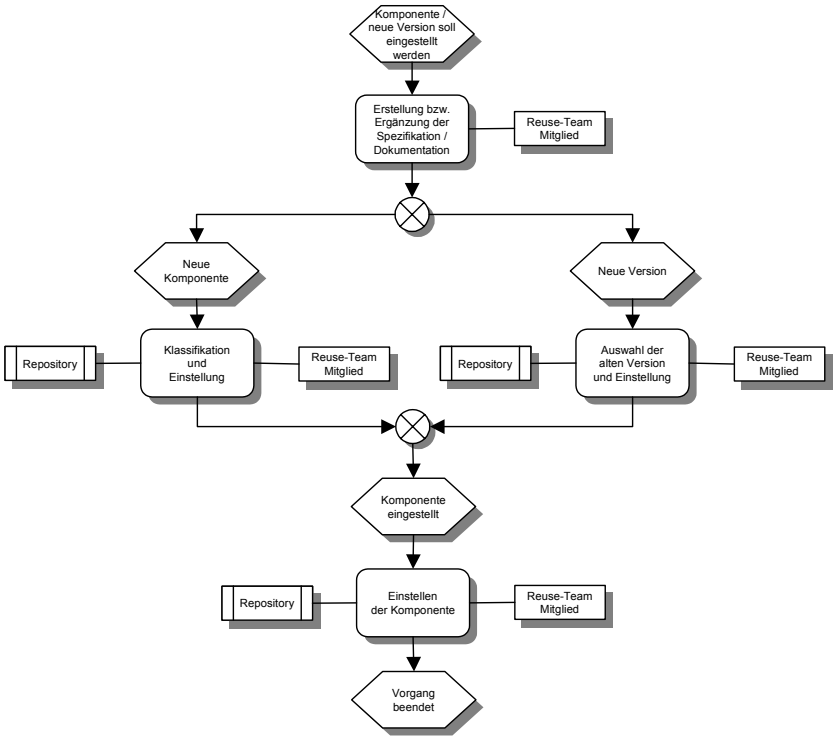


Bild 5: Einstellen von Komponenten bzw. Versionen

Wenn es sich um eine neue Version einer schon im Repository vorhandenen Komponente handelt, muss an dieser Stelle die vorherige Version der Komponente gesucht werden und die neue Version eingestellt werden.

Handelt es sich um eine neue, d.h. bisher nicht im Repository vorhandene Komponente, muss sie vor der Einstellung entsprechend klassifiziert werden, um später auch wieder auffindbar zu sein. In der Wissenschaft wurden schon eine Reihe von Ansätzen für die Klassifikation von Software-Komponenten entwickelt, wie z.B. die Klassifikation mit Facetten [Börs1995]. Derzeit besteht jedoch noch keine allgemein akzeptierte Klassifikationsmethodik für Komponenten. Ein Ansatz könnte sein, die Klassifikationsmethodik von bereits bestehenden Software-Märkten im Internet zu verwenden. Beispielsweise hat der Komponentenmarkt Flashline seine Klassifikationsschematik in Form einer XML-DTD veröffentlicht (siehe [Flas2002]).

Diese ist jedoch noch nicht sehr ausgereift und es besteht an dieser Stelle im Bereich der Definition eines vereinheitlichten Komponentenklassifikationsschemas definitiv noch Forschungs- bzw. Standardisierungsbedarf.

Ist die Komponente eingestellt, versendet das Repository in beiden Fällen automatisch Benachrichtigungen an die Anwender, die sich dafür registriert haben, bei neuen Versionen einer Komponente bzw. neuen Komponenten benachrichtigt zu werden (vgl. Abschnitt 3.4).

3.3 Integration von extern entwickelten Komponenten

Eine der Aufgaben des Reuse-Teams ist das regelmäßige Durchsuchen von externen Komponenten-Märkten und –Anbietern. Werden dort Komponenten gefunden, die für die Wiederverwendung geeignet sind, sollten diese nach einer Evaluationsphase in das Repository eingepflegt werden, d.h. es wird der COTS-Ansatz (Commercial Off-The-Shelf) unterstützt [Voas1998].

In vielen Fällen müssen jedoch vorher noch Anpassungen an die Spezifika des Unternehmens durchgeführt werden. Dabei ist darauf zu achten, dass diese Anpassungen bei Erscheinen einer neuen Version nicht verloren gehen, sondern übernommen werden können.

Der Optimalfall ist daher, dass keine Anpassungen notwendig sind. Dies ist jedoch bei komplexeren Komponenten eher selten der Fall.

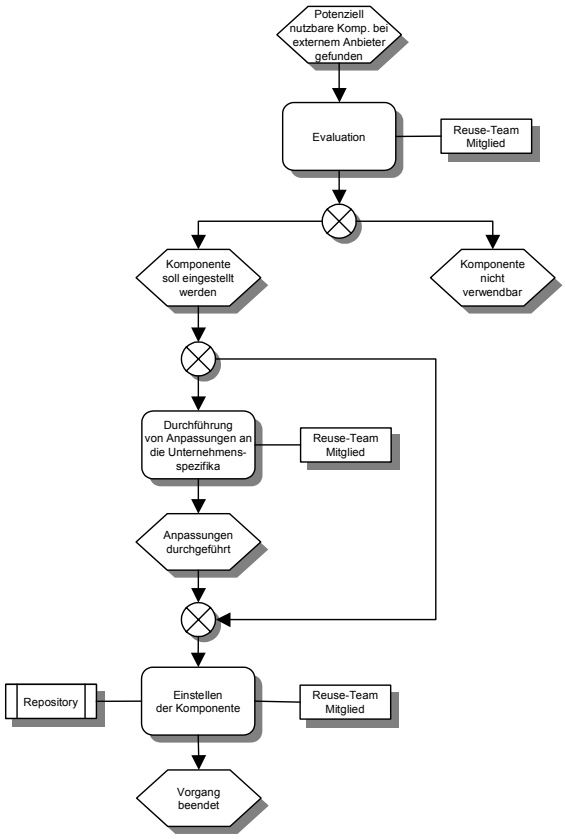


Bild 6: Integration von extern entwickelten Komponenten

3.4 Suchen und Verwenden von Komponenten

Der Verwendung einer Komponente können 2 unterschiedliche Ereignisse vorangehen:

- Eine Suche aufgrund des Bedarfs einer konkreten Komponente.
- Eine Benachrichtigung des Systems, dass eine neue Version einer Komponente existiert bzw. dass eine neue Komponente eingestellt wurde.

Im Falle einer Suche muss der Anwender die vorhandenen Komponenten auf Basis des verwendeten Klassifikationsschemas durchsuchen. Dabei sind unterschiedliche Suchmöglichkeiten, wie z.B. eine schlagwortbasierte Suche oder eine Baumsuche, denkbar (vgl. 4.1.2).

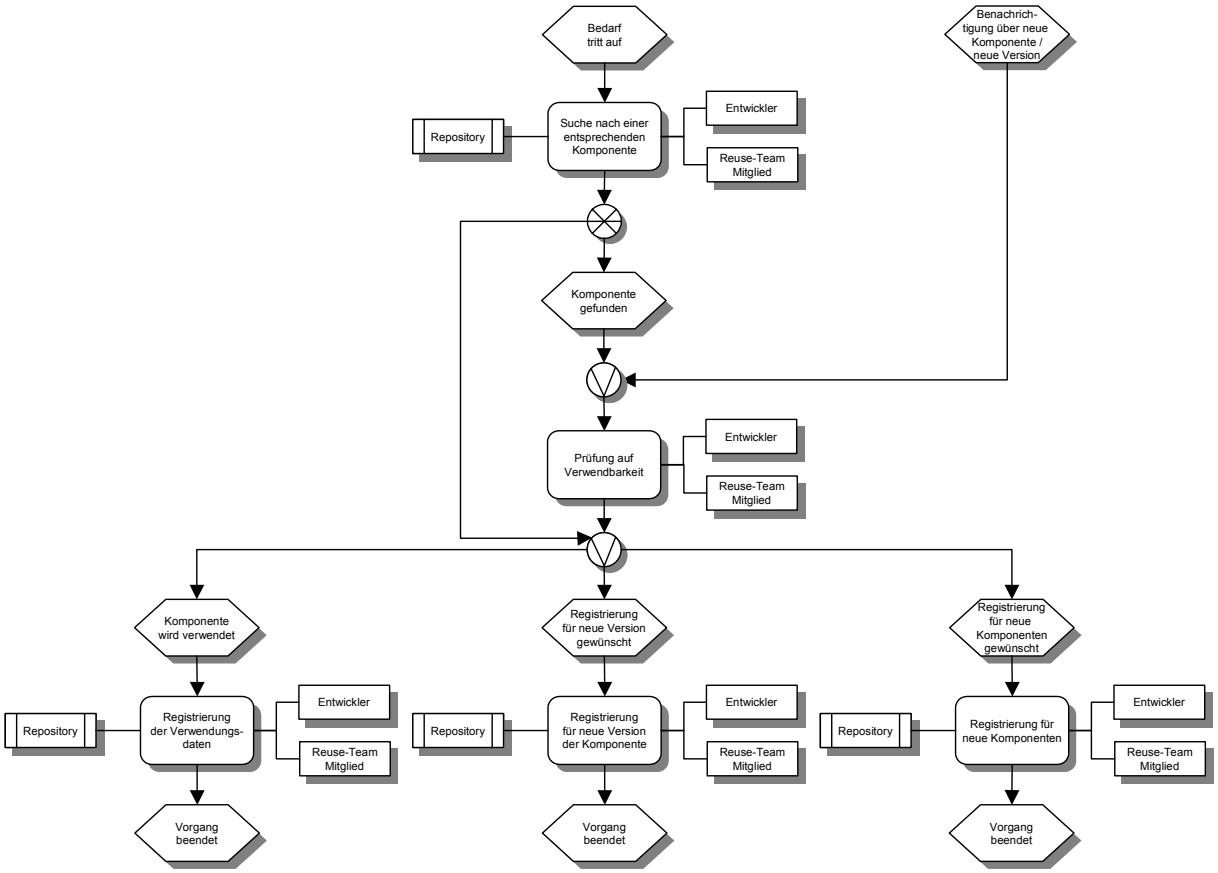


Bild 7: Entwicklung von neuen Komponenten bzw. Versionen

Wenn eine Komponente aufgrund der Klassifikationskriterien den Anforderungen entspricht, kann deren Dokumentation bzw. Spezifikation mit den Detailanforderungen abgeglichen werden. Dies wird umso einfacher, je mehr die Art der Dokumentation bzw. Spezifikation standardisiert ist, z.B. nach der Methode, die in [ABC+2002] beschrieben wird.

Wenn der Suchende eine Komponente findet, die seinen Anforderungen entspricht und er sich dafür entscheidet, sie zu verwenden, sollte das System automatisch die Daten der Verwendung (Adressdaten, Projektname, ...) erfassen, um eine Übersicht über die Verwendungen der Komponente zu erhalten und bei neuen Versionen eine Benachrichtigung generieren zu kön-

nen.

In jedem Fall, d.h. insbesondere auch im Fall einer nicht erfolgreichen Suche, kann sich der Benutzer für eine Benachrichtigung bei neuen Komponenten bzw. bei neuen Versionen von Komponenten registrieren. Dies ist nicht nur bei einzelnen Komponenten, sondern auch für ganze Gruppen von Komponenten, z.B. auf Grund des Klassifikationsschemas, möglich.

Bei der Wiederverwendung von bestehenden Komponenten ist insbesondere auch die Frage der Integration der Wiederverwendung in die Entwicklungsprozesse von großer Bedeutung, dabei insbesondere die Fragestellung, ob das Anwendungsdesign vor der Auswahl der geeigneter Komponenten erfolgt, oder ob das Anwendungsdesign abhängig von den vorhandenen Komponenten durchgeführt werden sollte (vgl. [PoBu2001]).

4 Benötigte Dienste eines Repositories

Um die in Abschnitt 3 beschriebenen Prozesse zu unterstützen, muss das Repository eine Reihe von Diensten bereitstellen. Die Dienste ergeben sich dabei hauptsächlich aus den funktionalen Anforderungen und können in Basis-Dienste und Mehrwert-Dienste unterteilt werden.

Basis-Dienste sind dabei Dienste, die Grundfunktionalitäten realisieren, die zur Funktion des Repositories unbedingt notwendig sind. Mehrwert-Dienste sind Dienste, die zwar nicht unbedingt notwendig sind, aber den Anwendern oder den Betreibern eines Repositories einen Mehrwert bringen.

Außerdem bestehen noch eine Reihe von technischen Anforderungen, die für den erfolgreichen Einsatz eines Repositories ausschlaggebend sind (siehe Bild 8).

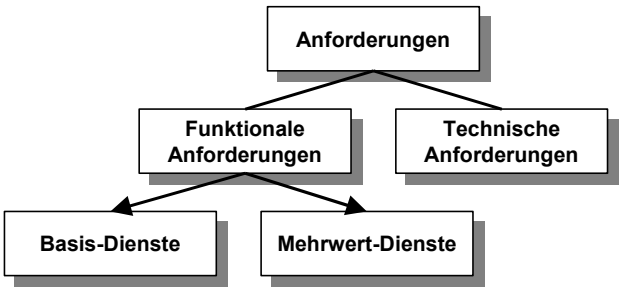


Bild 8: Anforderungen und Dienste

4.1 Basis-Dienste

Die Basis-Dienste beschränken sich im wesentlichen auf die Realisierung der Funktionalitäten für das Einstellen und Suchen von Komponenten sowie die Realisierung eines Benutzer- bzw. Rollenkonzepts.

4.1.1 Einstellen von Komponenten

Das Repository muss den Anwendern die Funktionalität anbieten, Komponenten jeglicher Art, d.h. beliebige Dateien, einzustellen. Im Falle von Komponenten, die z.B. aus lizenzrechtlichen Gründen nicht im Repository gespeichert werden können, muss ein Verweis auf die

Bezugsquelle eintragbar sein.

Die einzustellenden Komponenten müssen dabei mit entsprechenden Meta-Daten, wie z.B. Name, Kurzbeschreibung oder Autor versehen werden können. Ein besonders wichtiges Meta-Datum ist hierbei die Dokumentation bzw. Spezifikation der Komponente. Je nach Güte des Repositories kann diese nur als „Black Box“ oder in einer strukturierten und später auch auswertbaren Form abgelegt werden. In jedem Fall sollte eine standardisierte Spezifikations- und Dokumentationsmethodik verwandt werden, die obligatorisch bei allen abgelegten Komponenten eingehalten werden muss.

Desweiteren muss die Komponente in ein Klassifikationsschema eingeordnet werden. Hierbei sind auch domänenabhängige Klassifikationskriterien vorzusehen. Desweiteren können Komponenten nach unterschiedlichen Kriterien klassifiziert werden.

Um die Evolution der Komponenten zu unterstützen, muss eine Versionierungsfunktionalität vorhanden sein. Dabei können durchaus unterschiedliche Versionsnummern, z.B. interne und externe Versionsnummern, verwendet werden.

Um eine Qualitätssicherung zu unterstützen und um einen Wildwuchs zu vermeiden, sollten neu eingestellte Komponenten durch eine Clearing-Stelle, die durch das Reuse-Team gebildet wird, einer Prüfung unterzogen werden, bevor sie für die allgemeine Verwendung freigegeben werden.

4.1.2 Suchfunktionalitäten

Grundsätzlich sollten in einem Repository mindestens zwei Suchvarianten möglich sein:

- Eine schlagwortbasierte Suche.
- Eine baumbasierte Suche.

Die schlagwortbasierte Suche erlaubt nach Eingabe bzw. Auswahl von gewissen Suchkriterien eine Suche auf den Metadaten der Komponente. Die eingegebenen Kriterien werden durch die bekannten logischen Operatoren „UND“ und „ODER“ miteinander verknüpft. Bei der Formulierung bzw. Definition der Suchanfragen können beispielsweise auch Features wie z.B. eine Thesaurusunterstützung integriert werden (siehe auch [GCO+2000]). Die Suche wird solange verfeinert, bis die gewünschte Komponente gefunden ist bzw. die Suche aufgegeben wird.

Die baumbasierte Suche ermöglicht es dem Anwender, durch den Baum, der durch die Klassifikationsmerkmale aufgespannt wird, zu navigieren („browsen“) und sich auf jeder Ebene jeweils alle vorhandenen Komponenten anzeigen zu lassen. Dabei können dynamisch unterschiedliche Bäume aufgebaut werden, die je nach Art der Navigation unterschiedlich sind.

Die baumbasierte Suche ist dann besser geeignet, wenn der Anwender noch keine konkrete Vorstellung besitzt, durch die Eingabe welcher Schlagworte er die gesuchte Komponente finden kann, sondern er sich eher eine Übersicht über den Vorrat an Komponenten verschaffen möchte.

4.1.3 Unterstützung eines Benutzer- / Rollenkonzepts

Für den erfolgreichen Einsatz des Repositories ist es von großer Bedeutung, dass unterschiedliche Nutzergruppen unterstützt werden, die jeweils auch unterschiedliche Rechte besitzen.

Ein einfaches Benutzerkonzept kann mit folgenden vier Rollen realisiert werden:

- Gäste: Können nur lesend auf das Repository zugreifen, aber keine Komponenten einstellen.
- Angemeldete Nutzer: Können zusätzlich auch Komponenten einstellen.
- Reuse-Administratoren: Sind für die Pflege des Repositories sowie für die Freigabe von durch andere Nutzer eingestellten Komponenten zuständig.
- System-Administratoren: Besitzen alle Rechte und verwalten die Benutzer. Sie sind für den unterbrechungsfreien technischen Betrieb des Repositories zuständig.

Das verwendete Benutzer- / Rollenkonzept kann selbstverständlich noch beliebig detailliert werden. Es ist jedoch zu bedenken, dass auch ein entsprechender Pflegeaufwand einkalkuliert werden muss.

4.2 Mehrwertdienste

Zusätzlich zu den Basis-Diensten kann bzw. sollte ein Repository noch sog. Mehrwert-Dienste besitzen, die für den Anwender oder den Betreiber einen Mehrwert bieten. Im folgenden werden einige mögliche Dienste vorgestellt, der Kreativität sind an dieser Stelle jedoch kaum Grenzen gesetzt.

4.2.1 Benachrichtigungsmechanismen

Das Repository sollte eine Reihe von Benachrichtigungsdiensten anbieten. Alle Anwender, die eine bestimmte Komponente in ihrem Projekt nutzen, müssen sich im System registrieren, so dass sie automatisch benachrichtigt werden können, wenn eine neue Version dieser Komponente verfügbar wird (vgl. Abschnitt 3.4). Durch diese Verfahrensweise haben das Reuse-Team sowie andere Anwender auch einen Überblick, wer welche Komponenten in welchen Projekten nutzt.

Auch Benutzer, die die Komponente nicht nutzen, können sich für Benachrichtigungen bei neuen Versionen einer Komponente registrieren, da die neue Version evtl. den Anforderungen entspricht und dann genutzt werden kann.

Diese Registrierungsmöglichkeit kann auch auf ganze Kategorien von Komponenten ausgedehnt werden, so dass man sich z.B. dafür registrieren kann, benachrichtigt zu werden, wenn eine neue Komponente im Bereich „GUI-Komponenten“ eingestellt wird.

4.2.2 Einsatznachweise, Erfahrungsberichte und Bewertungen

Die Dokumentation der Wiederverwendung von aus dem Repository bezogenen Komponenten sollte obligatorisch sein, da diese Einsatznachweise eine wichtige Grundlage für die Berechnung von Kennzahlen (4.2.3) und die Implementierung von Anreizsystemen (4.2.4) darstellt.

Es sollte zusätzlich die Möglichkeit bestehen, dass die Anwender zu jeder Komponente Erfahrungsberichte und Bewertungen abgeben können. Auf diese Weise können Anwender von den Erfahrungen anderer Anwender profitieren.

Wichtig ist in diesem Zusammenhang ebenfalls eine Clearingstellen-Funktion der Reuse-Administratoren. Diese müssen neue Beiträge inhaltlich überprüfen und anschließend freige-

ben um eine hohe Qualität der Beiträge zu gewährleisten.

4.2.3 Unterstützung der Ermittlung und Auswertung von Kennzahlen

Um den ökonomischen Nutzen der Wiederverwendung von Software-Komponenten zu dokumentieren (oder im negativen Fall zu widerlegen) sollte das Repository eine Unterstützung für die Ermittlung und Auswertung von Kennzahlen, wie z.B. die Häufigkeit der Wiederverwendung einer Komponente, bereitstellen.

Somit soll das Reuse-Team befähigt werden, dem Management gegenüber Aussagen zum Kosten / Nutzen –Verhältnis von einzelnen Komponenten, zur Pflege und zum Betrieb des Repositories und somit zur ökonomischen Berechtigung des gesamten Reuse-Teams zu machen.

Eine besondere Bedeutung kommt dabei den in 4.2.2 beschriebenen Einsatznachweisen zu, da die Häufigkeit der Wiederverwendung einer Komponente für eine Reihe von Kennzahlen, z.B. aus dem Kosten / Nutzen-Bereich, eine wichtige Ausgangsgröße darstellt.

4.2.4 Implementierung von Anreizsystemen

Um die in Abschnitt 2.5 beschriebenen psychologischen Hemmnisse zu überwinden, ist es sinnvoll, Anreizsysteme in das Repository zu integrieren. Die Verwendung von Anreizsystemen ist ein häufig genutztes Mittel im Bereich des Wissensmanagements und kann daher auch für das Wissensmanagement im Bereich der Softwareentwicklung angewandt werden.

Der erste Ansatz dabei ist, den Einsteller einer Komponente zu belohnen. Diese Belohnung kann materieller, finanzieller oder auch ideeller Art sein. Der Nachteil bei dieser Vorgehensweise ist jedoch, dass die Belohnung auch dann erfolgt, wenn eine Wiederverwendung der eingestellten Komponente nicht erfolgt.

Daher ist ein anderer Ansatz der, dass die Belohnung erst dann wirksam wird, wenn die Komponente durch einen Dritten wiederverwendet wurde. Es besteht zusätzlich auch noch die Möglichkeit, statt dem Einsteller denjenigen zu belohnen, der eine Komponente wiederverwendet, um nicht nur die Einstellung sondern auch die Wiederverwendung und insbesondere auch die Dokumentation des Wiederverwendungseinsatzes zu belohnen. Auch in diesem Fall ist der Erfassung der Einsatznachweise eine wichtige Grundvoraussetzung (vgl. 4.2.2).

4.2.5 Portalfunktionalität

Das Repository sollte ebenfalls eine gewisse Portalfunktionalität anbieten, über die weitergehende Informationen unterschiedlichster Art abgerufen werden können. Beispiele hierfür sind:

- Eine strukturierte Linksammlung externer Komponentenmärkte und –anbieter.
- Ein Überblick über aktuelle Entwicklungsprojekte im Unternehmen.
- Eine Darstellung des Reuse-Teams (Personen und Aufgaben).
- Eine Übersicht über neu eingestellte Komponenten.

Dabei sollte eine Personalisierung der dargestellten Inhalte möglich sein, da dies ja auch eines der definierenden Merkmale eines Portals darstellt.

4.2.6 „Schwarzes Brett“

Da es in umfangreichen Datenbeständen nicht unbedingt garantiert ist, dass ein Anwender die gesuchte Komponente auch wirklich findet, ist es sinnvoll, eine Funktionalität anzubieten, die der eines „Schwarzen Bretts“ entspricht. Anwender können dort Suchanzeigen für Komponenten mit einer bestimmten Funktionalität aufgeben oder andere Fragen stellen.

Die Fragen können dann durch das Reuse-Team oder andere erfahrene Anwender beantwortet werden. Auf diese Weise können auch Anregungen für extern zu beschaffende Komponenten an das Reuse-Team herangetragen werden (vgl. Abschnitt 3.3).

4.3 Technische Anforderungen

Neben den in den Abschnitten 4.1 und 4.2 beschriebenen Basis- und Mehrwertdiensten existieren auch noch eine Reihe von technischen Anforderungen, die im folgenden beschrieben werden.

4.3.1 Verfügbarkeit über Intranet / Internet - Technologien

Da eine möglichst breite Verfügbarkeit gewährleistet werden soll, ist es notwendig, dass das Repository eine HTML-basierte Oberfläche besitzt. Somit können die Nutzer über ihre Standard-Webbrowser auf die Funktionalitäten des Repositories zugreifen, ohne zusätzliche Software installieren zu müssen.

Für die Administratoren ist zusätzlich ein Administrations-Client notwendig, da sich einige Funktionalitäten einer guten Benutzungsoberfläche (wie z.B. dynamische Baumdarstellungen) nur sehr unbefriedigend lösen lassen, dies aber für Administrations-Tätigkeiten oft notwendig ist.

4.3.2 Integration in die Entwicklungsumgebung

Um den Zugriff auf das Repository möglichst einfach zu gestalten, ist es notwenig, eine Integration des Repositories in die jeweiligen Entwicklungsumgebungen der Entwickler zu gewährleisten. Aufgrund der Vielfalt der Entwicklungsumgebungen und der fehlenden Standards in diesem Bereich ist eine Integration jedoch um so schwerer, je enger sie sein soll. Eine lose Integration, d.h. über einen Link, ist jedoch in den meisten Fällen zu realisieren.

4.3.3 Qualität der Benutzungsoberfläche

Wie bei jedem Wissensmanagement-System ist auf eine qualitativ hochwertige Gestaltung der Benutzungsoberfläche großen Wert zu legen, da die Motivation der Einsteller, also der Knackpunkt eines jeden Wissensmanagement-Systems, i.d.R. stark von der Qualität der Benutzungsoberfläche abhängt. Insbesondere ist in diesem Zusammenhang auch auf kurze Antwortzeiten bei der Einstellung, aber auch bei der Suche, zu achten.

4.3.4 Verfügbarkeit der Funktionalität über Web-Services

Um die Funktionalitäten des Repositories auch für andere Systeme verfügbar zu machen und um insbesondere die in Abschnitt 4.3.2 beschriebene Integration in die Entwicklungsumgebungen zu realisieren, ist es sinnvoll, die Funktionalitäten des Repositories auch über sog.

Web-Services anzubieten. Diese Web-Services nutzen das SOAP Protokoll, das derzeit durch das Word Wide Web Consortium standardisiert wird [W3C2002], um unterschiedliche Anwendungen auf eine plattform-unabhängige Art und Weise miteinander zu integrieren.

Unter Verwendung dieser Technologie könnten auch mehrere Repositories über ein Meta-Repository miteinander kombiniert werden oder es könnten die Daten von externen Komponenten-Märkten integriert werden, wenn diese ebenfalls eine derartige Schnittstelle bereitstellen. Diese Technologien spielen als Basis-Technologien für die Weiterentwicklung des Internet zu einem „Semantic Web“ eine immer größere Rolle.

5 Konzeption einer System-Architektur

Um die in diesem Beitrag beschriebenen Anforderungen und Dienste umzusetzen, muss eine System-Architektur gewählt werden, die die Funktionalität über unterschiedliche Wege den unterschiedlichen Nutzergruppen zur Verfügung stellt. Ein derartiges Komponenten-Repository sollte unter Verwendung von derzeit verfügbaren Komponenten-Technologien, wie z.B. die J2EE- oder die .NET-Technologien realisiert werden. In Bild 9 ist eine mögliche System-Architektur dargestellt.

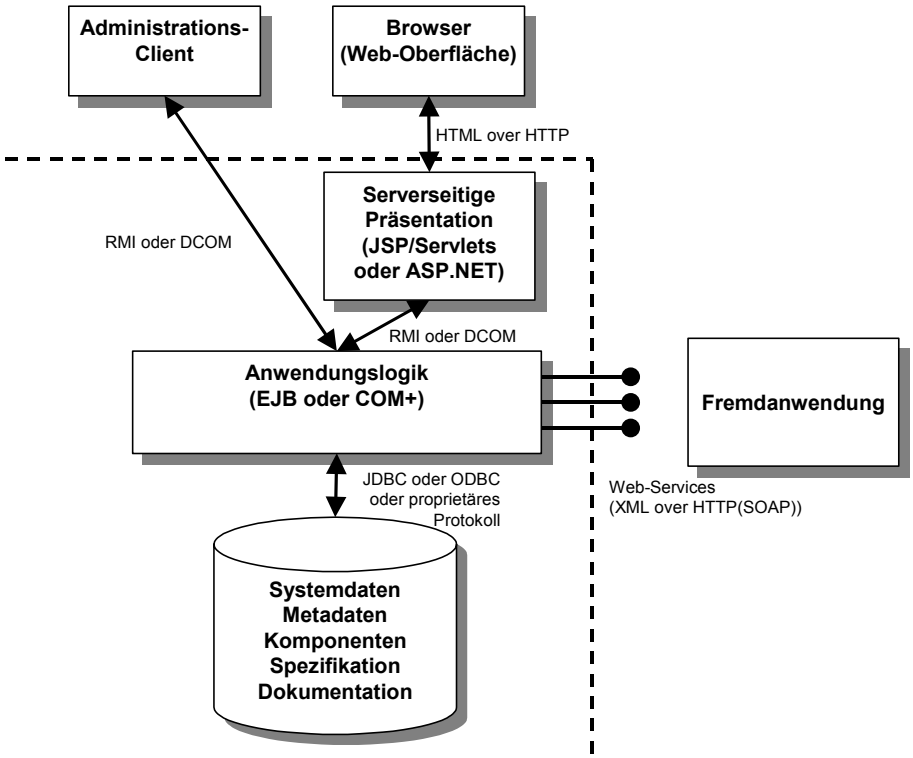


Bild 9: Mögliche System-Architektur eines Repositories

Für die Datenhaltung, d.h. für die Systemdaten, die Metadaten, die eigentlichen Komponenten sowie deren Spezifikation bzw. Dokumentation, können beispielsweise relationale Datenbank-Systeme eingesetzt werden. Falls die Dokumentation / Spezifikation XML-basiert ist, eignen sich in vielen Fällen auch XML-Datenbanken als Speichermedium.

Die Anwendungslogik sollte durch Komponenten der Technologien EJB oder COM+ realisiert werden. Die dort bereitgestellten Dienste können dann über drei unterschiedliche Kanäle zur Verfügung gestellt werden:

- Für die Endanwender werden die Daten durch eine serverseitige Präsentationsschicht in das HTML-Format aufbereitet, so dass sie überall verfügbar sind, wo ein entsprechender Browser vorhanden ist.
- Für Administratoren werden die Funktionalitäten über einen Administrations-Client zur Verfügung gestellt, da eine reine HTML-Oberfläche nicht den notwendigen Komfort für umfangreiche administrative Tätigkeiten bietet.
- Für die Integration in Fremdanwendungen, wie z.B. Entwicklungsumgebungen, werden die Dienste als Web-Services über das SOAP-Protokoll (Simple Object Access Protocol) zur Verfügung gestellt (siehe auch [W3C2002]).

Im Rahmen des durch das BMBF geförderten Projekts Adaptive READ wurde ein erster Prototyp eines Repositories in der beschriebenen Architektur erstellt. Dieser Prototyp wird derzeit insbesondere mit dem Fokus auf den Mehrwertdiensten weiterentwickelt.

6 Zusammenfassung und Ausblick

Im vorliegenden Beitrag wurden eine Reihe von Hemmnissen aufgezeigt, die die Wiederverwendung von Software-Komponenten behindern. Basierend auf den Hauptprozessen der komponentenbasierten Softwareentwicklung wurden die Anforderungen an ein Komponenten-Repository definiert, das einen Teil der angeführten Probleme lösen oder mildern kann. Kombiniert mit der Einführung eines Reuse-Teams, das für den Betrieb des Repositories verantwortlich ist, sowie von Anreiz- und Kennzahlensystemen können insbesondere Probleme im technischen, organisatorischen, psychologischen und ökonomischen Bereich adressiert werden.

Für den effektiven Einsatz eines derartigen Komponenten-Repositories ist es von enormer Bedeutung, einheitliche Klassifikations- und Spezifikationsmethoden für Software-Komponenten zu entwickeln. Die Klassifikationsmethoden sind dabei vor allem für das Einstellen und spätere Wiederfinden von Bedeutung, die Spezifikationsmethoden vor allem für die Evaluation und Anwendung von Komponenten.

Für einen Teil der angesprochenen Probleme, z.B. aus dem fachlichen, strategischen oder rechtlichen Bereich, kann ein Repository keine Lösung bieten. Teilweise sind diese Probleme in ihrem Grundsatz auch nur sehr schwer und auch in eher in einem anderen Kontext zu lösen.

Insgesamt kann zusammenfassend gesagt werden, dass der Bereich der Wiederverwendung von Software-Komponenten immer noch große Potenziale in sich verbirgt und noch enorme Anstrengungen in der Wissenschaft und in der anwendungsorientierten Forschung notwendig sind, um diese Potenziale nutzbar zu machen. Dies wird auch dadurch deutlich, dass im Bereich „Software Engineering“ des Arbeitsprogramms „IT 2006“ des BMBF (siehe [BMBF2002]) die Themen „Produktivitätserhöhung mittels Komponentenorientierung und Wiederverwendung“ sowie „Wissensmanagement in der Softwareentwicklung“ als Hauptthemen vertreten sind.

Literatur

[ABC+2002] *Ackermann, J.; Brinkop, F.; Conrad, S.; Fettke, P.; Frick, A.; Glistau, E.; Jaekel, H.; Klein, U.; Kotlar, O.; Loos, P.; Mrech, H.; Ortner, E.; Overhage, S.; Sahm, S.; Schmietendorf, A.; Teschke, T; Turowski, K.*: Vorschlag zur Vereinheitlichung der Spezifikation von Fachkomponenten. Memorandum des GI-Arbeitskreises 5.10.3: Komponentenorientierte betriebliche Anwendungssysteme, Version Februar 2002. http://wi2.wiso.uni-augsburg.de/gi-files/MEMO/Memorandum-Februar-2002_2.34.pdf, Abruf am 2002-02-28.

[Behl2000] *Behle, A.*: Wiederverwendung von Softwarekomponenten im Internet. Dissertation, Technische Hochschule Aachen, Dt. Univ.-Verlag, Wiesbaden 2000.

[BMBF2002] *Bundesministerium für Bildung und Forschung (BMBF)*: IT-Forschung 2006, Förderprogramm Informations- und Kommunikationstechnik. http://www.it2006.de/it-forschung_2006.pdf, Abruf am 2002-02-28.

[Börs1995] *Börstler, J.*: Feature-Oriented Classification for Software Reuse. In Proceedings SEKE 95, The 7th international conference on Software Engineering. Rockville, MD, USA, 22.-24.7.1995, S. 204-201.

[Bull2000] *Bullinger, H.-J. (Hrsg.)*: KoSPuD – Komponentenbasierte Software für Produkte und Dienstleistungen. Tagungsband, Fraunhofer IRB Verlag, ISBN 3-8167-5555-0, Stuttgart 2000.

[DiEs2001] *Dietzsch, A.; Esswein, W.:* Gibt es eine „Softwarekomponenten Industrie“ ? Ergebnisse einer empirischen Untersuchung. In: *Buhl, H. U.; Huther, A.; Reitwiesner, B. (Hrsg.):* Information Age Economy, 5. Internationale Tagung Wirtschaftsinformatik 2001, Tagungsband, 19.-21.9.2001, Augsburg, Physika-Verlag, ISBN 3-7908-1427-X , Heidelberg 2001, S. 697 – 710.

[Flas2002] *Flashline*: Component DTD, Beschreibung der Eigenschaften von Komponenten zur Einstellung in das Flashline-Archiv. http://www.componentregistry.com/dtd.jsp, Abruf am 2002-02-27.

[FWH+2000] *Fähnrich, K.-P.; Weisbecker, A.; Höß, O.; Kunsmann, J.:* Componentware – Vom Trend zum industriellen Einsatz. In: *Bullinger (Hrsg.)*: KoSPuD – Komponentenbasierte Software für Produkte und Dienstleistungen. Tagungsband, Fraunhofer IRB Verlag, ISBN 3-8167-5555-0, Stuttgart 2000, S. 1-37.

[GdV1999] *Gesamtverband der deutschen Versicherungswirtschaft e.V.:* Die Anwendungsarchitektur der Versicherungswirtschaft. Edition 1999. http://www.gdv-online.de/vaa/vaa_html/vaa3_0.htm, Abruf am 2002-02-27.

[GfK+2000] *GfK Marktforschung GmbH; Fraunhofer-Institut für Experimentelles Software Engineering (IESE); Fraunhofer-Institut für Systemtechnik und Innovationsforschung (ISI)*: Analyse und Evaluation der Softwareentwicklung in Deutschland. Abschlussbericht einer Studie für das Bundesministerium für Bildung und Forschung, Dezember 2000. http://www.dlr.de/IT/IV/Studien/evasoft_abschlussbericht.pdf, Abruf am 2002-02-25.

[GCO+2000] *Gibb, F.; McCartan, C.; O'Donnell, R.; Sweeney, N.; Leon, R.*: The integration of information retrieval techniques within a software reuse environment. In Journal of Information Science, Ausgabe 26 / 4, 2000.

[HöWe2001] *Höß, O.; Weisbecker, A.*: Komponentenbasierte Software für Produkte und Dienstleistungen (KoSPuD): Ergebnisse und Erfahrungen eines praxisorientierten Verbundforschungsprojekts. In: *Turowski, K. (Hrsg.)*: 3. Workshop komponentenorientierte betriebliche Anwendungssysteme (WKBA 3), Universität der Bundeswehr München und Gesellschaft für Informatik, Tagungsband, Frankfurt 2001, S. 1-13.

[JaGJ1997] *Jacobson, I.; Griss, M.; Johnson, P.*: Software Reuse – Architecture, Process And Organization For Business Success. Addison-Wesley Longman, New York 1997.

[Karl1995] *Karlsson, E.-A.*: Software Reuse – A Holistic Approach. Wiley, Chichester 1995.

[KeNS1992] *Keller, G.; Nüttgens, M.; Scheer, A.-W.*: Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". Veröffentlichung des Instituts für Wirtschaftsinformatik Heft 89, Saarbrücken 1992.

[PoBu2001] *Pohthong, A; Budgen, D.*: Reuse Strategies in software development: an empirical study. In: Information and Software Technology, 43 (2001), S. 561-575. 2001.

[Reif1997] *Reifer, D. J.*: Practical Software Reuse. Wiley, New York 1997.

[Same1997] *Sametinger, J.*: Software Engineering with Reusable Components. Springer, Berlin 1997.

[SoSo1998] *Sodhi, J.; Sodhi, P.*: Software Reuse – Domain Analysis and Design Processes. McGraw-Hill, New York 1998.

[Voas1998] *Voas, J. M.*: The Challenges of Using COTS Software in Component-Based Development. In IEEE Computer, June 1998, S. 53-59.

[WAD+2000] *Weisbecker, A.; Appl, J.; Drawehn, J.; Höß, O.; Schuster, E.*: Stand der Technik. In: *Weisbecker, A. (Hrsg.)*: KoSPuD – Komponentenbasierte Software für Produkte und Dienstleistungen. Abschlussbericht, Fraunhofer IRB Verlag, ISBN 3-8167-5556-9, Stuttgart 2000, S. 13-61.

[W3C2002] *World Wide Web Consortium (W3C):* Web Services Activity. http://www.w3.org/2002/ws/, Abruf am 2002-02-28.

[WeGr1998] *Weisbecker, A.; Groh, G. (Hrsg.)*: PROMPT – Organisationsgestaltung und Methoden für menschengerechte Software-Entwicklungsprozesse. Abschlussbericht, Fraunhofer IRB Verlag, ISBN 3-8167-5176-8, Stuttgart 1998

[WeHS2001] *Weisbecker, A.; Höß, O.; Strauß, O.*: Organisatorische und technische Maßnahmen zur Wiederverwendung von Komponenten. Deliverable D2.7, Projekt Adaptive READ, gefördert durch das BMBF, Stuttgart 2001.

Ein Modell zur Ermittlung der Reife des Softwaremarktes

Heiko Hahn

Universität der Bundeswehr München, Fakultät für Informatik, Institut für angewandte Systemforschung und Operations Research (IASFOR), Werner-Heisenberg-Weg 39, D-85577 Neubiberg, Tel: (89) 6004-3391, FAX:-3036, E-mail: heiko.hahn@informatik.unibw-muenchen.de

Zusammenfassung: Obwohl die Idee, Softwaresysteme aus vorproduzierten Softwarekomponenten (kompositorisch) zusammenzustellen schon lange in der Literatur diskutiert wird, haben sich Märkte für Softwarekomponenten noch nicht etablieren können. Märkte sind in einen institutionellen Rahmen eingebunden, denen Standards zugrunde liegen und diese sind wiederum Ausdruck der Reife eines Marktes. In dem vorliegenden Beitrag wird ein Modell zur Messung der Reife des Softwaremarktes vorgestellt, dass als Basis für eine spätere Operationalisierung und empirische Überprüfung dienen soll. Das Modell wird theoriegeleitet auf der Basis der Transaktionskostenökonomie, der Informationsökonomie, der Netzwerkökonomie und des Resource-based-view entwickelt.

Schlüsselworte: Komponenten, Komponentenmärkte, Transaktionskostentheorie, Informationsökonomik, Resource based view

1 Einleitung

Die Entwicklung komponentenbasierter Softwaresysteme beschäftigt die Software-Technik schon seit den späten 60er Jahre des vergangenen Jahrhunderts. McIllroy hatte 1968 bereits seine Vision von Softwaresystemen formuliert, die aus vorgefertigten Komponenten zusammengesetzt werden sollten [Mcil1976].

Die Vorteile solch komponentenbasierter Systeme sind nicht von der Hand zu weisen. Zunächst trägt der Ansatz der inhärenten Komplexität heutiger Softwaresysteme Rechnung. Durch die Bereitstellung genau spezifizierter Schnittstellen sind die Komponenten nur lose gekoppelt und die Interdependenzen zwischen den verschiedenen Komponenten bzw. Systembestandteilen sind auf die expliziten Schnittstellen reduziert [Ritt2000].

Zum einen ist der Austausch einzelner Systembausteine, die eine bestimmte Funktionalität bereitstellen, einfacher möglich. Der Kunde hat damit die Wahl zwischen alternativen Komponenten, die prinzipiell gleiche oder ähnliche Funktionalität in unterschiedlicher Güte und zu unterschiedlichen Preisen bereitstellen. Dadurch ist ein Wettbewerb zwischen den Herstellern der Basis funktionaler und nichtfunktionaler Kriterien auch auf Komponentenebene möglich.

Zum anderen kann der Kunde neue, komplementäre Funktionalität einfach und kostengünstig integrieren bzw. entfernen, um so den Umfang bereitgestellter Funktionalität (und damit auch die Komplexität des Systems) an seine Bedürfnisse anzupassen (vgl. [BaCl2000], insbesondere S. 123 ff. über die Besonderheiten und Charakteristika modularen Designs). Die einfache Integration neuer Funktionalität ist besonders aufgrund der häufig auftretenden komplementären Beziehung der von den einzelnen Systembausteinen angebotenen Funktionalität wichtig.

Bei hoher Komplementarität stellt sich nur unter Nutzung des Gesamtsystems der mit den hohen Investitionen verbundene Nutzen ein (vgl. [MiRo1995] zum Begriff der Komplementarität von Systembausteinen).

Neben solchen -sich aus der modularen Architektur ergebenden- Vorteile verspricht man sich von komponentenbasierten Anwendungssystemen durch die Wiederverwendung von bereits vorher entwickelten und getesteten Komponenten geringere Kosten, bessere Qualität und eine Reduktion des time-to-market [Behl2000, S.2]. Ein denkbares Szenario wäre die Verwendung von Standardplattformen oder Komponentenframeworks, die die für eine gewisse Anwendungsdomain generische Funktionalität bereitstellen würden. Letztlich wäre nur an den Stellen spezielle Software zu entwickeln, wo benötigte Funktionalität durch keine vorhandene Komponente bereitgestellt werden könnte [Schm2001]. Obwohl ähnliche Konzepte wie z.B. Plattformstrategien [Meye1997] etwa in der Automobilindustrie bereits genutzt werden, haben sich bisher keine Märkte für Softwarekomponenten etablieren können [DiEs2001].

Märkte sind in ein institutionelles Geflecht eingebunden. Dieses lässt die Beziehungen der Markteilnehmer sowie die Leistungen (bzw. Eigentumsrechte i.S. v. property rights), die zwischen den Teilnehmer ausgetauscht werden, effizient bestimmen. Das Wissen um die relevanten Informationen ist nicht zuletzt Ausdruck der Reife des Marktes.

Ziel des Beitrages ist die Formulierung eines Modells auf der Basis von Determinanten, welches die Reife von Märkten empirisch bestimmbar macht. Die grundlegende Annahme ist, dass die Etablierung eines Marktes für Softwarekomponenten nur dann möglich ist, wenn die zugrundeliegenden Transaktionsbeziehungen zwischen den Bereitstellern und Nachfragern von Funktionalität über eine hinreichende Reife verfügen.

2 Der Markt in der neoklassischen Theorie

Betrachtet man die neoklassische Mikroökonomik, so wird der Markt ebenso wie die Unternehmung (in Form der Produktionsfunktion) als black-box betrachtet. Der Markt ist - etwa nach WALRAS - ein abstrakter Ort, auf dem *commodities* friktionslos getauscht werden [Walr1954 S. 84]. WILLIAMSON hat in seinen Arbeiten zur Transaktionskostentheorie unter Bezugnahme auf MACNEIL die Charakteristika der zugrundeliegenden Verträge beschrieben. Die Transaktionspartner sind bei den als klassisch bezeichneten Verträgen vollständig informiert und es bestehen keine Präferenzen hinsichtlich einzelner Partner, so dass vollständige Anonymität zwischen den Beteiligten möglich ist. Es entstehen weder Kosten ex-ante für die Informationsbeschaffung über die Marktpartner, über die zugrundliegenden bzw. möglichen Eigentumsübertragungsmöglichkeiten oder Absicherungsmechanismen noch für die eigentliche Vertragsverhandlung oder die ex-post Anpassung und Durchsetzung der Verträge. Es herrscht letztlich vollständige Information und vollständige Sicherheit, und es fallen keine Transaktionskosten an. Der Preis ist der alleinige Allokationsmechanismus [Nort1991, S. 30]. Es werden alle möglichen und auch verifizierbaren Umweltzustände von dem Vertrag vollständig abgedeckt, der Wert der auszutauschenden Eigentumsrechte (inklusive der Sanktionsmechanismen) ist vollständig spezifizierbar und die Transaktionen können völlig diskret abgewickelt werden [Will1985, S. 69]. Es entstehen keinen längerfristigen Bindungen oder Präferenzen in bezug auf zukünftige Transaktionen aus voran gegangenen.

Ausgehend von dieser Beschreibung lassen sich drei Charakteristika reifer Märkte bestimmen (Bild 1): Diskretheit (zeitliche Abgeschlossenheit) der Marktransaktionen, Anonymität der Marktteilnehmer und (inhaltliche) Vollständigkeit der Verträge. Da die neoklassische Mikro-

ökonomik den Markt als black-box betrachtet, geht sie auch nicht der Frage nach, wie es zu den Bedingungen kommt, dass die Akteure derart agieren können und wie sich die zugrundeliegenden formellen und informellen Regeln herausbilden. Sie ist daher nur bedingt geeignet, reale Transaktionsprozesse und den Rahmen, in dem sie eingebettet sind, theoretisch zu durchdringen [Fisc1993, S. 30 ff]. Daher gilt es vielmehr zunächst entsprechende Theorien zu bestimmen.

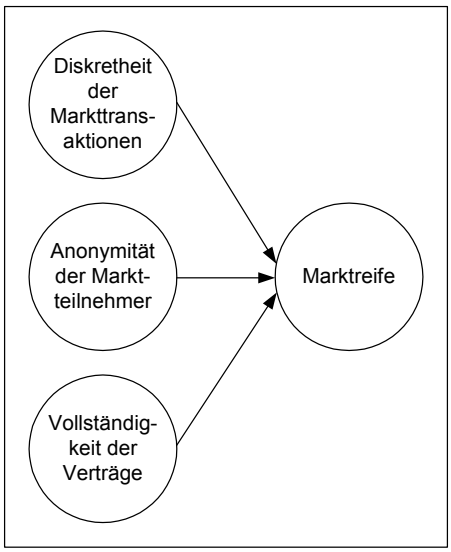


Bild 1: Charakteristika reifer Märkte

3 Theoretische Basis des Modells

Das zu entwickelnde Modell soll die Determinanten aufzeigen, welche die Auswahl des zugrundeliegenden Koordinationsmechanismus bestimmen. Reale Transaktionen finden nicht nur über Märkte statt, sondern sind in eine Vielzahl unterschiedlicher Transaktionsbeziehungen (z.B. Unternehmen, oder Kooperation/Netzwerk) eingebunden. Der Markt ist daher auch nicht immer die dominante Koordinationsform [Simo1991].

Die theoretische Basis für die weitere Diskussion bildet zunächst die Neue Institutionenökonomik. Sie unterscheidet sich von der neoklassischen Mikroökonomik dadurch, dass sie den institutionellen Rahmen, in dem die Transaktionen stattfinden, genauer untersucht. Institutionen sind die grundlegenden Mechanismen zur Reduktion der Transaktionskosten. Sie bestehen aus den formalen Regeln (z. B. Gesetzen, Verfassungen), informalen Regeln (z. B. Verhaltensnormen) und den entsprechenden Durchsetzungs- und Sanktionsmechanismen. Institutionelle Regelungen bestimmen das Anreizsystem von Gesellschaften und insbesondere ökonomischer Systeme [Nort1994, S. 360].

Die Neue Institutionenökonomik selbst ist keine homogene Theorie, sondern ein Forschungsprogramm, das aus der Auseinandersetzung mit den Schwächen der neoklassischen Theorie entstanden ist. Zu ihr gehören u.a. die Transaktionskosten-, die Property-rights- und die Principal-Agent Theorie, für einen Überblick über die verschiedenen und nicht immer leicht abgrenzbaren Theoriezweige s. [RiFu1999].

Weitere theoretische Basis bildet die mit der Neuen Institutionenökonomik eng verwandte

Informationsökonomik (z. B. [Stig1961], [Aker1970], [Aker1970]). Sie setzt sich u.a. mit den Problemen der Qualitätsbeurteilung von Gütern und dem möglichen Marktversagen bei zu großer Qualitätsunsicherheit [Aker1970] auseinander.

Aufgrund der Charakteristika von Softwarekomponenten als digitale Güter spielt die der (neoklassischen) Industrieökonomik zurechenbare und wohlfahrtsökonomisch geprägte Theorie der Netzwerkexternalitäten (für einen Überblick s. [Econ1996], grundlegend [KaSh1985] bzw. [FaSa1986]) eine weitere Rolle. Der Netzwerkeffekt ergibt sich aus der Notwendigkeit der Kompatibilität, um einzelne Komponenten zu einem Gesamtsystem kompositorisch zusammenstellen zu können. Abhängigkeit von Systemgütern, denen proprietäre Standards zugrunde liegen führen, zu einem lock-in Effekt.

Neben diesen volkswirtschaftlich geprägten Theorien bildet der Resource based view einen weiteren theoretischen Pfeiler. Der Resource based view entstammt dem Strategischen Management und setzt sich mit der Frage auseinander, worauf die Wettbewerbsvorteile einzelner Unternehmen beruhen (für einen Überblick s. [RaWo1994], grundlegend u.a. [Gran1991], [Barn1991]). Zudem betrachtet er die Unvollkommenheiten von Faktormärkten, bestimmte Ressourcen bereitzustellen und zu übertragen.

4 Die Vollständigkeit und Anonymität von Verträgen

Im vorherigen Abschnitt wurden drei Kriterien für reife Märkte formuliert. Zunächst sind Verträge vollständig spezifizierbar, die Transaktionspartner treten anonym auf und die einzelnen Transaktionen finden jeweils diskret statt, d.h. sie sind zeitlich eindeutig abgrenzbar und es entstehen keine weiteren Verflechtungen. Diese Annahmen sollen im folgenden diskutiert werden. Zunächst ist der Frage nachzugehen, inwieweit Verträge für Softwarekomponenten in der Tat vollständig spezifiziert werden können. Das Kriterium der vollständigen Spezifisierbarkeit soll anhand zweier Dimensionen analysiert werden. Zunächst steht die Frage im Mittelpunkt, inwieweit Unternehmen in der Lage sind, die Qualität der Leistungen vor dem Kauf beurteilen zu können (Leistungsbewertung). Allgemein zeichnen sich Verträge, die vollständig spezifiziert sind, dadurch aus, dass die zugrundeliegenden Property rights und Sanktionsmechanismen vollständig bestimmt sind. [FuPe1972].

Eine notwendige Bedingung hierzu ist, dass das Transaktionsobjekt vor der Transaktion umfassend beurteilt werden kann. Mit Transaktionsobjekt ist zudem die gesamte Leistung inklusive u.U. notwendiger zusätzlicher Dienstleistungen gemeint, damit sichergestellt werden kann, dass das Transaktionsobjekt (Softwarekomponente, die eine bestimmte Funktionalität bereitstellen soll) auch den von dem Kunden erwarteten Nutzen erzielt.

Die Informationsökonomie unterscheidet dabei drei Arten von Gütern. Zunächst solche Güter, deren Qualitäten ohne weiteres vor dem Kauf beurteilt werden können, sogenannte Suchgüter. Als zweite Kategorie führt sie solche Güter auf, deren Eigenschaften erst nach dem Gebrauch beurteilt werden können, sogenannte Erfahrungsgüter. Neben dieser von NELSON [Nels1970] gemachten Unterscheidung wurde von DARBY/KARNI der Begriff des Vertrauensguts eingeführt. Es handelt sich dabei um Güter, bei denen der Kunde selbst nach dem Kauf die Eigenschaften nicht ohne größeren Kostenaufwand (selbstständig) beurteilen kann [DaKa1973].

Grundsätzlich ist davon auszugehen, dass bei der Beurteilung unterschiedlicher Leistungen alle drei Kategorien in jeweils unterschiedlicher Gewichtung relevant sind [Adle1996], letztlich erlauben aber nur Suchgüter die Leistungsbeurteilung vor dem Kauf. Wendet man diese Kategorien auf die Bewertung von Komponenten an, so wäre theoretisch eine Beurteilung der

formalen Spezifikation naheliegend. Diese dürfte aber in der Regel nicht vorliegen bzw. die vollständige Spezifizierung technisch nicht möglich sein. Als weiterhin problematisch könnte sich die Frage erweisen, wer das Systemverhalten prüft und garantiert, wenn es etwa darum geht, eine weitere Komponente von einem dritten Anbieter in das System zu integrieren. Es könnte so zu nicht eindeutig spezifizierten Garantieleistungen kommen, die zu negativen Anreizen hinsichtlich Komponentenqualität führen können. Eine kompositorische Wiederverwendung hat daher Haftung für das Gesamtsystem zu klären. Auch die hohen Kosten der Implementierung (z. B. Parametrisierung und Anpassung), die heute noch mit der Einführung neuer Funktionalität verbunden sind, lassen sich nicht ohne weiteres vor dem Kauf bestimmen. Entsprechend ist auch die Bedeutung von Referenzinstallationen zu relativieren, wenn die Systeme nicht vergleichbar sind.

Neben der eigentlichen Qualitätsbeurteilung kann die Unternehmung aber u.U. auch bei der Bestimmung des eigentlichen Leistungsbedarfs (Leistungsspezifiszierung) überfordert sein, so dass sie auf Beratung bei der Auswahl und Beurteilung unterschiedlicher Software angewiesen ist. Eine Überprüfung der erbrachten Leistung setzt voraus, dass die Unternehmen diese vorher genau hat spezifizieren können. Erfolgt diese Spezifizierung erst während der Leistungserstellung in Interaktion mit dem Anbieter, so umfasst die Leistung einen u.U. schwer zu beurteilenden integralen Dienstleistungsanteil. Dieser ist von den einzelnen Anbietern nicht vollständig kontrollierbar, aber auch von den Nachfragern nicht ex-ante beurteilbar. (vgl. z.B. die Argumentation von [EnFr1995], insbesondere 38 ff.).

Die Einbindung von Beratungsunternehmen ist insofern von Interesse, weil es zu klären gilt, ob die von ihnen angebotene Leistung auf der Basis objektivierbarer Daten erfolgt (etwa vergleichbar allgemeiner Ingenieursberatung/TÜV) oder ob die Leistung den Charakter eines Vertrauensgut hat, also die Unternehmung aufgrund mangelnder Expertise die Qualität des Auswahlprozesses im nachhinein nicht vollständig beurteilen kann. Es könnte sich bspw. für ein Unternehmen als schwierig erweisen zu bestimmen, ob das von der Beratung empfohlene System wirklich optimal war oder inwieweit der Aufwand, den das Beratungsunternehmen erbracht hat, wirklich dem angesetzten Umfang entsprach. Im Falle von technischen Problemen bei der Integration, erhöhter Kosten bei der Einführung des Systems oder mangelnder Rentabilität der Investition (z.B. ließen sich erhoffte Kostenpotentiale nicht realisieren) kann es ebenso schwierig sein, die genauen Ursachen und Verantwortung zu klären. Dies betrifft nicht zuletzt auch das oben angesprochene Problem, dass der Kunde bei der Leistungserstellung integriert werden muss und die Leistung dadurch auch nicht voll von dem Anbieter kontrolliert und unabhängig bereit gestellt werden kann.

Ein Mittel zur Unterstützung der Leistungsbewertung bzw. zur Reduktion der Qualitätsunsicherheit auf anonymen Märkten sind Garantien. Dies betrifft v.a. Märkte, auf denen es schwierig ist, zwischen Angeboten unterschiedlicher Qualität zu diskriminieren. Diese unterliegen prinzipiell der Gefahr des Marktversagens. Dieser Aspekt wurde besonders von [Aker1970] problematisiert. Sofern es keine Möglichkeit gibt, zwischen Angeboten unterschiedlicher Qualität zu diskriminieren, besteht die Gefahr, dass es nicht genug Anreize gibt, Produkte mit entsprechender Qualität anzubieten, da sie keinen höheren bzw. angemessenen Preis erzielen können. Garantien gelten daher als Signale für überlegene Qualität, sofern sie auch einklagbar sind bzw. mit hohen Kosten für den Anbieter verbunden sind, der schlechte Qualität bereithält. Auch Investitionen in Zertifizierungsmaßnahmen können als solche Signale interpretiert werden, sofern sie für das Unternehmen mit entsprechenden Kosten verbunden sind.

In Abwesenheit entsprechender Garantien kommt der Reputation im Sinne eines Vertrauens in das Verhalten des Transaktionspartners große Bedeutung zu. Vertrauen in den Transaktionspartner unterscheidet sich aber grundsätzlich von Standards. Im Gegensatz zu Standards setzt Vertrauen und Reputation eine Identifikation des anderen Transaktionspartners voraus und widerspricht somit der Annahme der Anonymität. Im Gegensatz zu allgemeinen Standards ist Vertrauen in der Regel an ein bestimmtes Unternehmen oder rechtlich-organisatorische Einheit gebunden. In sehr innovativen Märkten, auf denen sich noch keine festen Standards etabliert haben, kommt der Reputation große Bedeutung in ihrer Funktion als Informationssubstitut zu [Adle1996]. Die zugrundeliegende Überlegung ist, dass Unternehmen, die in den Aufbau von Reputation investiert haben und daher höhere Preise fordern können, nicht das Risiko eingehen werden, durch das Angebot minderwertiger Leistungen dieses Kapital zu gefährden (so genannte fly-by-night Strategie). Die Fähigkeit einer Unternehmung, mit besserer Reputation einen höheren Preis zu verlangen, stellt daher einen Anreiz dar, bessere Qualität anzubieten [Shap1983]. Williamson spricht in diesem Zusammenhang auch von ´credible commitments`, die dem Kunden quasi eine Geisel (hostage) überlassen [Will1983]. Für ein Unternehmen stellt die Lieferung schlechter Qualität oder sonstiges Ruf schädigendes Verhalten also die Gefährdung der Investitionen da, die es in den Aufbau seiner Reputation getätigt hat. Hier sollen unter credible comittment also Investitionen verstanden werden, die als kostspielige Signale von den Kunden aufgefasst werden können und deren Amortisierung mit dem (dauerhaften) marktlichen Erfolg der Unternehmung verbunden ist. Reputation kommt transaktionskostensenkende Funktion zu, da sie als Signal Such-, Auswahl- und v.a. Bewertungskosten für die andere Marktseite reduziert. Auf der anderen Seite wiederspricht die an ein bestimmtes Unternehmen gebundene Reputation aber der Bedingung anonymer Marktpartner. Eine rein marktliche Transaktionsbeziehung würde Vertrauen in die Effektivität von Standards bedingen und keine weitere Absicherung durch solche Signale erfordern.

Neben der Bewertung der Leistung spielt das Problem der autonomen Leistungsspezifizierung, also der inhaltlichen Bestimmung von Leistungsangeboten, eine wichtige Rolle. Ohne eigenständige Kompetenz bei der Leistungsspezifierung gegenüber den Komponentenherstellern bzw. einem Intermediär ist auch kein autonomes Agieren möglich. Entsprechend groß ist in diesem Fall die Abhängigkeit von externer Kompetenz.

Betrachtet man dies im weiteren Kontext komponentenbasierter Anwendungssysteme, so besteht hier die Notwendigkeit zur umfassenden Standardisierung, die neben der Auswahl technischer Standards auch eine Standardisierung auf fachlicher Ebene erforderlich machen würde ([Turo2000]). Dies setzt auch das Wissen über die Bedürfnisse auf fachlicher Ebene und Fähigkeit zur Bewertung voraus, die Beteiligung der Kunden bei der Standardisierung ist unabdingbar für den Erfolg.

Schließlich gibt es noch einen zusätzlichen Aspekt, der die Annahme der Anonymität der Marktteilnehmer verletzt. Dies betrifft den Fall, dass aufgrund überlegender Kompetenz einzelner Anbieter der Markt nicht in der Lage ist, ein entsprechendes Angebot unter Wahrung einer Konkurrenzsituation bereitzustellen.

5 Die Diskretheit von Verträgen

Besondere Beachtung bei der Auswahl der verschiedenen Transaktionsmechanismen kommt darüber hinaus der Berücksichtigung der Spezifität einer Investition zu. Sie spielt eine zentrale Rolle in der Transaktionskostenökonomik [RiWi1985]. Die Höhe der Spezifität einer Investi-

tion ergibt sich aus dem Wert, den eine Investition bei einer alternativen Verwendung erzielen kann. [KlCA1978, S. 298]. Der potentielle Verlust bei einer solchen alternativen Verwendung wird auch als Quasi-Rente der Investition bezeichnet. Spezifische Investitionen führen zu einer sogenannten lock-in Situation, da die Investitionskosten durch den Verkauf oder alternative Nutzung nicht wieder hereingeholt werden können. Aufgrund der zugrundeliegenden Annahmen eines opportunistischen Verhaltens sind entsprechende Absicherungsmechanismen notwendig. Spezifische Investitionen sind daher auch nicht diskret, da ex-post die Gefahr des Ausnutzens der Situation durch die mächtigere Seite besteht. [Will1985] spricht hier von fundamentaler Transformation, da eine Situation der ex-ante Konkurrenz nicht zu einer Situation der ex-post Konkurrenz führt.

Betrachtet man komponentenbasierte Softwaresysteme, so ist bspw. die Entscheidung für ein bestimmtes Komponentenmodell spezifisch, da nicht beliebige Komponenten miteinander verbunden werden können. Die Investitionen in ein bestimmtes System sind i.d.R. an den entsprechenden Standard gebunden und zukünftig zu erwerbende Komponenten müssen diesen einhalten. Besondere Bedeutung kommen zudem Netzwerkeffekten zu. Netzwerkeffekte beschreiben das Phänomen, dass der Wert eines Gutes nicht nur von seinen inhärenten Eigenschaften abhängt, sondern zum Teil auch von der Anzahl der anderen Nutzer des Netzwerkgutes. Es lassen sich direkte und indirekte Netzwerkeffekte unterscheiden. Direkte Netzwerkeffekte richten sich nach der Anzahl der Nutzer, mit denen direkt kommuniziert werden kann. Der Wert indirekter Netzwerkeffekte bestimmt sich aus der Anzahl komplementärer Produkte und Dienstleistungen, die für einen bestimmten Standard angeboten werden. [KaSh1985].

Im Falle proprietärer Standards, bei denen ein Wechsel mit hohen Kosten und Verlust des Netzwerknutzens verbunden wäre, besteht die Gefahr, dass der Inhaber des Standards etwa die Kosten für ein Update künstlich erhöht und so einen Teil der Quasi-Rente für sich abschöpft. Zur Garantie vor opportunistischen Verhalten sind offene Standards besonders wichtig, da so alternative Anbieter und entsprechende Konkurrenz gewährleistet bleibt [FaGa1988]. Auch hier ist zu erwarten, dass sowohl die Kompetenz des Anbieters als auch Vertrauen in den Anbieter des Standards wichtig sind, sofern eine Auswahl zwischen verschiedenen Anbietern überhaupt besteht (Gefahr einer ´the-winner-takes-it-all` Situation bei hohen Netzwerkeffekten).

Die Kompetenz des standardsetzenden Unternehmens bzw. der standardsetzenden Unternehmen kann zudem bei der Etablierung eines Standards wichtig sein, da die Gefahr besteht, dass man sich auf einen Standard festlegt, der später scheitert und es zur Gefährdung der Investitionen kommt. Entsprechend große Bedeutung kommt der Erwartung der Marktteilnehmer zu, die sich verständlicherweise Investitionssicherheit wünschen.

6 Herleitung des Modells

Die rein marktliche Koordination ist an Voraussetzungen gebunden, auf die weiter oben eingegangen wurden. Zudem sind alternative Koordinationsmechanismen in der Realität zu finden, die u.U. kooperative Beziehungen erwarten lassen, etwa für den Fall, dass Reputation eine große Rolle spielt oder langfristige Bindungen eingegangen werden. Eine Sonderrolle nimmt dabei die Situation des lock-ins ein, wenn Unternehmen aufgrund von Netzwerkexternalitäten an einen (Monopol-) Anbieter gebunden sind. Im Gegensatz zur Kooperation bestimmt hier nicht Vertrauen und Kompetenz, sondern v.a. die Bindung aufgrund spezifischer Investitionen die Transaktionsbeziehung. Reputation in Form von Kompetenz ist aber nicht

ausgeschlossen, da der Anbieter aufgrund technischen Wandels seine Position langfristig nicht gefährden möchte.

Bild 2 gibt einen Überblick über das Modell, dass zur Ermittlung der Marktreife dient. Die oben beschriebenen Koordinationsmechanismen sind darin als eine Funktion der Marktreife zu verstehen. Auf der linken Seite erfolgt zudem eine Zuteilung der theoretischen Basis zu den Modellelementen. Eine zentrale Rolle spielen hierin die Souveränität/Autonomie des Käufers und die Machtposition des Verkäufers. Reputation und spezifische Investitionen begründen die Machtposition des Verkäufers. Dagegen wird die Souveränität/Autonomie durch die Fähigkeit des Käufers zur selbstständigen Leistungsbeurteilung und Spezifikation bestimmt. Beide bestimmen die Marktreife. Die zentralen Hypothesen lauten:

H.1: Je großer die Machtposition des Verkäufers, desto geringer die Marktreife

H.2: Je größer die Autonomie des Käufers, desto höher die Marktreife

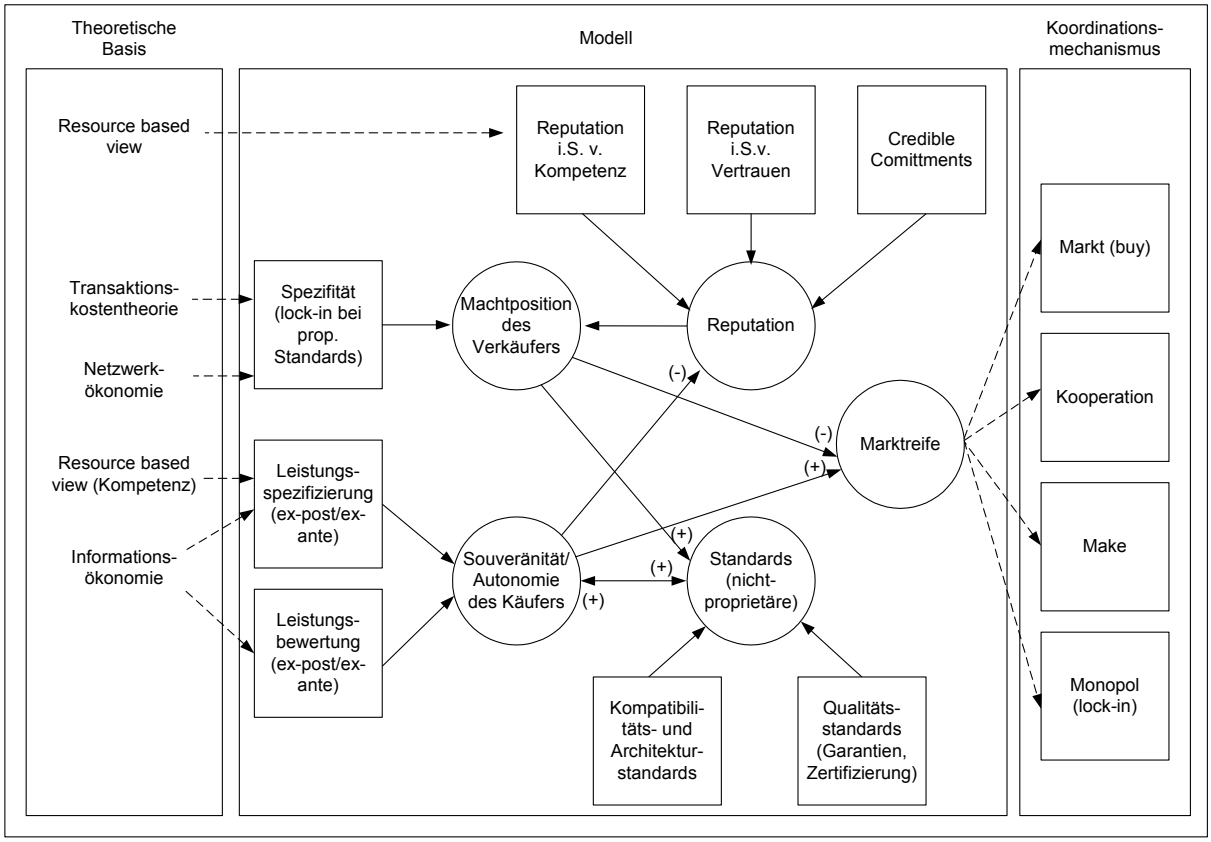


Bild 2: Modellüberblick

Darüber hinaus enthält das Modell weitere („kausale“) Annahmen.

H.3.: Je größer die Reputation einzelner Anbieter, desto größer ist ihre Bedeutung bei der Etablierung von Standards

H.3 ruht nicht zuletzt in der obigen Annahme, dass die Übernahme von Standards mit großen Unsicherheiten seitens Käufer und Verkäufer verbunden ist. Somit kommt dem Handeln marktmächtiger Unternehmen hohe Bedeutung zu, die aufgrund Ihrer Kompetenz bzw. Marktmacht bedeutenden Einfluss bei der Etablierung haben.

Der Einfluss der Autonomie/Souveranität des Käufers wird dagegen als wechselseitig angenommen, d.h.

> H.4: Je größer die Bedeutung von Standards, desto größer die Souveränität des Käufers
>
> H.5: Je größer die Souveränität/ Autonomie des Käufers, desto größer sein Einfluss auf die Gestaltung von Standards.

Mit H.5 kommt zum Ausdruck, dass auch auf der Seite des Kunden ausreichendes Wissen zur Standardetablierung vorhanden sein muss. Ein weiterer Aspekt, auf den hier nicht weiter eingegangen wird, ist die Tatsache, dass die Setzung von Standards natürlich mit Kosten verbunden ist. Somit sind – nicht zuletzt auch aufgrund des öffentlichen-Gut-Charakters von Standards – Anreizaspekte zu berücksichtigen. Unternehmen sollte zudem bewusst sein, dass Investitionen in die eigene Kompetenz bei der Beurteilung Einfluss auf die Qualität des Angebotes hat, da sie zur Reduktion von Informationsasymmetrien beitragen können und so die Gefahr opportunistischen Verhaltens durch die Gegenseite reduziert.

Schließlich wird auch davon ausgegangen, dass die Autonomie/Souveranität im gegensätzlichen Verhältnis zur Bedeutung der Reputation des Verkäufers steht, also

> H.6: Je größer die Autonomie/Souveränität, desto geringer die Bedeutung der Reputation

Bei genauren Hinsehen ist hier jedoch die Einschränkung zu machen, dass sich Reputation nur auf Vertrauen und nicht Kompetenz bezieht, sofern die Macht des Verkäufers auf der überlegenden Kompetenz seines Angebotes beruht.

7 Zusammenfassung

Ziel des Beitrages war die Herausarbeitung eines Modells, dass die Messung der Reife des Softwaremarktes erlaubt. Die zugrundeliegende Annahme lautet, dass die Etablierung eines Marktes für Softwarekomponenten nicht unabhängig von der allgemeinen Reife des Softwaremarktes bzw. der Beziehungen der einzelnen Transaktionspartnern und ihren Möglichkeiten zur Gestaltung und Überprüfung der zugrundeliegenden Verträge ist.

Als Charakteristika reifer Märkte wurden die Diskretheit (zeitliche Abgeschlossenheit) der Marktransaktionen, die Anonymität der Marktteilnehmer und die (inhaltliche) Vollständigkeit der Verträge bestimmt und dann ein Modell entwickelt, dass die realen Beziehungen empirisch messbar macht. Zwei institutionelle Aspekte sollen hier noch einmal in ihrem unterschiedlichen Charakter betont werden: Zum einen Standards, die marktliche Koordination repräsentieren und zum anderen Reputation. Zwei Unterschiede sind bei den Standards zu machen: zum einen Kompatibilitäts- und Architekturstandards, die u.U. auch eine inhaltliche Bestimmung der Leistung erlauben und zum anderen Qualitätsstandards, die die Qualität einer Leistung beurteilbar machen.

Die Bedeutung von Reputation wird dahingehend gedeutet, dass hier die Unzulänglichkeiten von Standards zum Ausdruck kommen. Es ist daher als Ausdruck auch der mangelnden Reife und Abhängigkeit von einzelnen Anbietern zu sehen. Eine Ausnahme betrifft den Fall, dass letztlich die überlegende Kompetenz eines einzelnen Anbieters als Grund für die Abhängigkeit aufgeführt werden kann, da dies auch Ausdruck des Wettbewerbs zwischen verschiedenen Anbietern um überlegende Lösungen ist.

Als nächster Schritt ist die Operationalisierung und empirische Überprüfung des Modells notwendig.

Literatur

[Adle1996] *Adler, J.*: Informationsökonomische Fundierung von Austauschprozessen: eine nachfrageorientierte Analyse. Gabler, Wiesbaden 1996.

[Aker1970] *Akerlof, G. A.*: The Market for "Lemons": Quality uncertainty and the Market Mechanism. In: The Quarterly Journal of Economics 84 (1970) 3, S. 488-500.

[BaCl2000] *Baldwin, C. Y.; Clark, K.*: Design Rules:. The MIT Press, Cambridge (Mass.) London 2000.

[Barn1991] *Barney, J.*: Firm Resources and Sustained Competitive Advantage. In: Journal of Management 17 (1991) 1, S. 99-120.

[Behl2000] *Behle, A.*: Wiederverwendung von Softwarekomponenten. Deutscher Universitäts-Verlag, Wiesbaden 2000.

[DaKa1973] *Darby, M. R.; Karni, E.*: Free Competition and the Optimal Amount of Fraud. In: The Journal of Law and Economics 16 (1973), S. 68-88.

[DiEs2001] *Dietzsch, A.; Esswein, W.*: Gibt es eine"Softwarekomponenten Industrie"? Ergebnisse einer empirischen Untersuchung. In: *H. U. Buhl; A. Huther; B. Reitwiesner (Hrsg.)*: Information Age Economy: 5. Internationale Fachtagung Wirtschaftsinformatik 2001. Physika-Verlag, Heidelberg 2001.

[Econ1996] *Economides, N.*: The Economics of networks. In: International Journal of Industrial Organization 14 (1996) 6.

[EnFr1995] *Engelhardt, W. H.; Freiling, J.*: Integrativität als Brücke zwischen Einzeltransaktion und Geschäftsbeziehung. In: Marketing ZfP 17 (1995) 1, S. 37-43.

[FaGa1988] *Farrell, J.; Gallini, N.*: Second-Sourcing as a Commitment: Monopoly Incentives to Attract Competition. In: Quarterly Journal of Economics 103 (1988) 4, S. 673-694.

[FaSa1986] *Farrell, J.; Saloner, G.*: Installed Base and Compatibility: Innovation, Product Preannoucements and Predation. In: American Economic Review 76 (1986) 5, S. 940-955.

[Fisc1993] *Fischer, M.*: Make-or-Buy-Entscheidungen im Marketing: Neue Institutionenlehre und Distributionspolitik. Gabler, Wiesbaden 1993.

[FuPe1972] *Furubotn, E. G.; Pejovich, S.*: Property Rights and Economic Theory: A Survey of Recent Literature. In: Journal of Economic Literature 10 (1972) 4, S. 1137-1162.

[Gran1991] *Grant, R. M.*: The Resource-Based Theory of Competitive Advantage: Implications for Strategy Formulation. In: California Management Review 33 (1991) 3, S. 114-135.

[KaSh1985] *Katz, M. L.; Shapiro, C.*: Network externalities, competition, and compatibility. In: American Economic Review 75 (1985) 3, S. 424-440.

[KlCA1978] *Klein, B.; Crawford, R. G.; Alchian, A. A.*: Vertical Integration, appropriable Rents, and the Competitive Contracting Process. In: Journal of Law and Economics 21 (1978) 2, S. 297-326.

[Mcil1976] *McIlroy, M. D.*: Mass produced software components. In: *P. Naur; B. Randell (Hrsg.)*: Software engineering. Petrocelli Charter, New York 1976, S. 88-95.

[Meye1997] *Meyer, M. H.; Lehnerd, A. P.*: The Power of Product Platforms: Building Value and Cost Leadership. Free Press, New York 1997.

[MiRo1995] *Milgrom, P.; Roberts, J.*: Complementarities and fit -Strategy, structure, and organizational change in manufacturing. In: Journal of Accounting and Economics 19 (1995), S. 179-208.

[Nels1970] *Nelson, P.*: Information and Consumer Behavior. In: The Journal of Political Economy 78 (1970) 2, S. 311-329.

[Nort1991] *North, D. C.*: Institutions, institutional change and economic performance. Cambridge University Press, Cambridge [u.a.] 1991.

[Nort1994] *North, D. C.*: Economic Performance Through Time. In: The American Economic Review 86 (1994) 3, S. 359-368.

[RaWo1994] *Rasche, C.; Wolfrum, B.*: Ressourcenorientierte Unternehmensführung. In: Die Betriebswirtschaft 54 (1994) 4, S. 501-517.

[RiFu1999] *Richter, R.; Furubotn, E. G.*: Neue Institutionenökonomik. Bd. 2, Mohr Siebeck, Tübingen 1999.

[RiWi1985] *Riordan, M. H.; Williamson, O. E.*: Asset Specifity and Economic Organization. In: International Journal of Industrial Organization 3 (1985), S. 365-.

[Ritt2000] *Ritter, J.*: Prozessorientierte Konfiguration komponentenbasierter AnwendungssystemeUniversität Oldenburg 2000.

[Schm2001] *Schmitzer, B.*: Beiträge zur Verwendung der Framework-Technologie bei der Entwicklung und Einführung von Systemen der betrieblichen Informationsverarbeitung. dissertation.de, Berlin 2001.

[Shap1983] *Shapiro, C.*: Premiums for High Quality Products as Returns on Reputation. In: Quarterly Journal of Economics 98 (1983) 4, S. 659-680.

[Simo1991] *Simon, H. A.*: Organizations and Markets. In: Journal of Economic Perspectives 5 (1991) 2, S. 25-44.

[Stig1961] *Stigler, G.*: The Economics of Information. In: The Journal of Political Economy 69 (1961) 3, S. 213-225.

[Turo2000] *Turowski, K.*: Establishing Standards for Business Components. In: *K. Jakobs (Hrsg.)*: Information Technology Standards and Standardisation: A Global Perspective. Idea Group Publishing, 2000, S. 131-151.

[Walr1954] *Walras, L.*: Elements of pure economics or the theory of social wealth. Allen & Unwin, London 1954.

[Will1983] *Williamson, O. E.*: Credible Comittment: Using Hostages to Support Exchange. In: American Economic Review 73 (1983) 4, S. 519-540.

[Will1985] *Williamson, O. E.*: The economic institutions of capitalism. The Free Press, New York [u.a.] 1985.

Herausgeber:

Prof. Dr. Klaus Turowski
Lehrstuhl für Betriebswirtschaftslehre, insbesondere Wirtschaftsinformatik II
Universität Augsburg
Universitätsstraße 16, 86135 Augsburg
Phone: +49(821)598-4431; Fax : -4432
E-Mail: klaus.turowski@wiwi.uni-augsburg.de
URL: http://wi2.wiwi.uni-augsburg.de

ISSN 1619-9014