Methoden- und Werkzeugunterstützung für evolutionäre, objektorientierte Software-Projekte

Dissertation
zur
Erlangen des Doktorgrades
der Naturwissenschaften
(Dr. rer. nat.)

dem

Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
vorgelegt von Siar Sarferaz

Marburg/Lahn, 2003

Vom Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg als Dissertation am
angenommen.

Erstgutachter Prof. Dr. Wolfgang Hesse
Zweitgutachter Prof. Dr. Bernhard Seeger

Tag der mündlichen Prüfung am 16.07.2003

Inhaltsverzeichnis

1 Einleitung

In den letzten Jahren wurden verstärkt objektorientierte Entwicklungsmethoden mit entsprechenden Vorgehensmodellen vorgeschlagen. Zu den bekanntesten Vorgehensmodellen für die objektorientierte Software-Entwicklung gehört z.B. *OOA&D* von P. Coad und E. Yourdon, *OOSA* von S. Shlaer und S. Mellor, *OMT* von J. Rumbaugh, *OOSE* von I. Jacobson oder *OOAD* von G. Booch. Die drei zuletzt genannten Autoren vereinten ihre Werke zu einem gemeinsamen Prozeßmodell, das 1999 von der Firma „Rational Software“ als *Rational Unified Process (RUP)* veröffentlicht wurde. Der RUP ist inzwischen sehr weit verbreitet und wird vermehrt in neuen Projekten eingesetzt. Dies liegt u.a. an bewährten Konzepten, wie z.B. iterative und inkrementelle Entwicklung, die im RUP verwendet werden.

Der RUP weist jedoch einige grundsätzliche Mängel auf. Zum Beispiel steht er mit seinen Phasen und Meilensteinen in der Tradition des Wasserfall-Modells und genügt daher nur bedingt den Anforderungen der objekt- und komponenten-orientierten Entwicklungs-methoden. Zwar wird betont, daß der RUP architektur-zentriert sei, jedoch wird auf die Architektur zu wenig Bezug genommen. Bewährte Konzepte, wie z.B. hierarchische Strukturen oder Rekursion, die zur Beherrschung der Komplexität heutiger Projekte nötig sind, werden in RUP nicht verwendet. Die Software-Entwicklung vollzieht sich im Zusammenspiel mehrerer parallel verlaufender Prozesse. Zu diesen Prozessen gehört u.a. auch die Qualitätssicherung und die Nutzer-Partizipation. Diese Prozesse werden aber in RUP nicht ausreichend berücksichtigt. Die Wiederverwendung wird zwar in RUP angepriesen, findet jedoch kaum Beachtung. Erste Erfahrungsberichte zeigen, daß die Anpassung von RUP an die Bedürfnisse eines Projekts sich als eine komplizierte und zeitaufwendige Tätigkeit herausgestellt hat. Dies wird u.a. mit der unscharfen Beziehung zwischen Phasen, Prozessen und Iterationen begründet. So wie bei den meisten Vorgehensmodellen wird auch beim RUP der evolutionäre Charakter der Software-Entwicklung unzureichend berücksichtigt. Die Software-Entwicklung verläuft nämlich in der Regel als evolutionärer Prozeß, d.h. als eine Folge von Erweiterungs- und Anpassungszyklen, beruhend auf Nutzung und Revisionen.

Das von W. Hesse vorgeschlagene Prozeßmodell für die evolutionäre, objektorientierte Software-Entwicklung (EOS) begegnet den oben genannten Defiziten von RUP. Der Kerngedanke bei diesem Prozeßmodell ist die Verbindung der evolutionären System-Entwicklung mit den Prinzipien der Objektorientierung. Die evolutionäre, objektorientierte Software-Entwicklung orientiert sich an Leitlinien wie *Hierarchischer Systemaufbau*, *Zyklische Entwicklung*, *Objektorientierung als durchgängige Entwicklungsmethodik*, *Einbezug von Erprobung und Nutzung* und *Weiter- und Wiederverwendung von Software-Bausteinen*. Das EOS-Modell ist methodenunabhängig und gliedert sich in die Subprozesse *Software-Entwicklung*, *Projektmanagement*, *Qualitätssicherung*, *Konfigurationsmanagement* und *Nutzung und Bewertung*.

Das Ziel der Arbeit ist es, das abstrakt gehaltene EOS-Modell für den praktischen Einsatz in Software-Projekten zu konkretisieren. Dazu müssen zwei Aspekte berücksichtigt werden. Die Projektmitglieder müssen wissen, wie das EOS-Modell anzuwenden ist. Daher müssen für die fünf Subprozesse konkrete Methoden vorgeschlagen werden (*Methoden-Unterstützung*). Für den erfolgreichen Projekteinsatz solcher Methoden ist aber eine systematische Unterstützung durch Werkzeuge unumgänglich. Daher muß auch geklärt werden, wie ein Werkzeug gebaut sein muß, damit es die vorgeschlagenen Methoden durchgängig unterstützt (*Werkzeug-Unterstützung*).

Das Ziel „Methoden-Unterstützung“ stellt eine schwierige Herausforderung dar. Denn zur Definition von Methoden müssen die einzelnen Teilgebiete der Softwaretechnik mit EOS-spezifischen Methoden abgedeckt werden. Die einzelnen Teilgebiete stellen aber separate Forschungsgebiete dar, die jeweils erkundet und hinsichtlich neuer Methoden analysiert werden müssen. Bei der Definition von Methoden wird zunächst eine Prozeßmodellierung für die fünf Subprozesse des EOS-Modells vorgenommen. Hierbei wird geklärt, wer (*Rolle*) tut was (*Aktivität*), wann und mit welchem Ergebnis (*Produkt*). Der Subprozeß Projekt-management wird in die Teilschritte *Projekt-Initialisierung und Planung*, *Projekt-Steuerung* und *Projekt-Abschluß* gegliedert. Die Software-Entwicklung wird entsprechend der Entwicklungsebenen *System*, *Komponente/Subsystem* und *Klasse* in drei Teilprozesse zerlegt. Der Subprozeß Qualitätssicherung wird in die Abschnitte *Prüfplanung*, *Prüfvorbereitung* und *Prüfdurchführung* eingeteilt. Das Konfigurationsmanagement wird in die Teilprozesse *Strukturierung der Bausteinbibliothek* und *Fortschreibung der Bausteinbibliothek* zerlegt. Der Subprozeß Nutzung und Bewertung wird in einen *Planungs*-, *Durchführungs*- und *Abschluß-Schritt* gegliedert. Die definierten Methoden für solche Teilschritte werden in einem Prozeß-Handbuch zusammengefaßt. Das 200 Seiten lange Prozeß-Handbuch befindet sich im Anhang A und kann als solches Projektmitgliedern zur Verfügung gestellt werden. Damit sichergestellt werden konnte, daß die im Prozeß-Handbuch definierten Methoden praktisch relevant sind, wurden sie mit Experten aus der Software-Industrie, wie z.B. der Siemens AG, intensiv diskutiert und gegebenenfalls modifiziert.

Das Ziel „Werkzeug-Unterstützung“ stellt ebenfalls eine große Herausforderung dar. Denn eine schriftliche Dokumentation der Methoden allein reicht für den Projekteinsatz nicht aus. Für den erfolgreichen Einsatz von Methoden ist eine systematische Unterstützung durch Werkzeuge notwendig. Wie müßte aber ein Werkzeug gebaut sein, das alle Subprozesse des EOS-Modells mit den jeweiligen Methoden durchgängig unterstützt? Die Antwort auf diese Frage wird in Form eines Prototyps für ein Prozeßmanagement-System gegeben. Der Prototyp integriert die fünf Subprozesse des EOS-Modells, ermöglicht eine systematische Unterstützung der Methoden und erlaubt eine Navigation durch das Prozeßmodell. Die Navigation wird durch einen Systembaum und durch Aktivitätsdiagramme ermöglicht. Die Aktivitätsdiagramme beschreiben die einzelnen Subprozesse in grafischer Form. Die systematische Unterstützung der einzelnen Methoden erfolgt durch ihre Einbettung in die Aktivitätsdiagramme. Von hieraus können die Methoden angestoßen und gesteuert werden. Die Integration der einzelnen Subprozesse wird durch eine Funktion zur Prozeß-Auswahl ermöglicht. Der Prototyp enthält eine Reihe von vollständig implementierten Komponenten, wie z.B. eine Komponenten zur Aufwandsschätzung von EOS-Projekten. Für eine komplette Implementierung wird ein mehrschichtiges Architektur-Konzept vorgeschlagen.

Die Arbeit besteht aus zwei Teilen. Der erste Teil befaßt sich mit dem Aspekt der Werkzeug-Unterstützung und der zweite Teil mit dem Aspekt der Methoden-Unterstützung. Nach diesem ersten einführenden Kapitel werden im zweiten Kapitel Prozeßmodelle allgemein behandelt. Hierbei wird u.a. auch das EOS-Modell detailliert vorgestellt, da es die Grundlage für diese Arbeit bildet.

Im dritten bis fünften Kapteil wird die Werkzeug-Unterstützung betrachtet. Im dritten Kapitel werden die Aktivitätsdiagramme für die fünf Subprozesse diskutiert. Wie bereits erwähnt, beschreiben die Aktivitätsdiagramme die einzelnen Subprozesse in grafischer Form. Die Aktivitätsdiagramme sind Bestandteil des Prototypen und werden zusammenfassend beschrieben. Details können dem Prozeß-Handbuch im Anhang entnommen werden. Das vierte Kapitel gliedert sich in zwei Abschnitte und widmet sich dem Prototypen für ein Prozeßmanagement-System. Im ersten Abschnitt wird anhand der Benutzerschnittstelle des

Prototypen gezeigt, wie ein Anwendungssystem zur systematischen Unterstützung des EOS-Modells aufgebaut sein sollte. Exemplarisch wird im zweiten Abschnitt eines der vollständig implementierten Module beschrieben. Hierbei wird ein Modul zur Aufwandsschätzung von EOS-Projekten behandelt. Ausgehend von dem Prototypen wird im fünften Kapitel eine Architektur für ein reales Prozeßmanagement-System vorgeschlagen. Solch ein System muß neben der geforderten Funktionalität auch weitere Anforderungen, wie z.B. an Transaktionsmanagement, Sicherheitsmechanismen oder Persistenzdienste, genügen. Im letzten Kapitel wird die Arbeit zusammengefaßt und mit einigen abschließenden Bemerkungen beendet.

Anhang A beinhaltet das Prozeß-Handbuch, das sich in fünf Kapitel gliedert. Zunächst werden unterschiedliche Rollen definiert, die an einem EOS-Projekt beteiligt sein können. Danach folgt für jeden Subprozeß des EOS-Modells ein Kapitel, das jeweils die definierten Methoden beinhaltet. Zur Dokumentation von Ergebnissen werden zahlreiche Produktmuster bzw. Vorlagen vorgeschlagen. Der Arbeit ist eine CD-ROM mit Quellcode beigelegt. Der Inhalt der CD-ROM wird in Anhang B beschrieben.

Personen-Bezeichnungen sind in der Arbeit funktional gemeint, so daß zwischen einer männlichen und weiblichen Formulierung nicht unterschieden wird. Zur visuellen Orientierung wurden einige Piktogramme mit folgender Semantik verwendet:

 Ziel des Kapitels

 Kenntnisse, die für das Verständnis des Kapitels vorausgesetzt werden

 Detaillierte Inhaltsangabe des Kapitels

 Zusammenfassung des Kapitels

 Liste der für das Kapitel wichtigen und in dem Kapitel zitierten Literatur

2 Prozeßmodelle

In diesem Kapitel werden Prozeßmodelle zur Software-Entwicklung behandelt. Insbesondere wird ein Prozeßmodell für evolutionäre, objektorientierte Software-Entwicklung vorgestellt. Dieses Prozeßmodell bildet die Grundlage der Arbeit und wird in den nächsten Kapiteln vertieft.

☑ Das objektorientierte Paradigma und die Modellierungssprache „UML“ werden als bekannt vorausgesetzt.

2 Prozeßmodelle

Seit Software entwickelt wird, gibt es auch Vorgehens- und Prozeßmodelle zur Strukturierung des Entwicklungsprozesses. Ein Vorgehensmodell dient dazu, einen Software-Lebenszyklus in idealisierter und generalisierter Form zu beschreiben (Synonyme: Phasenmodell, Entwicklungsmodell, Software life cycle). Die Verkörperung eines Rahmens innerhalb dessen projekt-spezifische Software-Prozesse definiert werden, wird als Prozeßmodell bezeichnet. Ausgehend von „Code and fix"-Modellen wurden Phasen- und Wasserfall-Modelle, Transformations-Modelle, V-Modelle, Prototypen-Modelle, Spiral- und prozeßorientierte Modelle, objektorientierte Modelle und viele weitere entwickelt. Im folgenden beschreiben und kommentieren wir kurz einige der Modelle, deren historische Entwicklung in Abbildung 1 dargestellt ist. Details können den angegebenen Literaturquellen entnommen werden. Im Abschnitt 2.1 bis 2.3 werden wir exemplarisch auf drei Prozeßmodelle näher eingehen.

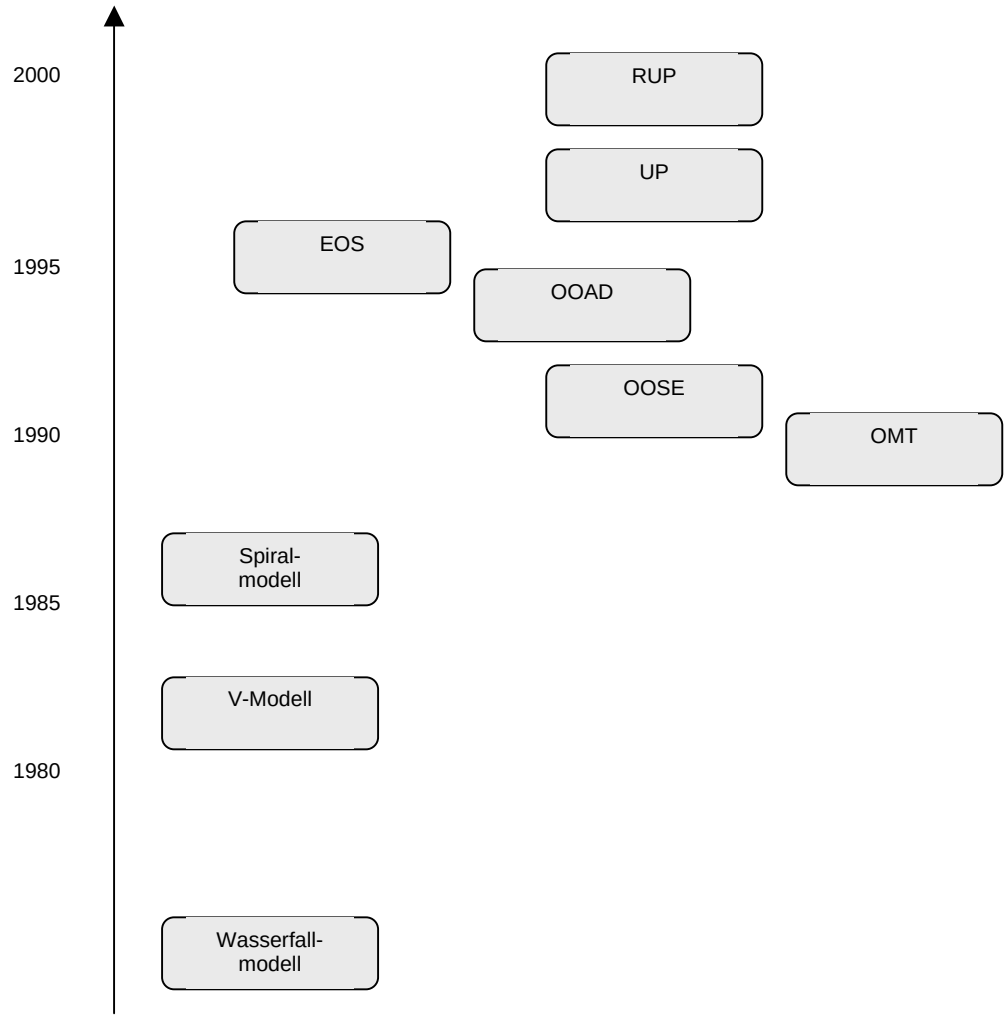


Abb. 1: Historie einige Vorgehens- bzw. Prozeßmodelle

Das *Wasserfall-Modell* gehört zu den ältesten Vorgehensmodellen. Ursprünglich wurde es 1956 von H. Benington veröffentlicht und später u.a. durch W. Royce und B. Boehm erweitert /Benington 1956/, /Royce 1970/, /Boehm 1981/. Das Wasserfall-Modell besteht aus den aufeinanderfolgenden Phasen „System-Anforderungen", „Software-Anforderungen",

„Grob-Entwurf“, „Fein-Entwurf“, „Codierung“, „Testen“ und „Betrieb“. In den ersten beiden Phasen werden die Anforderungen an das zu realisierende Anwendungssystem identifiziert. Ausgehend von den Anforderungen wird in der Grob- und Fein-Entwurfsphasen eine Lösung konzipiert, die in der Codierungsphase in Code umgesetzt wird. Nachdem das Anwendungssystem integriert und getestet ist, wird es in Betrieb genommen (Betriebsphase). Die einzelnen Phasen werden jeweils vollständig durchgeführt. Am Ende jeder Phase werden die erstellten Ergebnisse geprüft. Erst wenn keine Mängel vorliegen, wird die nächste Phase angegangen. Das Wasserfall-Modell ist einfach, leicht verständlich und benötigt wenig Managementaufwand. Durch das starren Phasenmodell werden aber Risiken und Probleme oft zu spät erkannt.

Das *V-Modell* ist eine Erweiterung des Wasserfall-Modells. Beim V-Modell wurde die Qualitätssicherung noch intensiver im Vorgehensmodell eingebunden /Boehm 1984/. Die entstehenden Produkte werden beim V-Modell stets verifiziert (*Wird ein korrektes Produkt entwickelt?*) und validiert (*Wird das richtige Produkt entwickelt?*). Ausgehend von diesem V-Modell wurde später zwecks Einsatz bei Behörden Erweiterung vorgenommen. Das V-Modell für Behörden wurde in den Subprozessen „Systemerstellung“, „Qualitätssicherung“, „Konfigurationsmanagement“ und „Projektmanagement“ gegliedert /V-Modell 1997/. Inzwischen gibt es auch Varianten des V-Modells, die den Einsatz von objektorientierten Methoden erlauben /www.v-modell-iabg.de/. Durch die Integration und die detaillierte Darstellung der vier Subprozesse ist das V-Modell für große Projekte gut geeignet und ermöglicht eine standardisierte Abwicklung. Bei kleinen und mittleren Projekten fallen trotz eines Zuschnittes (*Tailoring*) eine Vielzahl unnötiger Produkte an, die ohne Werkzeugunterstützung nicht bewältigt werden können.

Bei dem *Spiralmodell* von B. Boehm handelt es sich um ein Metamodell /1988/. Denn das Spiralmodell erlaubt die Kombination von Prozeßmodellen zur Erstellung eines Produkts. Für die Produkte der nächsten Phase (hier als Teilprodukt bzw. Inkrement bezeichnet) und Verfeinerungsebenen werden vier zyklische Schritte durchlaufen. Zunächst werden die Ziele des Teilprodukts bestimmt und alternative Möglichkeiten zur Realisierung des Teilprodukts entwickelt. In einem zweiten Schritt werden die Alternativen unter Berücksichtigung der Ziele und Randbedingungen evaluiert und Strategien zur Abwehr von Risiken ausgearbeitet. Im dritten Schritt wird in Abhängigkeit von den Risiken ein Prozeßmodell für die Entwicklung des Teilprodukts festgelegt, z.B. Wasserfall-Modell oder Protoypen-Modell. Zuletzt werden die Ergebnisse der ersten drei Schritte überprüft und der nächste Zyklus geplant. Das Spiralmodell ist sehr flexibel, erlaubt die Integration anderer Prozeßmodelle und ermöglicht die Minimierung von Risiken. Da oft Entscheidungen über den weiteren Verlauf des Prozesses getroffen werden, ist der Aufwand für das Projektmanagement sehr hoch. Das Spiralmodell steht in der Tradition des Wasserfall-Modells, da z.B. die Analyse und der Entwurf sich auf das Gesamtsystem beziehen. Wegen des evolutionären Charakters des Spiralmodells, werden wir im nächsten Abschnitt darauf noch genauer eingehen.

Die von J. Rumbaugh et. al. vorgeschlagene *Object Modeling Technique (OMT)* gliedert die Systemerstellung in die sequentiellen Phasen „Problemstellung“, „Analyse“, „Systementwurf“, „Objektentwurf“, „Implementierung“, „Test“ und „Operationale Phase“ /Rumbaugh 1991/. Das Analyse-Modell besteht bei OMT aus drei Teilmodellen, dem Objektmodell, dem Dynamischen Modell und dem Funktionalen Modell. Objektorientierte Aspekte, wie z.B. die Identifikation von Klassen, werden beim „Objektentwurf“ berücksichtigt. Nach der Implementierung und dem Test wird das Anwendungssystem in Betrieb genommen. Diese letzten Phasen werden beim OMT jedoch sehr spärlich erklärt.

OMT bietet daher hier kaum eine konkrete Hilfestellung. OMT wird von der Wasserfall-Struktur dominiert und erschwert somit das frühzeitige Erkennen von Risiken.

In dem *Object-Oriened Analysis and Design (OOAD)* Ansatz von G. Booch werden zwei Arten von Zyklen unterschieden: Mikro- und Makro-Prozeß. Der Makro-Prozeß ähnelt dem traditionellen Wasserfall-Modell und gliedert sich in den Phasen „Konzeptualisierung", „Analyse", „Entwurf", „Evolution" und „Wartung". Die neu eingeführte Konzeptualisierungs-Phase dient zur Identifikation von Kern-Anforderungen. Statt der üblichen Implementierungs-Phase steht bei OOAD die „Evolution". Damit wird der evolutionäre Charakter dieser Phase auch terminologisch betont. Auch die darauf folgende Wartungs-Phase hebt den evolutionären Charakter hervor, da nach der Wartung neue Makro-Zyklen angestoßen werden können. Der Makro-Prozeß bildet den steuernden Rahmen für die darin enthaltenen Mikro-Prozesse. Ein Miro-Entwicklungsprozeß besteht aus den Schritten „Klassen ermitteln", „Semantik der Klassen ermitteln", „Beziehungen zwischen Klassen ermitteln" und „Klassen implementieren". Der Mikro-Prozeß ähnelt eher einer spiralartigen Vorgehensweise und ermöglicht eine iterativ-inkrementelle Entwicklung. OOAD unterstützt die „Objekt-orientierung im Kleinen", bleibt aber „im Großen" dem Wasserfall-Modell treu.

Auch das von I. Jacobson vorgeschlagene *Object-Oriented Software Engineering (OOSE)* beinhaltet zwei Arten von Zyklen. Es wird zwischen methodisch-technischen und Management-Aspekten getrennt. Daher werden zwei parallele Prozesse unterschieden: Software-Entwicklung und Projektmanagement. Der Prozeß zur Software-Entwicklung gliedert sich in die Phasen „Vorphase", „Anforderungs-Modellierung", „Objekt-Modellierung", „System-Konstruktion" und „System-Test". Dieser Prozeß beschreibt die Sicht von objektorientierter Systementwicklung. Ausgehend von den identifizierten Anwendungsfällen (*use cases*), wird die Software als System kooperierender Klassen bzw. Objekte modelliert. Nach den Phasen „System-Konstruktion" und „System-Test" steht ein lauffähiges Anwendungssystem zur Verfügung. Der Projektmanagement-Prozeß konzentriert sich dagegen auf planende und steuernde Aktivitäten. Hierbei werden folgende fünf Phasen unterschieden: „Vorstudie", „Machbarkeitsstudie", „Etablierung", „Ausführung" und „Abschluß". In OOSE wird zu Recht zwischen Projektmanagement und Software-Entwicklung unterschieden. Die beiden Prozesse laufen im Wesentlichen linear ab und berücksichtigen zu wenig den iterativen Charakter.

Die drei oben betrachteten Vorgehensmodelle zur objektorientierten Software-Entwicklung folgen dem hybriden Ansatz. Das bedeutet, daß die objektorientierte Software-Entwicklung auf einer untergeordneten Klassen- bzw. Objektebene stattfindet und in einem übergeordneten, wasserfall-artigen Gesamtprozeß eingebettet ist. Detaillierte Ausführungen und Analysen können in der Literatur nachgelesen werden, z.B. in /Hesse 1997b/ oder /Martin, Odell 1995/.

Anfang 1999 wurde der *Unified Software Development Process (UP)* von I. Jacobson, G. Booch und J. Rumbaugh publiziert /Jacobson, Booch, Rumbaugh 1999/. Mit UP versuchen die drei Autoren ihre unterschiedlichen Vorgehensmodelle zu einem gemeinsamen Prozeßmodell zu vereinen. Ausgehend vom UP veröffentlichte die Firma „Rational Software" den *Rational Unified Process* (*RUP*) /Kruchten 1999/. Der RUP ist eine spezielle Ausprägung von dem generischen UP. Er gliedert sich in eine zeitliche und eine inhaltliche Dimension. In der zeitlichen Dimension wird zwischen Phasen und Iterationen unterschieden. Während die technische Realisierung in Iterationen statt findet, erfolgt die Projektsteuerung in Phasen. Neben den Phasen werden Prozesse unterschieden. Diese sind phasenübergreifend und orientieren sich am Inhalt der Tätigkeiten. Der RUP steht mit seinen Phasen und

Meilensteinen in der Tradition des Wasserfall-Modells und übernimmt daher auch Nachteile des Wasserfall-Modells. In der Praxis wird der RUP inzwischen vermehrt eingesetzt und in der Forschung intensiv diskutiert. Daher werden wir weiter unten den RUP detailliert vorstellen und kritisch analysieren. Da das Sprialmodell zu den wenigen Prozßmodellen gehört, die den evolutionären Aspekt der Software-Entwicklung berücksichtigen, werden wir es im nächsten Abschnitt näher beschreiben und bewerten.

2.1 Spiralmodell

Das Spiralmodell von B. Boehm ist ein risikogetriebenes Prozeßmodell, das in einer Zeit entstand, in der mit Softwareprojekten noch in der Regel schlechte Erfahrungen gemacht worden /Boehm 1986/, /Boehm 1988/. Hervorstechende Probleme, auf die es B. Boehm ankam, waren u.a. mangelnde Berücksichtigung der Anwender-Bedürfnisse, unzuverlässige Programme, Doppelt-Entwicklungen oder sehr hohe Entwicklungskosten.

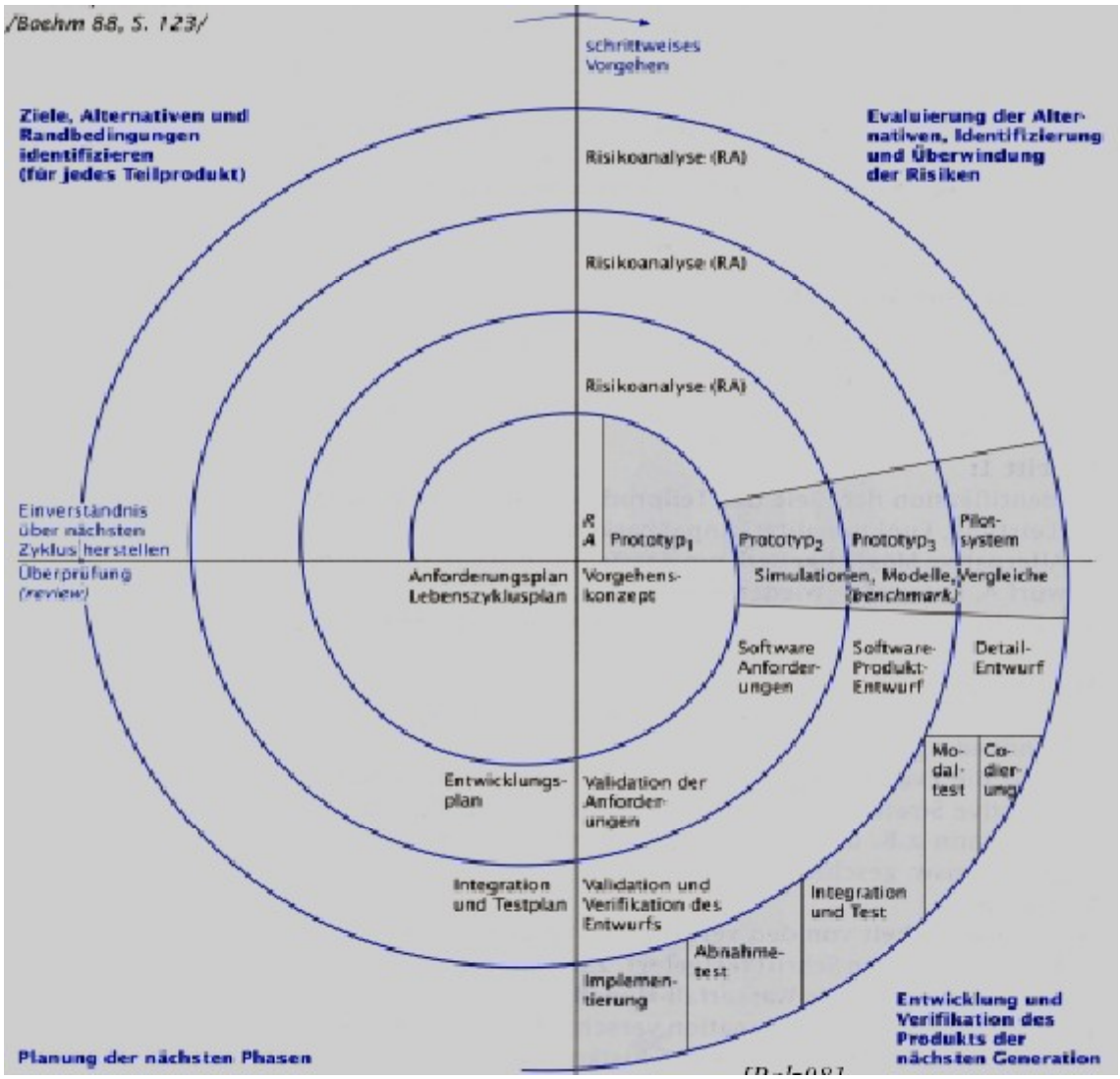


Abb. 2: Das Spiralmodell /Boehm 1988/

Im Spiralmodell wird die Software-Entwicklung als ein zyklischer Prozeß verstanden. Jede Windung der Spirale enthält die Aktivitäten (Abb. 2):

- *Ziele, Alternativen und Randbedingungen identifizieren (für jedes Teilprodukt),*
- *Alternativen evaluieren, Risiken identifizieren und überwinden,*
- *Produkt der nächsten Generation entwickeln und verifizieren,*
- *Nächste Phase planen*

Jede Spiral-Windung stellt einen Zyklus durch dieselben Aktivitäten dar. Am Ende jeder Windung steht ein Review, bei dem der aktuelle Projektfortschritt bewertet und geprüft wird. Anschließend werden die Pläne für die nächste Windung verabschiedet sowie die benötigten Ressourcen festgelegt oder aber das Projekt abgebrochen. Das Spiralmodell ist als ein Metamodell zu verstehen, da in den Zwischenschritten andere Prozeßmodell verwendet werden können. Die Fläche der Spirale repräsentiert die akkumulierten Kosten, die bei der bisherigen Entwicklung angefallen sind. Der Winkel eines Punkts auf der Spirale - bezogen auf die waagerechte Hauptachse - zeigt den Entwicklungsfortschritt des jeweiligen Zyklus an. Beim Spiralmodell kann von Kernanforderungen ausgegangen und das Software-Produkt allmählich und stufenweise entwickelt werden. Somit ist der evolutionäre Aspekt der Software-Entwicklung im Prozeßmodell verankert.

Im folgenden werden die oben genannten vier Aktivitäten näher beschrieben.

2.1.1 Ziele, Alternativen und Randbedingungen identifizieren

Am Begin jedes Zyklus werden inhaltliche Vorgaben an das zu entwickelnde Teilprodukt festgelegt. Solche inhaltliche Vorgaben könnten z.B. Funktionalität oder Qualitätskriterien sein. Neben den inhaltlichen Vorgaben werden alternative Vorgehensweisen herausgearbeitet, wie z.B. unterschiedliche Analyse- oder Entwurfstechniken, Einsatz von Werkzeugen oder Kauf von Software. Als Rahmenbedingungen werden Einschränkungen bzgl. Zeit, Personal, Kosten, Hard- und Software-Umgebungen festgelegt. Diese sind bei den verschiedenen Alternativen zu beachten.

2.1.2 Alternativen evaluieren, Risiken identifizieren und überwinden

In diese Phase werden die Alternativen hinsichtlich der festgelegten Ziele und Rahmenbedingungen untersucht und bewertet. Zeigt die Evaluierung, daß es Risiken gibt, dann ist eine kosteneffektive Strategie zu entwickeln, um die Risiken zu überwinden bzw. zu vermindern. Dazu können unterschiedliche Techniken eingesetzt werden, wie z.B. Prototyp-Entwicklung, Simulation, Benchmark-Tests oder Benutzerbefragung. Die weitere Projektarbeit gründet sich auf das verbliebene Restrisiko. Dabei hat beim weiteren Vorgehen die Minimierung des Restrisikos die höchste Priorität.

2.1.3 Produkt der nächsten Generation entwickeln und verifizieren

In diesem Schritt wird die gewählte Alternative unter Einhaltung der Ziel- und Ressourcen-Vorgaben realisiert und getestet. In Abhängigkeit von den verbleibenden Risiken wird das Prozeßmodell für diesen Schritt festgelegt, z.B. evolutionäres Modell, Prototypen-Modell oder Wasserfall-Modell. Die Entscheidung für ein Prozeßmodell bzw. eine sinnvolle Kombination mehrerer Vorgehensweisen erfolgt ausschließlich risikoorientiert und exklusiv für die aktuelle Windung.

2.1.4 Nächste Phase planen

In diesem Schritt wird der nächste Zyklus inhaltlich und organisatorisch geplant. Die Planung kann sich auch auf mehrere Zyklen beziehen und ist nicht auf den unmittelbar folgenden Zyklus beschränkt. Außerdem ist die Aufteilung des Projekts in weitgehend unabhängige Teilprojekte möglich. Die Teilprojekte können von verschiedenen Entwicklungsteams parallel durchgeführt und zu einem späteren Zeitpunkt integriert werden. In einem Review werden die Projektfortschritte und alle entwickelten Produkte des letzten Zyklus analysiert, die Ergebnisse bewertet und die Projektperspektiven diskutiert. Wenn z.B. die technischen oder wirtschaftlichen Risiken einer Projektfortsetzung zu hoch sind, dann wird das Projekt abgebrochen. Erfolgt kein vorzeitiger Projektabbruch, so liegt die Planung für den nächsten Zyklus vor.

2.1.5 Bewertung des Spiralmodells

Das Spiralmodell weist folgende Vorteile auf:

- Das Spiralmodell ist risikogetrieben und hat die Minimierung des Risikos als oberstes Ziel. Der Risikobezug orientiert sich im besonderen an den angelaufenen Kosten des Projekts. Im Spiralmodell werden Techniken zu Reduktion von Risiken vorgeschlagen, wie z.B. die Entwicklung von Prototypen.
- Durch die Evaluierung von Alternativen werden Fehler und ungeeignete Alternativen frühzeitig eliminiert. Die Überprüfung der Ergebnisse am Ende eines Zyklus bereinigt weitere Fehler bzw. Fehlentscheidungen.
- Die Neuheit beim Spiralmodell war, daß von einer zyklischen Entwicklung ausgegangen wurde. Dieser Fortschritt unterschied das Spiralmodell von klassischen Modellen, die bis Ende der 80 Jahre veröffentlicht worden waren.
- Da beim Spiralmodell nicht ein festgelegtes Prozeßmodell für die gesamte Entwicklung verwendet wird, bietet es eine hohe Flexibilität.
- Die Wiederverwendung von Software wird im Prozeßmodell berücksichtigt.

Dem stehen folgende Nachteile gegenüber:

- Das Spiralmodell ist hoch komplex und birgt einen hohen Administrationsaufwand. Daher ist es wenig geeignet für kleine und mittlere Projekte. Wünschenswert wäre aber ein Prozeßmodell, das unabhängig von der Projektgröße ist.
- Beim Spiralmodell können zwar unterschiedliche Prozeßmodelle zum Einsatz kommen, jedoch entsteht hierdurch ein hoher Managementaufwand, da über den weiteren Prozeßablauf stets neu entschieden werden muß. Der Einsatz unterschiedliche Prozeßmodelle setzt voraus, daß die Projektmitglieder über sehr hohe Qualifikation verfügen müssen. Wünschenswert wäre aber ein Prozeßmodell, das sich mit wenig Aufwand auf die jeweiligen Projektbedürfnisse zuschneiden läßt.
- Im Spiralmodell wird keine klare Trennung zwischen den einzelnen Teilprozessen eines Softwareprojekts vorgenommen. Der Teilprozeß *Software-Entwicklung* vollzieht sich im wesentlichen in den ersten drei Schritten eines Zyklus. Der Teilprozeß *Projekt-management* findet hauptsächlich im vierten Schritt bei der Planung statt, begleitet aber das Projekt durch die Risikoanalyse. Weitere Teilprozesse werden überhaupt nicht berücksichtigt, wie z.B. *Qualitätssicherung* oder *Konfigurationsmanagement*. Wünschens-wert wäre ein Prozeßmodell, das die einzelnen Teilprozesse separat nebeneinander betrachtet und detailliert beleuchtet.

- Projektübergreifende Prozesse, wie z.B. *Qualitätsmanagement*, *Wiederverwendungs-management*, *Geschäftsprozeß-Modellierung* oder *Prozeß-Verbesserung*, werden kaum berücksichtigt. Wünschenswert wäre ein Prozeßmodell, das zu genannten unternehmensweiten Prozessen Schnittstellen vorsieht.
- Gegenstand eines Zyklus im Spiralmodell ist ein Teilprodukt bzw. ein Inkrement, an dem die vier oben genannten Aktivitäten ausgeübt werden. Der Begriff Teilprodukt wird sehr unscharf formuliert und nicht nähr definiert. Vernünftig wäre es, wenn ein Zyklus an den Bausteinen (z.B. Komponente oder Klasse) ausgeführt werden würde. Daher wäre ein Prozeßmodell wünschenswert, das die Systemstruktur mit ihren Bausteinen berücksichtigt.
- Vernünftigerweise wird im Spiralmodell die Wiederverwendung im Sinne von gekaufter Software berücksichtigt. Das Ablegen von Ergebnissen in einer Bibliothek wird jedoch nicht ausreichend betrachtet. Dies liegt sowohl an der unscharfen Definition des Begriffs „Teilprodukt“ (besser wäre Baustein) als auch am Fehlen eines Gesamtkonzepts zur Wiederverwendung. Wünschenswert wäre ein Prozeßmodell, das ein Gesamtkonzept für die Wiederverwendung beinhaltet. Zu einem Gesamtkonzept gehören z.B. ein Bausteinbibliotheks-Konzept oder definierte Prozesse zum Ablegen bzw. Entnehmen von wiederverwendbarer Software.
- Eine Operationalisierung des Prozeßmodells fehlt, d.h. es werden keine konkrete Methoden angegeben. Diese aufwendige und komplexe Tätigkeit muß somit von Projektmitgliedern im Vorfeld eines Projekts erbracht werden. Wünschenswert wäre aber ein Prozeßmodell, das zumindest eine Basis an definierten Methoden zur Weiter-entwicklung bereitstellt.
- Eine Prozeß-Unterstützung durch Werkzeuge, Produktmuster oder Richtlinien wird erheblich erschwert, da unterschiedliche Prozeßmodelle zum Einsatz kommen können. Wünschenswert wäre ein Prozeßmodell, das eine systematische Werkzeugunterstützung durchgängig ermöglicht.
- Es bleiben eine Reihe von Fragen offen, wie z.B.: *Wie werden Zyklen synchronisiert? Wie werden Meilensteine definiert? Wie erfolgt die Planung für den aller ersten Zyklus (denn die Zyklus-Planung findet jeweils im vierten Schritt statt)?*

2.2 Rational Unified Process (RUP)

Der RUP wird als ein anwendungsfallgetriebenes, architekturzentriertes und iterativ-inkrementelles Prozeßmodell bezeichnet. Er gliedert sich in eine *zeitliche* und eine *inhaltliche* Dimension (Abb. 3).

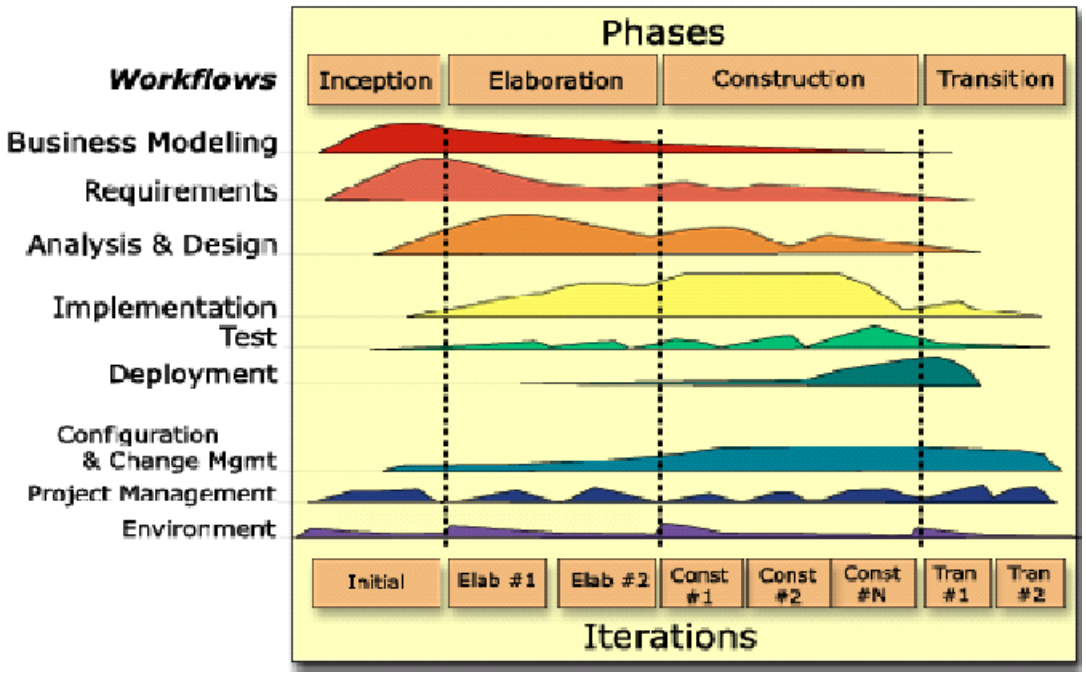


Abb. 3: RUP im Überblick /Kruchten 2000/

2.2.1 Zeitliche Dimension

In RUP ist die zeitliche Dimension in Phasen und Iterationen gegliedert. Während die technische Realisierung in Iterationen abläuft, erfolgt die Projektsteuerung anhand der Phasen. Es werden vier aufeinander folgender Phasen unterschieden:

Grobkonzeptphase (*Inception*)

In dieser Phase wird unter allen Projektbeteiligten ein einheitliches Verständnis über die Ziele und den Gegenstandsbereich des Projekts geschafft. Die kritischen Anforderungen werden identifiziert. Aus ihnen werden Kriterien zur Akzeptanz bzw. zur Ablehnung des Endsystems abgeleitet. Eine erste grobe Architektur des zu realisierenden Systems wird entwickelt und diskutiert. Der Entwicklungsaufwand und die Projektrisiken werden abgeschätzt. Die wesentlichen Ergebnisse dieser Phase sind eine Machbarkeitsstudie und ein Projektplan, der die Phasen und Iterationen beinhaltet. Am Ende dieser Phase ist der erste Meilenstein erreicht (genannt *Lifecycle Objective*). Hierbei wird entschieden, ob das Projekt fortgeführt oder abgebrochen wird.

Entwurfsphase (*Elaboration*)

Im Rahmen dieser Phase werden die funktionalen und nicht-funktionalen Anforderungen analysiert. Das so entwickelte Analysemodell sollte bis zu 80% der Anforderungen enthalten. Je nach Bedarf werden auch die Geschäftsprozesse modelliert und optimiert. Die Architektur wird überarbeitet und die Komponenten identifiziert. Es wird ein lauffähiger Prototyp zur Evaluierung der Architektur entwickelt. Die Entwicklungsumgebung und die benötigten Werkzeuge werden eingerichtet. Parallel hierzu wird der Entwicklungsprozeß auf die Bedürfnisse des Projekts zugeschneidert. Der Projektplan und die Risikoliste werden verfeinert und ergänzt. Die wichtigsten Ergebnisse dieser Phase sind ein Analysemodell, eine dokumentierte Architektur und ein überarbeiteter Projektplan. Die Phase endet mit dem *Lifecycle Architekture* Meilenstein. Hierbei werden die erzielten Ergebnisse überprüft und über den Fortgang des Projekts entschieden.

Konstruktionsphase (*Construction*)

In dieser Phase werden die einzelnen Komponenten programmiert und getestet. Das spezifizierte Anwendungssystem entsteht durch die Integration der fertiggestellten Komponenten. Die notwendigen Ressourcen werden optimiert, um die Entwicklungskosten möglichst gering zu halten. Als Ergebnis dieser Phase liegt ein *beta* Release des Anwendungssystems vor. Außerdem muß ein Benutzerhandbuch angefertigt werden. Am Ende dieser Phase ist der *Initial Operational Capability* Meilenstein erreicht. Es ist zu entscheiden, ob das Anwendungssystem soweit ausgereift ist, um beim Kunden eingesetzt werden zu können. Außerdem wird der aktuelle Projektstand mit den Planwerten verglichen und gegebenenfalls Korrekturen vorgenommen.

Übergangsphase (*Transition*)

In dieser Phase wird das entwickelte Anwendungssystem dem Anwender übergeben. Es finden Beta-Tests statt, um zu überprüfen, ob das Anwendungssystem den definierten Funktionsumfang erbringt. Auftretende Mängel, wie z.B. Fehler oder niedrige Performanz, werden beseitigt. Das Anwendungssystem inklusive Installationsroutinen werden auf Medien, wie z.B. CD-ROM, kopiert und den Anwendern zur Verfügung gestellt. Die Anwender werden geschult, Hotlines eingerichtet und Handbücher angefertigt. Am Ende dieser Phase ist das Anwendungssystem ausgeliefert und der letzte Meilenstein (*Product Release*) erreicht. Je nach Bedarf wird entschieden, ob weitere Entwicklungszyklen (*development cycle*) durchgeführt werden.

Iterationen (*Iterations*)

Jede der vier Phasen kann aus einer oder mehreren Iterationen bestehen. Eine Iteration besteht aus einer Folge von Aktivitäten (Abb. 4). Der Schwerpunkt der einzelnen Iterationen ist unterschiedlich. Bei einer Iteration während der Entwurfsphase werden z.B. die architekturbezogenen Aktivitäten intensiver als die Entwicklungstätigkeiten sein. Die Anzahl und Dauer der Iterationen hängt vom Umfang und der Komplexität des Projekts ab. Durch die Unterteilung der Phasen in Iterationen ergeben sich weitere Meilensteine (Ende jeder Iteration), die neben den Phasen-Meilensteinen als Anhaltspunkt für die Projektsteuerung genutzt werden können.

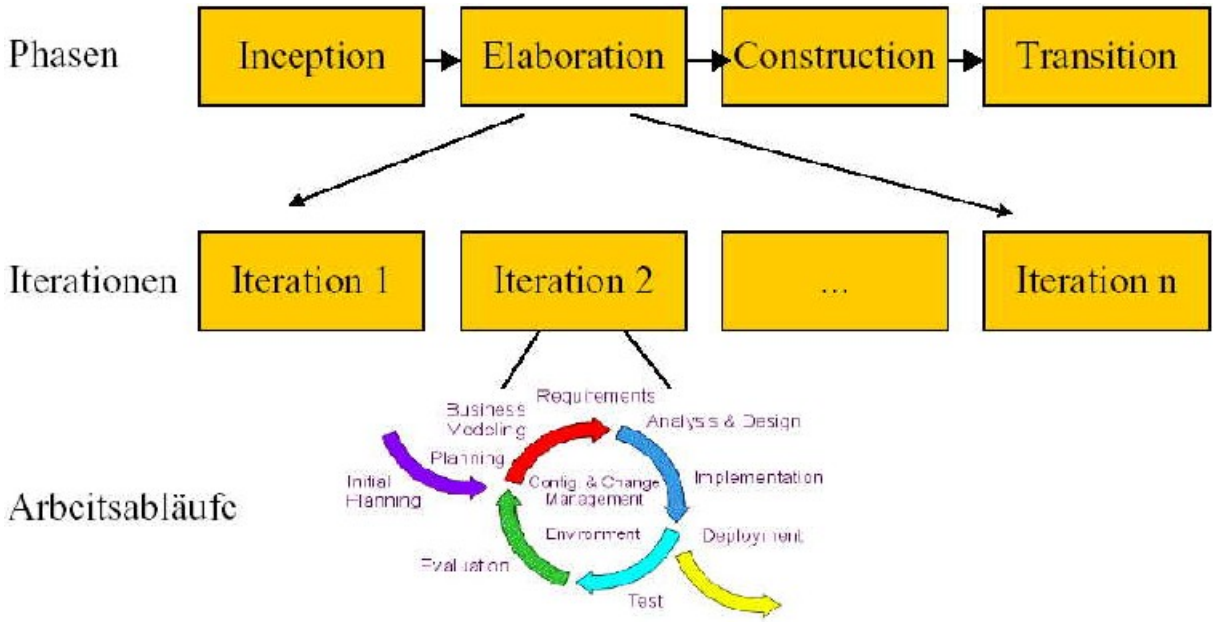


Abb. 4: Zusammenhang zwischen Phasen, Iterationen und Arbeitsabläufe

2.2.2 Inhaltliche Dimension

Neben den vier Phasen werden neun Prozesse (*Workflows*) unterschieden. Diese sind phasenübergreifend und orientieren sich am Inhalt der Tätigkeiten. Die einzelnen Prozesse gliedern sich in Aktivitäten (*Activities*), Produkte (*Artifacts*) und Rollen (*Workers*). Durch eine Aktivität wird beschrieben, was gemacht werden soll. Das Ergebnis einer Aktivität wird als Produkt bezeichnet. Ein Produkt kann auch als Eingangsinformation für eine Aktivität dienen. Aktivitäten werden den Projektmitgliedern über Rollen zugeordnet. Eine Rolle faßt die Kenntnisse, Fähigkeiten und Erfahrungen zusammen, die zur Durchführung einer Aktivität benötigt werden. In der Abbildung 3 ist der Aufwand der jeweiligen Prozesse pro Phase grob veranschaulicht. Folgende neuen Prozesse werden unterschieden.

Geschäftsprozeß-Modellierung (*Business Modeling*)

Das Ziel der Geschäftsprozeß-Modellierung ist es, die Organisationsstruktur der Anwender zu analysieren. Hierbei können zusätzliche Anforderungen an das zu realisierende Anwendungssystem identifiziert werden. Ausgehend von einer ersten groben Bewertung der Organisationsstruktur werden die Geschäftsprozesse identifiziert, optimiert und dokumentiert. Das wesentliche Ergebnis dieses Prozesses ist ein dokumentiertes Geschäftsmodell.

Anforderungsermittlung (*Requirements*)

Im Rahmen dieses Prozesses wird geklärt, was das Anwendungssystem leisten soll. Dazu werden die Probleme und Bedürfnisse der Anwender analysiert. Aus diesen Informationen werden funktionale und nicht-funktionale Anforderungen abgeleitet und der Gegenstandsbereich des Projekts abgegrenzt. In einem zweiten Schritt werden die Anforderungen verfeinert, indem z.B. ein Oberflächen-Prototyp implementiert wird. Das wichtigste Resultat dieses Prozesses ist ein Anwendungsfallmodell und gegebenenfalls auch ein Oberflächen-Prototyp.

Analyse und Entwurf (*Analysis & Design*)

Ausgehend von den identifizierten Anforderungen wird in diesem Prozeß beschrieben, wie das Anwendungssystem zu entwickeln ist. Dazu wird zunächst eine grobe Architektur des Anwendungssystems definiert, die im Verlauf des Prozesses verfeinert wird. Die definierten Anforderungen werden überprüft und aus ihnen Komponenten abgeleitet. Es wird ein systemweites Entwurfsmodell (*Design Model*) erstellt, das die benötigten Klassen und ihre Beziehungen darstellt. Beim Einsatz eines Datenbanksystems muß auch eine Datenmodellierung vorgenommen werden. Die wichtigen Produkte dieses Prozesses sind eine dokumentierte Architektur, ein Entwurfsmodell und gegebenenfalls ein Datenmodell.

Implementierung (*Implementation*)

Im Rahmen dieses Prozesses wird ein Implementierungsmodell erstellt, das die Subsysteme und ihre Beziehungen darstellt. Subsysteme sind ablauffähige Teile des Anwendungssystems. Sie entstehen aus der Integration von Komponenten. Nachdem das Implementierungsmodell angefertigt ist, wird die Integration der Komponenten geplant. Die Komponenten werden einzeln implementiert und getestet. Für das Testen des gesamten Anwendungssystems ist ein eigener Prozeß vorgesehen. Zur Reduzierung von Risiken wird der Einsatz von Prototypen vorgeschlagen. Die interessanten Resultate dieses Prozesses sind ein Implementierungsmodell, ein Integrationsplan und implementierte Komponenten.

Test (*Test*)

Das Ziel dieses Prozesses ist es, die Integration von Komponenten zu prüfen. Es wird überprüft, ob alle Anforderungen implementiert und alle identifizierten Fehler beseitigt worden sind. Der Test-Prozeß besteht aus Planung, Entwurf, Implementierung, Durchführung und Evaluierung von Tests. Das Testen ist mit kontinuierlich wachsendem Aufwand über alle vier Phasen verteilt. Die wesentlichen Produkte dieses Prozesses bestehen aus dem Testplan, der Testspezifikation und den Testresultaten.

Einsatz (*Deployment*)

Im Rahmen dieses Prozesses wird das Anwendungssystem in der Zielumgebung getestet (*beta test*). Es werden Installationsroutinen entwickelt und die Software auf Speichermedien, wie z.B. CD-ROM, kopiert. Die Anwender werden durch Schulungen und Handbücher auf den Einsatz des Anwendungssystems vorbereitet. Folgende Produkte entstehen bei diesem Prozeß: einsatzbereites Anwendungssystem, Handbücher und Schulungsunterlagen.

Konfigurations- und Änderungsmanagement (*Configuration & Change Management*)

Zu diesem Prozeß gehört die Identifizierung der zu verwaltenden Einheiten des zu erstellenden Anwendungssystems, das Überwachen von Änderungen, das Managen und Definieren der Konfigurationen der genannten Einheiten. Dieser Prozeß ist mit wachsendem Aufwand über alle vier Phasen verteilt. Der Konfigurationsmanagement-Plan und die Dokumentation der Änderungen sind die wesentlichen Produkte des Prozesses.

Projektmanagement (*Project Management*)

Das Ziel dieses Prozesses ist es, Methoden zur Planung und Steuerung von Projekten bereitzustellen. Zur Projektplanung werden die Phasen und die in ihnen enthaltenen

Iterationen herangezogen. Eine grobe Planung wird anhand der vier Phasen-Meilensteine vorgenommen. Das Ende jeder Iteration definiert weitere Meilensteine innerhalb einer Phase. In einer Iteration können mehrere der neuen Prozesse beteiligt sein. Die durchzuführenden Aktivitäten und die erzeugten Produkte können zur Definition von Meilensteinen innerhalb einer Iteration genutzt werden. Somit kann die erste grobe Planung mit dem Fortgang des Projekts verfeinert werden. Zur Steuerung des Projekts werden die definierten Meilensteine genutzt, um Ist-Plan-Vergleiche vorzunehmen. Ein weiterer wichtiger Aspekt ist das Risikomanagement. Hierbei werden Projektrisiken identifiziert, entsprechende Risikostrategien eingeleitet und erkannte Risiken verfolgt. Der Projektplan mit der enthaltenen Risikoliste und der Iterationsplan sind die wichtigsten Ergebnisse des Prozesses.

Umgebung (*Environment*)

Zu den Aktivitäten des Umgebungs-Prozesses gehört die Bereitstellung der notwendigen Entwicklungswerkzeuge für die jeweiligen Projektmitglieder. Außerdem wird der Rational Unified Process auf das aktuelle Projekt zugeschneidert. Das Einrichten der Umgebung vollzieht sich auf der Projekt- und Iterations-Ebene. Im ersten Fall werden projektübergreifende Vorkehrungen, wie z.B. das Einrichten von Vorlagen oder die Definition von Richtlinien, getroffen. Auf der Iterations-Ebene werden Vorbereitungen durchgeführt, die sich nur auf einzelne Iterationen beziehen, so z.B. das Zuschneidern von Prozessen. Das wichtigste Produkt dieses Prozesses ist ein Zuschnitt von RUP (*development case*).

2.2.3 Bewertung des Rational Unified Process

Es ist vernünftig, daß in RUP bewährte Konzepte genutzt werden, wie z.B. inkrementelle Entwicklung. Durch das breite Angebot von Werkzeugen der Firma „Rational“ ist außerdem eine softwaregestützte Entwicklung gut möglich. Da der RUP inzwischen in der Praxis weit verbreitet ist, können mit ihm Erfahrungen gesammelt werden, die zur Verbesserung des Prozesses genutzt werden können. Der RUP weist jedoch auch einige grundsätzliche Mängel auf. Diese beruhen unter anderem auf der Schwierigkeit, die sehr unterschiedlichen Vorentwürfe der drei Hauptautoren Jacobson, Rumbaugh und Booch zu vereinen. W. Hesse bemängelt folgende Punkte /Hesse 2000/, /Hesse 2001/:

- Der RUP steht mit seinen Phasen und Meilensteinen in der Tradition des Wasserfall-Modells und genügt daher nur bedingt den Anforderungen der objekt- und komponentenorientierten Entwicklungsmethoden. Das traditionelle Phasen-Konzept ist insbesondere für Manager leicht handhabbar. Es verdeckt aber Risiken und ist wenig für den Einsatz neuerer Methoden, wie z.B. Prototyping, Objektorientierung oder inkrementelle/ evolutionäre Entwicklung geeignet. Der RUP suggeriert mit seinen vier Phasen ein projektweites phasensynchrones Arbeiten. Dies erweist sich aber bei der komponentenorientierten Entwicklung, wo mehrere Komponenten in unterschiedlichen Entwicklungsstadien bearbeitet und koordiniert werden müssen, als eine Fiktion. Das Phasen- und Meilenstein-Konzept ist hierfür zu grob und muß durch differenzierte Strukturen ersetzt werden. Im nächsten Abschnitt werden wir ein Prozeßmodell vorstellen, in dem das Meilenstein-Konzept durch feinere *Revisionspunkte* abgelöst wird. Die projektweiten Phasen werden an die Architektur-Bausteine gebunden sein, so daß eine höhere Flexibilität gegeben sein wird.

- Zwar wird behauptet, daß der RUP architektur-zentriert sei, jedoch wird auf die Architektur zu wenig Bezug genommen. Es werden UML-Modelle zur Spezifikation und Dokumentation der Architektur definiert. Ein architektur-zentriertes Prozeßmodell müßte aber die Systemstruktur mit ihren Bausteinen, Aktivitäten und Arbeitsergebnissen viel stärker in die Entwicklungsmethodik einbringen. Das im nächsten Abschnitt vorgestellte Prozeßmodell nutzt durchgängig die Architektur, um z.B. Aktivitäten, Ergebnisse oder Iterationen zu definieren.
- Ein Vorgehen in Iterationen ist vernünftig. Im RUP werden diese aber an Phasen gebunden. In einem so arbeitsteiligen Prozeß wie der Software-Entwicklung ist es jedoch sehr unwahrscheinlich, daß alle Ergebnisse und Teilprodukte, die in der betreffenden Phase entstanden sind, revisionsbedürftig sind. In der Regel verursachen dagegen Mängel an Entwicklungs-Bausteinen, bzw. den daran gebundenen Ergebnissen, Iterationen. Das bedeutet, daß Iterationen gezielt dort angestoßen werden sollten, wo sie benötigt werden - nämlich bei den Bausteinen und nicht bei den Phasen. Daher sollten Iterationen an Bausteine, wie z.B. Klassen oder Komponenten, gebunden werden und nicht an Phasen (siehe nächster Abschnitt).
- Die früheren Phasennamen „Analyse“, „Entwurf“, „Implementierung“, „ Test“ und „Einsatz“ tauchen im RUP als Prozesse (*Workflows*) auf. Die Prozesse und die vier Phasen stehen in einer unscharfen Beziehung zueinander. Der Prozeß „Analyse und Entwurf“ wird z.B. hauptsächlich in der Entwurfs- und Konstruktionsphase ausgeführt. Wozu braucht man dann beide Konzepte nebeneinander? Diese Sicht suggeriert, daß bestimmte Aktivitäten eines Prozesses kontinuierlich am gesamten System vollzogen werden. Hierbei wird verkannt, daß die Prozesse (*Workflows*) zu verschiedenen Zeitpunkten an unterschiedlichen Bausteinen durchgeführt werden.
- Der Umfang und die Komplexität heutiger Projekte nehmen stetig zu. Zur Beherrschung dieses Problems bietet die Informatik Konzepte wie hierarchische Strukturen und Rekursion. Der RUP macht aber von diesen mächtigen Konzepten keinen Gebrauch und erhöht sogar die Komplexität mit dem überflüssigen *Workflow*-Konzept. Dabei könnten Prozesse an die hierarchische Systemstruktur in Form von Komponenten, Subsystemen, Paketen, Modulen oder Klassen gebunden werden und dementsprechend rekursiv in Unterprozesse aufgegliedert werden. Das würde zu klaren Führungsstrukturen und Verantwortlichkeiten führen. Das Prozeßmodell im nächsten Abschnitt berücksichtigt diesen Zusammenhang.
- Die Software-Entwicklung beschränkt sich nicht nur auf einen technischen Prozeß. Sie vollzieht sich im Zusammenspiel mehrerer parallel verlaufender Teilprozesse. Der RUP sieht in den unterstützenden Workflows drei solche Teilprozesse vor. Die Qualitätssicherung ist jedoch ebenfalls ein unterstützender Workflow und sollte mitberücksichtigt werden. Außerdem spielen die Anwender phasenübergreifend eine entscheidende Rolle. Diese werden aber im RUP nicht ausreichend berücksichtigt.
- Die Wiederverwendung wird zwar in RUP angepriesen, findet jedoch kaum Berücksichtigung. Zum Beispiel sind in keiner der neun Prozesse konkrete Aktivitäten zum Ablegen von Software zwecks Wiederverwendung vorgesehen. Dies liegt u.a. an dem Iterationskonzept, das die Identifikation von Einheiten zur Wiederverwendung erschwert. Bei einem Prozeßmodell, das sich an den Entwicklungsbausteinen orientiert, gestaltet sich die Wiederverwendung systematischer. Die Bausteine bilden nämlich die Grundlage der Wiederverwendung (siehe Abschnitt 2.3 und Kapitel 3).

- Eine Iteration wird aus den Aktivitäten, Produkten und Rollen der neun Prozessen zusammengestellt. Dieses Zuschneidern von Iterationen hat sich aber in der Praxis als eine komplizierte und sehr zeitaufwendige Tätigkeit erwiesen. Denn die inhaltliche Abgrenzung einer Iteration ist oft sehr schwierig und daher auch das Zusammenstellen von Aktivitäten und Produkten. Als Hilfestellung wird zwar im RUP-Panorama (Abb. 3) die Intensität der einzelnen Prozesse über die Phasen grafisch dargestellt. Dies ist aber nicht sehr hilfreich, da der Aufwand einzelner Prozesse bei unterschiedlichen Projekten variiert.

Bei den oben genannten Mängeln haben wir mehrmals auf ein bausteinorientiertes Prozeßmodell verwiesen, das wir im nächsten Abschnitt beschreiben wollen.

2.3 Evolutionäre, objektorientierte Software-Entwicklung (EOS)

Die Software-Entwicklung verläuft in der Regel nicht nach einem starren Phasenschema, sondern evolutionär, d.h. als eine Folge von Erweiterungs- und Anpassungszyklen, beruhend auf Nutzung und Revisionen /Lehman 1980/. Die bereits weiter oben genannten Vorgehensmodelle zu objektorientierten Methoden werden dem evolutionären Charakter der Software-Entwicklung nicht genügend gerecht. Dies zeigte eine von W. Hesse verfaßte Studie /Hesse 1997b/. Um diese Lücke zu füllen, wurde das EOS-Modell entworfen.

2.3.1 EOS-Leitlinien

Das EOS-Modell wurde von W. Hesse im Jahre 1994 vorgeschlagen /Hesse, Weltz 1994/ und an anderen Stellen z.B. /Hesse 1995/ und /Hesse 1998b/ publiziert. Das EOS-Modell soll die Kluft zwischen Theorie und Praxis der Software-Entwicklung überwinden helfen, die Prinzipien der Objektorientierung im Vorgehensmodell einbeziehen und den evolutionären Charakter der Software-Entwicklung berücksichtigen /Hesse 1997b/.

Das EOS-Vorgehensmodell orientiert sich an den folgenden Leitgedanken /Hesse, Weltz 1994/:

- Objektorientierung als durchgängige Entwicklungsmethodik
- Hierarchischer Systemaufbau
- Zyklische Entwicklung
- Einbezug von Erprobung, Nutzung und Revision in die Entwicklungszyklen
- Weiter- und Wiederverwendung von *Software-Bausteinen*
- Mit der Entwicklungsmethodik eng abgestimmte Management-Verfahren

Wir behandeln die Leitlinien in den folgenden Teilabschnitten.

2.3.1.1 Objektorientierung als durchgängige Entwicklungsmethodik

Die objektorientierten Techniken für die Analyse (OOA), den Entwurf (OOD) und für die Programmierung (OOP) ermöglichen einen durchgängigen Entwicklungsprozeß nach den Prinzipien der Objektorientierung. Elemente des Anwendungsbereichs werden als Objekte modelliert und zu Klassen zusammengefaßt. Klassen werden durch ihre passiven und aktiven Merkmale (Attribute und Methoden) beschrieben und zu Software-Bausteinen gruppiert.

Damit stehen Bausteine wie *Systeme*, *Komponenten*, *Subsysteme* und *Klassen* im Mittelpunkt. Diese werden nach dem Prinzip der Datenkapselung entworfen und stehen untereinander in Beziehungen wie Vererbung, Aggregation und Assoziation.

2.3.1.2 Hierarchischer Systemaufbau

Um größere Systeme besser strukturieren zu können, unterscheidet das EOS-Modell drei Ebenen der Systementwicklung:

(S) System-Ebene
(X) Komponenten- / Subsystem-Ebene
(K) Klassen-Ebene

Die drei Ebenen unterscheiden sich durch die unterschiedlichen Größenordnungen ihrer Gegenstandsbereiche. Das System bildet die oberste Ebene, die Komponenten bzw. Subsysteme die mittlere Ebene und die Klassen die unterste Ebene (Abb. 5).

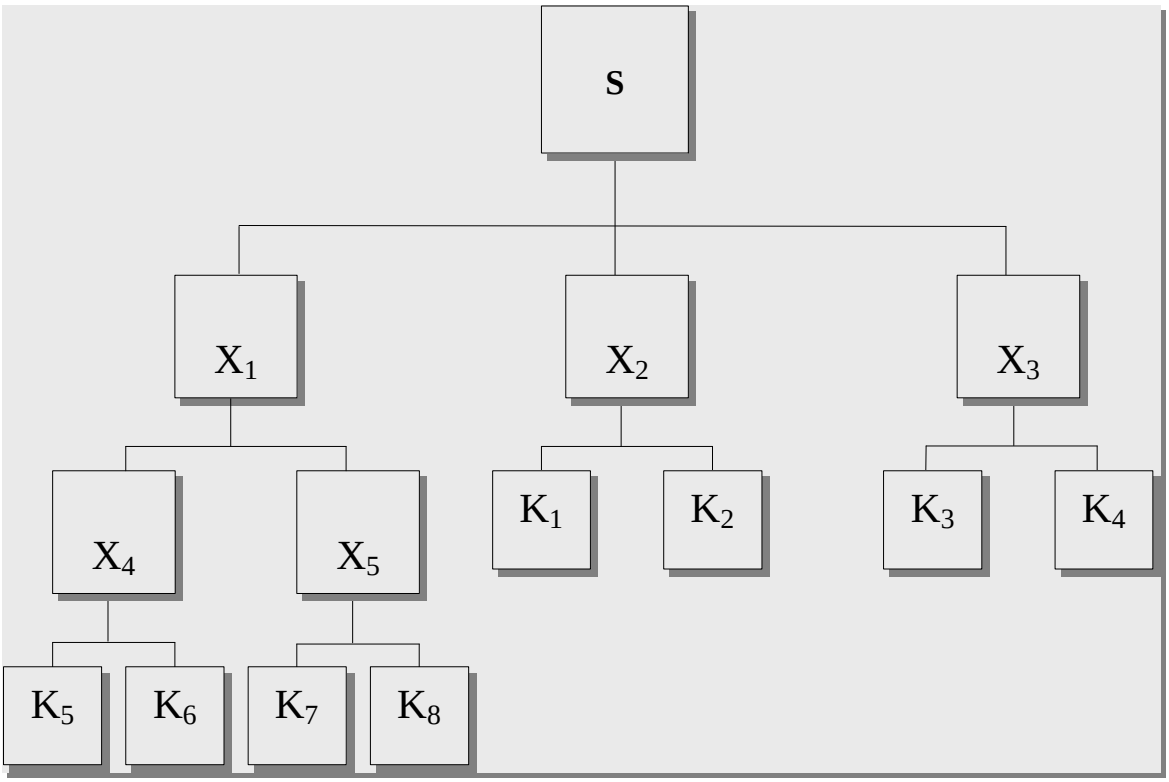


Abb. 5: Hierarchischer Systemaufbau

Komponenten fassen logisch oder organisatorisch zusammengehörige Klassen zusammen und dienen zur Unterstützung der Analyse-, Entwurfs- und Planungsprozesse. Komponenten sind zueinander disjunkt, d.h. die Klassen sind eindeutig den Komponenten zugeordnet. Anhand der Komponenten wird eine statische Zerlegung des Systems vorgenommen, was nicht mit der Benutzungsstruktur verwechselt werden darf. Denn die Klassen einer Komponente dürfen von Klassen anderer Komponenten benutzt werden.

Durch Komponenten können wiederverwendbare Systemteile gebündelt werden und in der Bausteinbibliothek abgelegt werden. Außerdem dienen sie als Planungsgrundlage zur Definition und Abwicklung von Arbeitspaketen.

Subsysteme sind eine nicht-disjunkte Zusammenfassung von Klassen, die gemeinsam zum Ablauf (im Sinne von Test oder Integration) gebracht werden sollen. Somit sind Subsysteme vor allem für die Phasen Implementierung und Operativer Einsatz wichtig.

2.3.1.3 Zyklische Entwicklung

An jedem Baustein werden die folgenden vier Tätigkeiten bzw. Tätigkeitsgruppen ausgeführt:

(A) Analyse
(E) Entwurf
(I) Implementierung
(O) Operationeller Einsatz

Bei der Analyse eines Bausteins werden die Anforderungen an diesen festgelegt und es wird ein Modell gebildet, das den Baustein im Zusammenhang mit seiner Umgebung darstellt. Außerdem wird die Bausteinbibliothek herangezogen, um zu prüfen, ob solch ein Baustein bereits vorhanden ist oder andere Bausteine genutzt werden können.

Ein Baustein wird entworfen, indem seine Schnittstellen spezifiziert werden und sein struktureller Aufbau konstruiert bzw. festgelegt wird.

Einen Baustein implementieren bedeutet, den Entwurf in Code umzusetzen, diesen zu testen und zu integrieren.

Implementierte Bausteine müssen eine Bewährungsprobe im operationellen Einsatz bestehen. Zum Einsatz gehören die Erprobung, die Nutzung und mögliche Revisionen eines Bausteines.

Eine Abfolge der vier Tätigkeiten wird als *Zyklus* bezeichnet (vgl. Abb. 6).

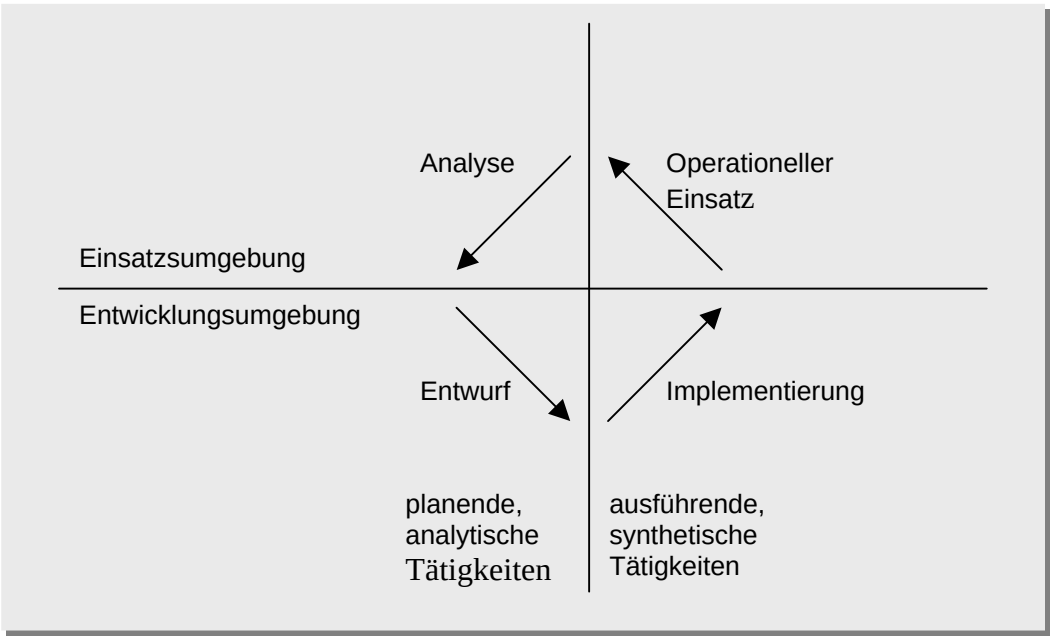


Abb. 6: Tätigkeiten eines Entwicklungssyklus /Hesse 1996/

Die Tätigkeiten laufen in der Regel vollständig ab, es sei denn, sie werden durch innere Zyklen unterbrochen. Zyklen und Tätigkeiten können miteinander gekoppelt werden. Die

Systementwicklung vollzieht sich als Zyklus auf der System-Ebene, von der aus Zyklen auf der Komponenten-Ebene angestoßen werden können. Auf der Komponenten-Ebene werden gegebenenfalls weitere Komponenten-Zyklen oder Klassen-Zyklen angestoßen.

Die zeitliche Abfolge der Entwicklungszyklen der einzelnen Bausteine wird nicht wie beim Wasserfall-Modell durch einen übergeordneten Phasenplan festgelegt, sondern je nach den Projekt-Erfordernissen geplant und gesteuert. Komponenten, Subsysteme und Klassen werden in eigenen Zyklen entwickelt und im Rahmen des jeweils übergeordneten Zyklus, wie z.B. des System-Zyklus, koordiniert. Diese zeitlich verzahnten Entwicklungszyklen werden in der Abbildung 7 vereinfachend dargestellt.

2.3.1.4 Einbezug von Erprobung, Nutzung und Revision in die Entwicklungszyklen

Ein wesentlicher Unterschied zwischen dem EOS-Verfahren und herkömmlichen Verfahren besteht darin, daß Software-Entwicklung nicht mehr als ein isolierter, auf das jeweilige Projekt begrenzter Prozeß betrachtet wird. Software-Entwicklung wird als ein langfristiger und kontinuierlicher Prozeß gesehen, so daß Aspekte wie Wartung, Pflege, Erprobung, Nutzung, Revision und Wiederverwendung auf allen Entwicklungsebenen mit berücksichtigt werden.

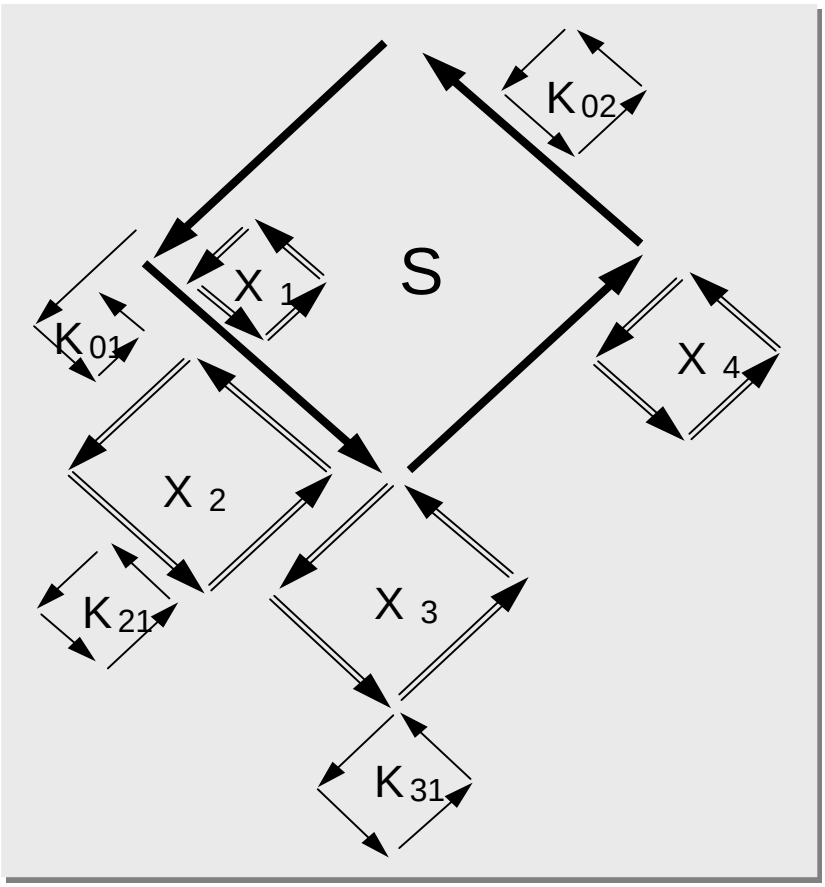


Abb. 7: Zeitlich verzahnte Tätigkeitszyklen /Hesse 1996/

2.3.1.5 Weiter- und Wiederverwendung von Software-Bausteinen

Ein wichtiges Ziel der objektorientierten Verfahren ist es, die Wiederverwendung zu fördern. Die Wiederverwendung wird im EOS-Modell durch die Entwicklungszyklen und mit dem

Konzept der Bausteinbibliothek unterstützt. Die Bausteinbibliothek faßt alle freigegebenen Bausteine zusammen und ermöglicht somit die Wieder- und Weiterverwendung (ggf. nach Modifikationen) von Bausteinen. Eine entscheidende Rolle spielt die Bausteinbibliothek in den beginnenden und abschließenden Tätigkeiten eines Zyklus. Während der Analyse wird die Bausteinbibliothek zur Suche und zum Auffinden von wiederverwendbaren Bausteinen benötigt. Ist ein Baustein bereits im operationellen Einsatz und sind die ersten erfolgreichen Nutzungsläufe erfolgreich durchlaufen, dann kann dieser Baustein in die Bausteinbibliothek eingefügt werden.

2.3.1.6 Mit der Entwicklungsmethodik eng abgestimmte Management-Verfahren

Im EOS-Modell findet die Projektplanung und -steuerung auf der Basis von Zyklen und Tätigkeiten statt. Die drei Entwicklungs-Ebenen bilden die Basis für Planungsschritte unterschiedlicher Detaillierung. Der System-Zyklus ermöglicht eine projektweite Grobplanung. Die Grobplanung wird ergänzt, indem die Komponenten-Zyklen zu feineren Planungsschritten für Team-Arbeitspakete herangezogen werden. Für die Planung der Tätigkeiten von einzelnen Mitarbeitern werden die Klassen-Zyklen verwendet. Detailplanungen können anhand der einzelnen Zyklusschritte vorgenommen werden. Damit kann das Management die Projektplanung flexibel an die jeweilige Projektsituation anpassen. Anstatt der klassischen Meilensteine an Phasengrenzen werden im EOS-Modell differenziertere Revisionspunkte betrachtet, um z.B. Reviews oder Plan-/Ist-Vergleiche vorzunehmen. Ein Beispiel für einen möglichen Revisionspunkt wäre: *Die X-Zyklen für die Komponenten A und B müssen abgeschlossen sein und die Implementierung für die Komponenten C und die Klasse D müssen vollzogen sein*. Revisionspunkte werden am Beginn des Projektes vorgesehen, aber noch nicht in allen Einzelheiten festgelegt. Der genaue Inhalt der Revisionspunkte wird erst mit fortschreitender Detailplanung bestimmt.

2.3.2 EOS-Subprozesse

Die Software-Entwicklung läßt sich als ein Zusammenspiel verschiedener Prozesse und Gruppen von beteiligten Personen beschreiben. Die in der Regel nebenläufigen Prozesse sind einzelnen Gruppen mit einer spezifischen Sicht auf das Projekt zugeordnet. Im EOS-Modell werden folgende Subprozesse unterschieden:

- Projektmanagement
- Entwicklung
- Qualitäts-Management
- Konfigurations-Management und Unterstützung
- Nutzung und Bewertung

Jeder Subprozeß gliedert sich in Aktivitätstypen, Produkttypen und Rollen. Die einzelnen Subprozesse werden im nächsten Kapitel detailliert dargestellt. Die Abbildung 8 faßt die Hauptaktivitäts- und Produkttypen der Subprozesse zusammen. Das *Projektmanagement* vollzieht sich in den Teilschritten „Projekt-Inititialisierung und Planung“, „Projekt-Steuerung“ und „Projekt-Abschluß“. Im ersten Schritt werden die Projektziele definiert, der Projektgegenstand wird abgegrenzt, Risiken werden analysiert und organisatorische Maßnahmen, wie z.B. die Planung der Wiederverwendung oder das Konfigurations-management, vorbereitet. Das Projekthandbuch faßt die Ergebnisse zusammen. Ziel der Projekt-Steuerung ist es, das Projekt erfolgreich durchzuführen. Dazu muß das Projekt kontinuierlich begleitet, Risiken müssen rechtzeitig identifiziert und Änderungswünsche koordiniert werden. Beim Projekt-Abschluß werden die gesammelten Erfahrungen in Form

eines Abschlußberichts systematisch aufbereitet, um sie für künftige Projekte nutzen zu können.

Der Gegenstand der *Software-Entwicklung* im EOS-Modell sind die Bausteine „System“, „Komponenten/Subsystme“ und „Klassen“. Die identifizierten Bausteine gilt es zunächst zu analysieren, d.h. die gestellten Anforderungen werden festlegt. Beim Entwurf werden die spezifizierten Bausteine konstruiert. Implementierung bedeutet, den Entwurf in Quellcode umzusetzen, zu testen und zu integrieren. Die Erprobung, Nutzung und Revision der Bausteine findet im Rahmen des operationellen Einsatzes statt.

Bei der *Qualitätssicherung* werden im ersten Schritt die zu prüfenden Produkttypen, wie z.B. Diagramme, Quellcode oder Dokumentation, definiert. Für diese Produkttypen, die im Rahmen eines Bausteins entstehen, werden entsprechende Prüfmaßnahmen gewählt. Zuletzt wird die Prüfung durchgeführt und anhand der Prüfergebnisse für eine Freigabe bzw. Überarbeitung entschieden. Aspekte wie Qualitätsziele und -merkmale werden im Rahmen des Qualitätsmanagements behandelt (siehe Abschnitt 2.3.2.1).

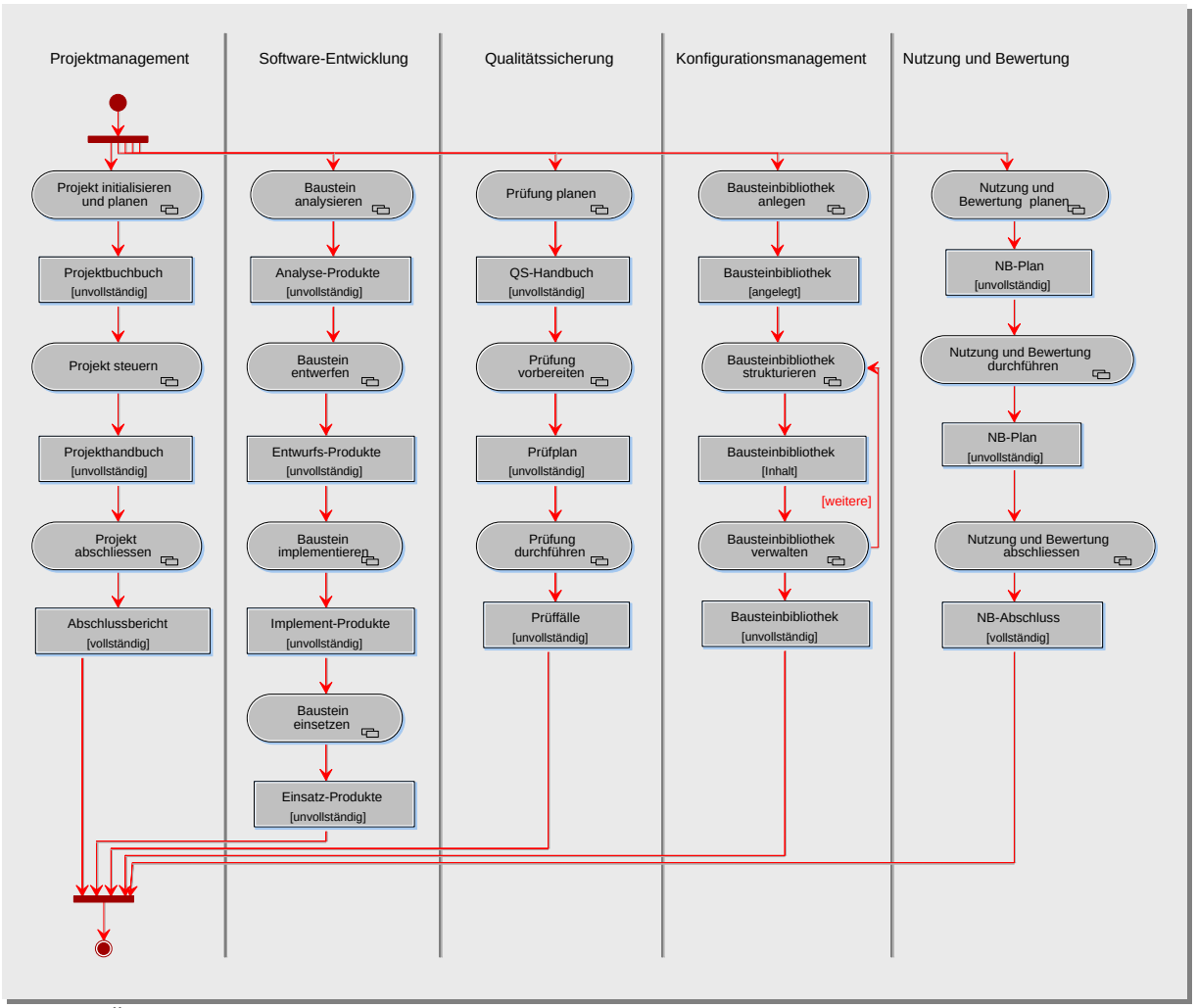


Abb. 8: Überblick über die Aktivitäts- und Produkttypen

Eine *Konfiguration* ist eine benannte und zusammengehörende Menge von Produkten. Konfigurationen werden benötigt, um Probleme bei den Änderungen und Erweiterungen von Produkten besser zu regeln. Das *Konfigurationsmanagement* sorgt dafür, daß die Projektmitglieder stets auf einem definierten Entwicklungs- oder Änderungsstand arbeiten

können. Dazu wird eine Bausteinbibliothek benötigt, die alle Produkte verwaltet, die im Kontext eines Bausteines entstehen. Die Bausteinbibliothek muß als erstes eingerichtet werden, z.B. müssen die Benutzer und Benutzerrechte angelegt werden. Danach werden die zu entwickelten Bausteine der Bausteinbibliothek zugeordnet. Pro Baustein werden eine Reihe von Verwaltungsinformationen, wie z.B. die Version oder Angaben zur Freigabe für Wiederverwendung, mitgeführt.

Im Rahmen des Subprozesses *Nutzung und Bewertung* werden die Anwender von Anfang bis zum Ende des Projekts begleitet und betreut. Zum Beispiel werden die Anwender im Subprozeß „Software-Entwicklung“ von der Anforderungsanalyse bis hin zur Abwicklung von Schulungen sinnvoll einbezogen. Die zyklische Entwicklung der Bausteine und das Revisionspunkt-Konzept sind hierbei sehr hilfreich. Beim Erreichen eines Revisionpunkts oder nach Abschluß eines Entwicklungs-Zyklus stehen Ergebnisse, wie z.B. neue Systemfunktionen, zur Verfügung. Solche Ergebnisse können den Anwendern präsentiert werden. Die Anwender-Bewertung sollte zur Verbesserung der Ergebnisse genutzt werden.

Die einzelnen Prozesse müssen kontinuierlich koordiniert und zu vordefinierten Zeitpunkten synchronisiert werden. Die Abfolge der koordinierenden Management-Tätigkeiten wird ebenfalls als Subprozeß betrachtet. Die Synchronisation verschiedener nebenläufiger Prozesse kann z.B. anhand der Revisionspunkte vorgenommen werden.

2.3.2.1 Schnittstellen von EOS

Neben den fünf genannten Prozessen, die ein einzelnes Projekt betreffen, gibt es projektübergreifende, i.a. im Unternehmen verankerte, Prozesse, zu denen das EOS-Modell eine Schnittstelle besitzt. Diese Schnittstellen werden in der nächsten Abbildung dargestellt.

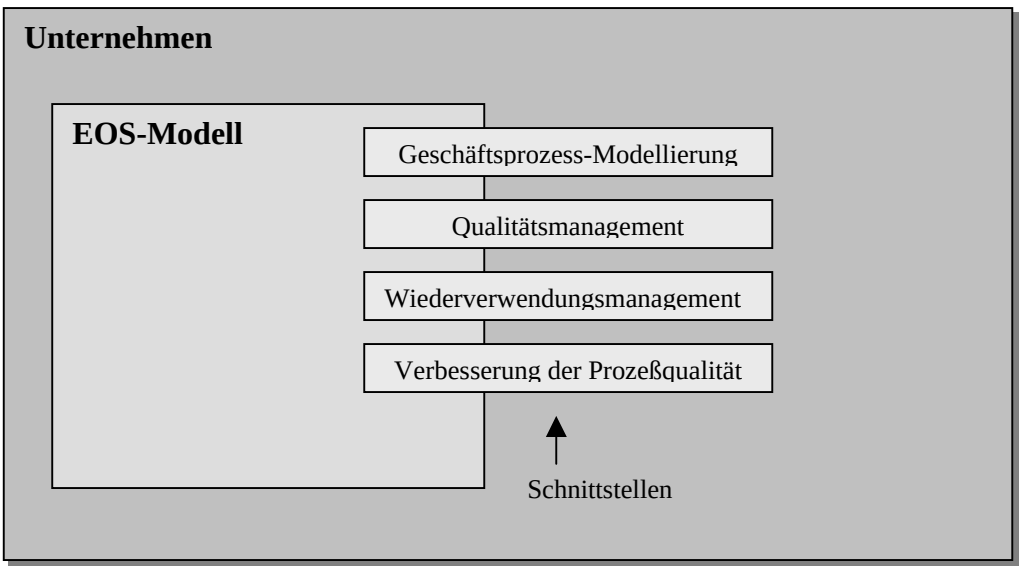


Abb. 9: Die Schnittstellen von EOS

Im Rahmen der *Geschäftsprozeß-Modellierung* werden die Geschäftsprozesse eines Unternehmens reorganisiert. Dabei wird zwar häufig die Notwendigkeit für neue Software oder für Änderungen an vorhandener Software erkannt, dennoch hat die Geschäftsprozeß-Modellierung einen anderen Schwerpunkt als die Software-Entwicklung. Deshalb wird bewußt die Modellierung von Geschäftsprozessen aus dem EOS-Modell ausgeklammert. Jedoch müssen die Ergebnisse der Geschäftsprozeß-Modellierung bei der Software-

Entwicklung berücksichtigt werden. So sind inhaltliche Aspekte, wie z.B. erkannte Schwachstellen oder Optimierungen, für eine möglichst effektive Entwicklung und Integration des neuen Anwendungssystems zu beachten. Ebenso kann aber das eingeführte Anwendungssystem neue Ansatzpunkte für die Geschäftsprozeß-Modellierung liefern. Die bei der Geschäftsprozeß-Modellierung analysierten Aktivitäten der zu automatisierenden Geschäftsprozesse können als Eingangsinformation für den Subprozeß „Software-Entwicklung“ (Anforderungsanalyse) dienen.

Da inzwischen in fast allen Lebensbereichen Software eingesetzt wird, hat ihre Qualität eine entscheidende Bedeutung erlangt. Software sollte fehlerfrei funktionieren. Doch das ist nur ein Qualitätsaspekt. *Qualitätsmanagement* legt projektübergreifende Qualitätsziele für ein Unternehmen fest und sorgt für ihre Umsetzung. Dazu muß ein Qualitätsmodell entwickelt werden, das die gesetzten Qualitätsziele operationalisiert. Das unternehmensweite Qualitätsmodell und die zugehörigen Verfahren zur Qualitätssicherung müssen bei dem Subprozeß „Qualitätssicherung“ berücksichtigt werden.

Wiederverwendung reduziert Kosten und Komplexität der Software-Entwicklung. Um Software wiederverwenden zu können, muß wiederverwendbare Software vorliegen. Das Ziel des *Wiederwendungsmanagements* ist es, eine systematische Wiederverwendung und Wiederverwendbarkeit in einem Unternehmen zu erreichen. Dazu ist eine weitreichende Reorganisation und Umdenken nötig. Im Subprozeß „Software-Entwicklung“ wird zwar die Wiederverwendung von Bausteinen berücksichtigt, auf lange Sicht ist aber nur eine projektübergreifende und systematische Wiederverwendung von Bausteinen bzw. eine Entwicklung zur Wiederverendung erfolgsversprechend.

Die Qualität eines Produkts wird wesentlich von der Qualität des Erstellungsprozesses beeinflußt. Daher wird im Rahmen von *Verbesserung der Prozeßqualität* der Software-Entwicklungsprozeß eines Unternehmens kontinuierlich verbessert, um in allen Projekten gleichmäßig hohe Ergebnisqualität in der vorgegebenen Zeit und im vorgegebenen Budget zu erreichen. Kein Prozeßmodell ist perfekt, das gilt auch für das EOS-Modell. Zur kontinuierlichen Verbesserung des EOS-Modells müssen die in den Projekten gewonnenen Erfahrungen zur systematischen Verbesserung der Prozeßqualität genutzt werden.

Das EOS-Modell ist im Gegensatz zu den bekannten Vorgehensmodellen zu objektorientierten Methoden ein durchgängig evolutionäres Prozeßmodell. Die Systementwicklung findet auf der Basis der Gliederungseinheiten System, Komponente und Klasse statt, die gemeinsam als Bausteine bezeichnet werden. Der Entwicklungsprozeß der einzelnen Bausteine ist jeweils als Zyklus mit den Tätigkeiten Analyse, Entwurf, Implementierung und operationellem Einsatz ausgelegt. Mit Hilfe der Zyklen und Tätigkeiten läßt sich die Entwicklung und Planung flexibler gestalten. Revisionspunkte ersetzen die klassischen Meilensteine und ermöglichen die zeitliche Gliederung und Verfolgung von Projekten. Das EOS-Modell gliedert sich in die Subprozesse „Software-Entwicklung“, „Projektmanagement“, „Qualitätssicherung“, „Konfigurationsmanagement“ und „Nutzung und Bewertung“. Es hat Schnittstellen zur „Geschäftsprozeß-Modellierung“, „Qualitätsmanagement“, „Verbesserung der Prozeßqualität“ und „Wiederverwendungs-management“.

3 Aktivitäts- und Produkttypen des EOS-Modells

In diesem Kapitel werden für die einzelnen Subprozesse des EOS-Modells die auszuführenden Aktivitätstypen und die erzeugten Produkttypen definiert. Diese sind ein möglicher EOS-Zuschnitt und bilden die Grundlage des Prozeßmanagement-Prototypen, der im vierten Kapitel vorgestellt wird.

☑ Fundierte Kenntnisse in Objektorientierung, UML und Softwaretechnik werden vorausgesetzt.

3 Aktivitäts- und Produkttypen des EOS-Modells

Im vierten Kapitel werden wir einen Prototypen für ein Prozeßmanagement-System (PMS) vorstellen. PMS ist ein Werkzeug, das das EOS-Modell durchgängig mit Software unterstützen soll. PMS baut auf einem Metamodell für das EOS-Verfahren auf, das in der nächsten Abbildung vorgestellt wird.

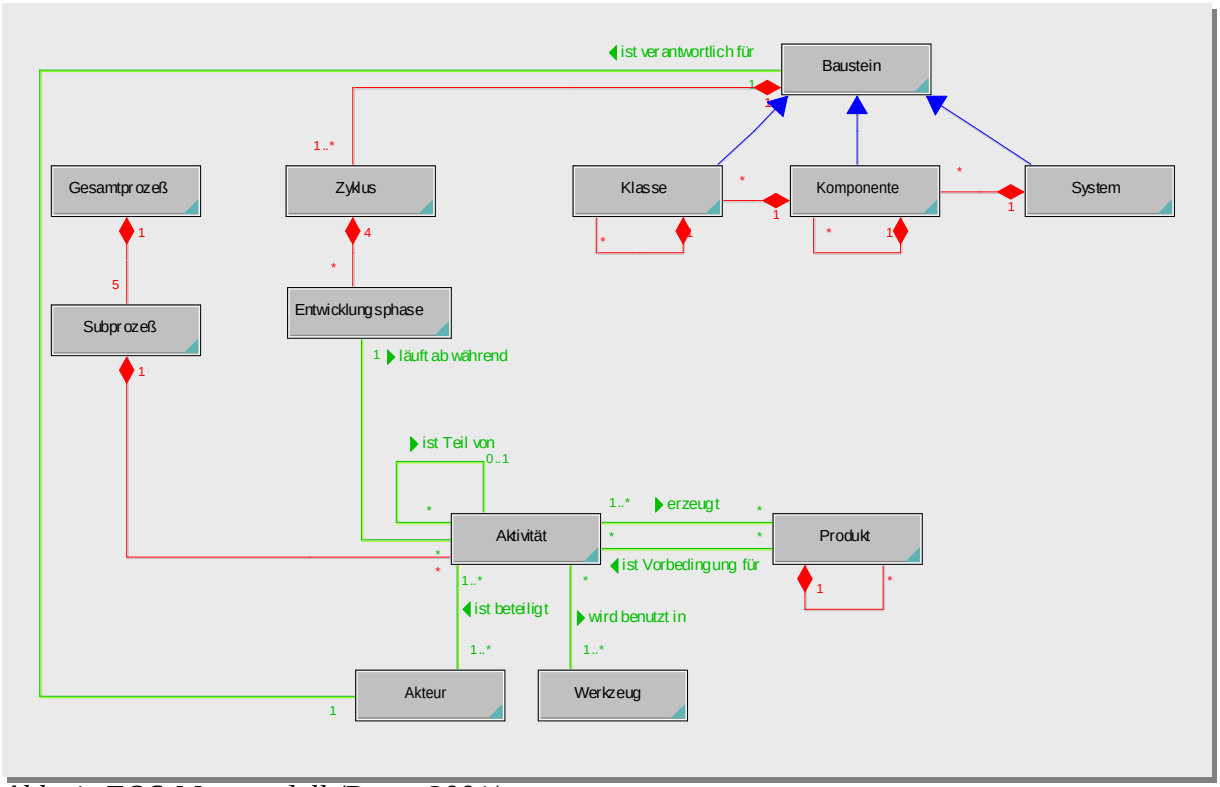


Abb. 1: EOS-Metamodell /Beyer 2001/

Neben Bausteinen bilden Aktivitäten den Kern des Metamodells. Aktivitäten beschreiben die an den Bausteinen durchzuführenden Tätigkeiten und definieren so die einzelnen Subprozesse. An einer Aktivität sind Akteure und Werkzeuge beteiligt, die dazu beitragen, daß ein Produkt erzeugt werden kann.

In PMS nutzen wir die Aktivitäts- und Produkttypen, um ein flexibles Prozeßmanagement-Werkzeug zu entwickeln. Daher beschreiben wir in den nächsten Abschnitten die Aktivitäts- und Produkttypen der einzelnen Subprozesse des EOS-Modells zusammenfassend. Hierbei handelt es sich um einen möglichen EOS-Zuschnitt. Denn das EOS-Modell ist in seiner ursprünglichen Fassung methodenfrei und läßt auch andere Methodendefinitionen zu. Detaillierte Darstellungen der fünf Subprozesse befinden sich im Anhang der Arbeit. Dort haben wir die Subprozesse in Form eines Prozeß-Handbuchs beschrieben, mit dem Anspruch, dieses Projektmitgliedern für den Einsatz in der Praxis zur Verfügung zu stellen. Im Handbuch werden die einzelnen Aktivitätstypen und die zugehörigen Methoden vertiefend behandelt, die Rollenzuweisungen werden geklärt und Beschreibungsschemata für die Produkttypen definiert. Zur Definition der Aktivitäts- und Produkttypen haben wir uns in erster Linie an dem Prozeßmodell für das auf UML basierende CASE-Werkzeug „objectiF“ /objectiF 4.5, microTOOL GmbH/ bzw. das V-Modell /www.v-modell-iabg.de/ orientiert, ziehen aber /Jacobson, Booch, Rumbaugh 1999/, /Kruchten 2000/, /Booch 1994/, /Jacobson

1993/, /Coad, Yourdon 1991/, /Rumbaugh 1991/, /Shlaer, Mellor 1991/, /Dröschel 1998/, /Larman 2001/, /Müller-Ettrich 1998/ und /Oestereich 1999/ hinzu.

3.1 Projektmanagement

Der erfolgreiche Abschluß eines Projekts hängt wesentlich von der Güte des Projektmanagements ab. Denn Fehlentscheidungen durch das Management sind in der Regel schwer zu korrigieren. Daher ist das Projektmanagement eine große Herausforderung und bringt zugleich eine große Verantwortung mit sich. Die Planung und Steuerung eines evolutionären, objektorientierten Projekts stellt noch weitere Anforderungen an das Projektmanagement, wie z.B. das Planen von Zyklen und Revisionspunkten. Kurz gesagt, Projektmanagement wird nicht leichter, berücksichtigt aber stärker die Dynamik eines Projekts.

Das Projektmanagement im EOS-Modell haben wir in die Teilabschnitte „Projekt-Initialisierung und Planung“, „Projekt-Steuerung“ und „Projekt-Abschluss“ gegliedert. Im folgenden behandeln wir diese Teilabschnitte in gekürzter Form. Detaillierte Ausführungen können dem Prozeß-Handbuch im Anhang entnommen werden.

3.1.1 Projekt-Initialisierung und Planung

Gegenstand der Projekt-Initialisierung und -Planung ist es, die Projektziele zu definieren, das Projekt abzugrenzen, Risiken zu analysieren und organisatorische Maßnahmen, wie z.B. die Planung der Wiederverwendung oder des Konfigurationsmanagements, vorzubereiten. Die Ergebnisse der Projekt-Initialisierung werden in einem zentralen Dokument, dem Projekthandbuch, zusammengefaßt. Den Prozeß der Projekt-Initialisierung und -Planung haben wir wie folgt definiert (Abb. 2):

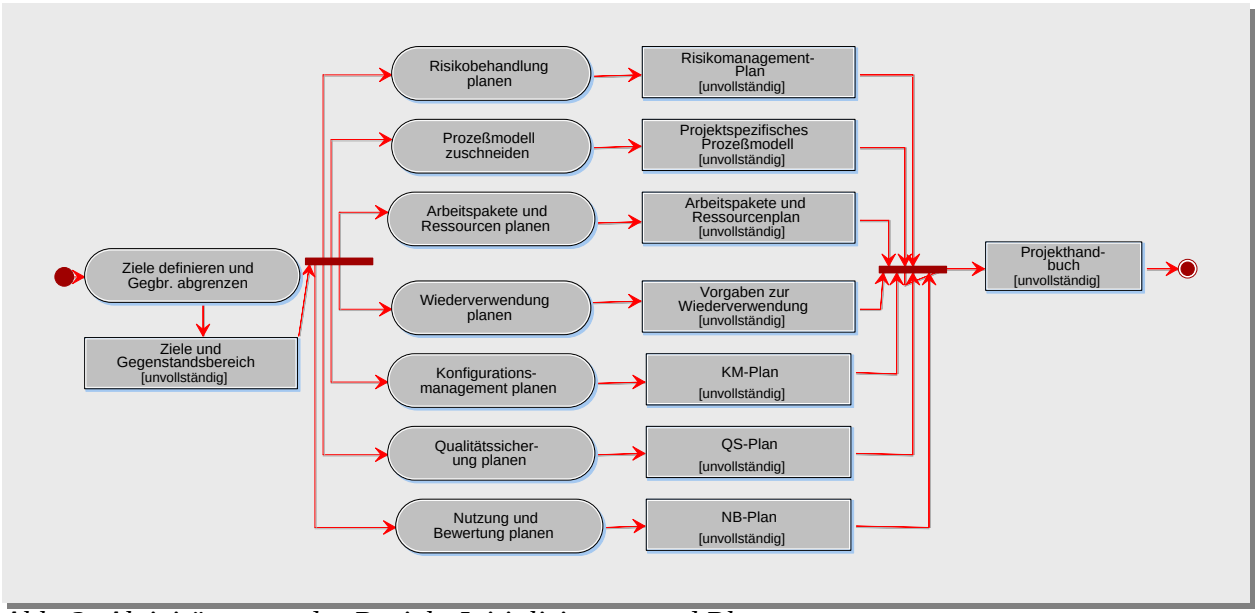


Abb. 2: Aktivitätstypen der Projekt-Initialisierung und Planung

Unter allen Projektbeteiligten muß ein einheitliches Verständnis dafür geschaffen werden, was mit der geplanten Entwicklung beabsichtigt wird. Hierzu sind folgende Fragen zu klären: Aus welchem Grund wird das Anwendungssystem entwickelt? Welche Ziele werden mit dem zu entwickelnden Anwendungssystem verfolgt? Neben der Zieldefinition muß der

Gegenstandsbereich des Projekts abgegrenzt werden. Eine detaillierte Lösung für die erkannten Probleme braucht man nicht anzugeben. Jedoch sollten ein grundsätzlicher Lösungsansatz und Vorstellungen über eine mögliche Grobarchitektur vorhanden sein, um daran die Planungsschritte des Projektmanagements zu orientieren.
Alles, was den Erfolg des Projekts gefährdet, ist ein Risiko, dem präventiv zu begegnen ist. Da alle Projekte Risiken bergen, ist es wichtig, sich dieser Risiken bewußt zu sein, um sie soweit wie möglich zu vermeiden. Das Ziel des Aktivitätstyps „Risikobehandlung planen“ ist es, Risiken zu identifizieren und zu bewerten, damit entsprechende Abwehrmaßnahmen ermittelt und eingeleitet werden können.

Das EOS-Modell besitzt eine hohe Allgemeingültigkeit und muß auf die Gegebenheiten eines Projekts angepaßt werden. Das bedeutet, daß die Aktivitätstypen, Produkttypen und das Rollenmodell, bezogen auf das jeweilige Projekt, modifiziert und ergänzt werden müssen. Dieser Prozeß wird auch als *Tailoring* bzw. *Zuschneiden* bezeichnet, der im Rahmen des Aktivitätstyps „Prozessmodell zuschneiden“ durchgeführt wird. Das Ergebnis ist ein spezialisiertes Prozeßmodell, das nur solche Rollen, Aktivitätstypen und Produkttypen vorsieht, die für das konkrete Projekt relevant sind.

Der Aktivitätstyp „Arbeitspakete und Ressourcen planen“ dient der Definition von Arbeitspaketen. Für die Arbeitspakete muß der Aufwand geschätzt werden und die benötigten Ressourcen, wie z.B. Personal oder Arbeitsmittel, festgelegt werden. Zunächst wird eine Grobplanung für das System vorgenommen, in der die Arbeitspakete im Projektteam verteilt werden und die Termine und die benötigten Arbeitsmittel festgelegt werden. Diese Grobplanung wird schrittweise durch Feinplanungen für Komponenten und Klassen detailliert. Die Planung wird durch das von uns vorgeschlagene Aufwandschätzverfahren CEOS unterstützt.

Das Ziel des Aktivitätstyps „Wiederverwendung planen“ ist es, Vorgaben zu machen, welche vorhandenen Ergebnisse wieder verwendet und welche Produkte für eine Wiederverwendung in nachfolgenden Zyklen oder Projekten entwickelt werden sollen. Sowohl die Wiederverwendung als auch die Entwicklung zum Zweck der Wiederverwendung kosten Zeit und Geld, so daß sie bei der Projektplanung mit berücksichtigt werden müssen. Wiederverwendung kann zwar Zeit und Geld sparen, ist aber nicht umsonst zu bekommen. Denn wiederverwendete Bausteine müssen gefunden und analysiert werden. Das Entwickeln von wieder zu verwendenden Bausteinen stellt hohe Ansprüche an Entwurf, Dokumentation und Qualitätssicherung. Kurzfristig werden so höhere Kosten verursacht als bei einer Speziallösung für ein Projekt.

Zu einem Baustein gehören neben dem produzierten Quellcode auch alle sonstigen Ergebnisse, die im Laufe der Baustein-Entwicklung entstehen. Zum Baustein „System“ gehören z.B. Produkte wie Ressourcenplan, Entwicklungsstrategie oder Komponenten-Diagramme. Diese Produkte, die verschiedene Versionen haben können, werden zu einer Konfiguration zusammengefaßt. In großen Projekten mit mehreren tausend Bausteinen stellt das Konfigurationsmanagement eine logistische Aufgabe dar, die ohne Werkzeugunterstützung nicht zu bewältigen ist. Daher werden im Rahmen des Aktivitätstyps „Konfigurationsmanagement planen“ organisatorische Aspekte, wie z.B. die Regelung von Namenskonventionen, behandelt.

Im Rahmen des Aktivitätstyps „Qualitätssicherung planen“ werden projektbezogene Regelungen für die Qualitätssicherung getroffen. Es wird festgelegt, welche Produkttypen einer Qualitätssicherung unterzogen werden müssen (*Was wird Qualitäts-gesichert?*). Die

Qualitätsziele, -Merkmale und -Vorgaben werden präzisiert und gewichtet. Außerdem werden die anzuwendenden Verfahren zur Qualitätssicherung bestimmt (*Wie wird gesichert?*). Hierbei wird zwischen *konstruktiver* und *analytischer* Qualitätssicherung unterschieden. Die geplanten Maßnahmen müssen in den Entwicklungsprozeß eingegliedert werden (*Wann wird gesichert?*). Analytische Maßnahmen können z.B. bei Erreichen eines Revisionspunktes (siehe Projekt-Steuerung) oder nach Abschluß einzelner Schritte eines Baustein-Zyklus ausgeführt werden. Zuletzt muß noch definiert werden, wer für die Durchführung der Maßnahmen verantwortlich ist (*Wer sichert?*).

Im Rahmen des Subprozesses „Nutzung und Bewertung“ wird der Nutzer von Anfang bis zum Ende des Projekts begleitet und betreut. Beispielsweise werden bei der Software-Entwicklung die Anwender von der Anforderungsanalyse bis hin zur Abwicklung von Schulungen stets einbezogen. Für den Erfolg des Projekts sind u.a. regelmäßige Workshops und Sitzungen mit dem Auftraggeber bzw. Anwender entscheidend. Solche Projektsitzungen müssen geplant werden (Aktivitätstyp „Nutzung und Bewertung planen“).

3.1.2 Projekt-Steuerung

Das primäre Ziel des Projektmanagements ist es, das Projekt erfolgreich durchzuführen. Das bedeutet, die Erwartungen des Auftraggebers zu erfüllen und das gewünschte Produkt kosten- und zeitgerecht zu liefern. Zu diesem Zweck muß das Projekt kontinuierlich begleitet und der Entwicklungsstand mit Hilfe der Revisionspunkte synchronisiert werden. Risiken müssen rechtzeitig identifiziert und Änderungswünsche systematisch koordiniert werden. Abbildung 3 faßt die von uns definierten Aktivitätstypen für die Projekt-Steuerung zusammen.

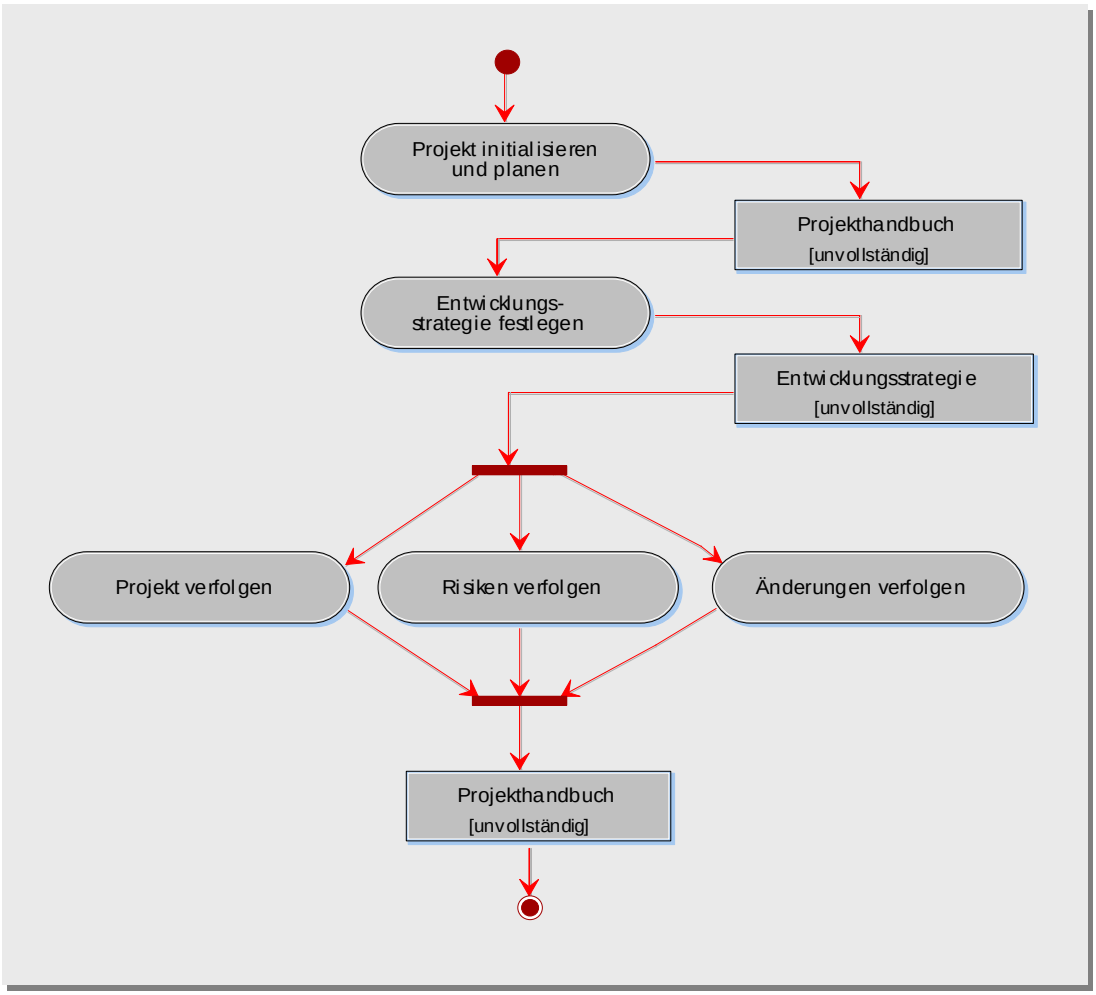


Abb. 3: Aktivitätstypen der Projekt-Steuerung

Die Grundlage für die Projekt-Steuerung bildet die Projekt-Initialisierung und Planung. Bausteine sind die Hauptgegenstände der Projekt-Steuerung. Ihre fortschreitende Entwicklung gilt es ständig zu begleiten und zu koordinieren. Bei jeder der vier Phasen wird die Planung anhand der beobachteten Bausteinzustände revidiert. Die Planung in den verschiedenen Detaillierungsstufen System, Komponente und Klasse muß an veränderte Rahmenbedingungen angepaßt werden. Zur Synchronisation und Steuerung von Zyklen und Bausteinen werden weitere Revisionspunkte geschätzt bzw. bestehende verfeinert.

Der Aktivitätstyp „Entwicklungsstrategie festlegen“ dient zur Identifikation und Definition von EOS-Elementen, wie z.B. ersten Bausteinen, Zyklen oder Revisionspunkten. Der Entwicklungszyklus auf der Systemebene ist bereits angestoßen, so daß zumindest die Kernanforderungen beschrieben worden sind (z.B. durch Anwendungsfälle oder Tabellen). Sicherlich sind auch schon Bausteine aus den funktionalen bzw. nicht-funktionalen Anforderungen abgeleitet worden. Die Anforderungen dienen hierbei als ein möglicher Ausgangspunkt zur Identifikation von Bausteinen (keine 1:1 Beziehung zwischen Anforderungen und Bausteine). Mit der Unterstützung der Projektleitung wird versucht, zusätzliche Bausteine zu den bereits gefundenen zu identifizieren, z.B. durch Auf- oder Abspaltung von Bausteinen oder durch das Erkennen wiederverwendbarer Bausteine. Die gefundenen Bausteine werden genutzt, um die Grobarchitektur des Anwendungssystems weiter zu erarbeiten und zu überdenken.

Nach der Identifikation von Bausteinen sind Entwicklungszyklen zu planen. Wir betrachten hierzu die funktionalen bzw. nicht-funktionalen Anforderungen, die durch Bausteine realisiert werden. Je höher die Priorität der Anforderung, desto höher ist auch die Priorität der Bausteine, die die Anforderung realisieren. Aus den Prioritäten der Bausteine und ihren gegenseitigen Abhängigkeiten ist eine Realisierungsreihenfolge für die Bausteine zu entwickeln. Aus der Realisierungsreihenfolge sind Entwicklungszyklen abzuleiten, die jeweils eine Ausbaustufe des Bausteins darstellen. Bei der Planung von Entwicklungszyklen sind stets die Projektgröße, die Komplexität der Projektaufgaben, die Stabilität der vorliegenden Anforderungen und die verfügbaren Ressourcen mit zu berücksichtigen. Die Planung von Zyklen muß stets den Bausteinen angepaßt werden. Am Anfang wird man sicherlich nur die Zyklen für das System und möglicherweise für einige wenige Komponenten planen können. Im späteren Projektverlauf wird die Planung von weiteren Komponenten- und Klassen-Zyklen hinzukommen.

Zur Projektverfolgung werden zu Projektbeginn Revisionspunkte vorgesehen, die inhaltlich erst mit fortschreitender Detailplanung festgelegt werden. Revisionspunkte werden auf der Basis von Bausteinen, Zyklen und Tätigkeiten definiert. Ein Revisionspunkt ist mit einem Termin und einem inhaltlichen Ergebnis verknüpft. Die Ergebnisse sind durch die Phasen „Analyse“, „Entwurf“, „Implementierung“ und „operationeller Einsatz“, die an jedem Baustein durchgeführt werden, inhaltlich definiert. Wie ist es aber mit dem terminlichen Aspekt? Wann ist ein Revisionspunkt erreicht? Um diese Fragen im voraus beantworten zu können, muß man den Entwicklungsaufwand für Bausteine und ihre Zyklen abschätzen können. Hierzu kann wieder das CEOS-Verfahren zur Schätzung des Entwicklungsaufwands genutzt werden. Die Grundidee des CEOS-Verfahrens ist es, für Bausteine Komplexitätswerte zu berechnen, aus denen mit Hilfe statistischer Techniken der Aufwand geschätzt wird. Für einen beliebigen Baustein kann der Aufwand für die einzelnen Phasen und Zyklen geschätzt werden. Wie das Verfahren im Detail funktioniert, wird in Kapitel 4 erklärt. Für die Planung der Revisionspunkte ist das Ergebnis des Schätzverfahrens interessant. Denn für einen

beliebigen Baustein kann der Aufwand bis zu einem definierten Zeitpunkt, wie z.B. „bis einschließlich Entwurf“, geschätzt werden, so daß Revisionspunkte leicht definiert werden können. Dazu werden die geschätzten Aufwände der einzelnen Entwicklungsphasen als Balkendiagramme dargestellt und die Revisionspunkte rückwärts definiert. Im Kapitel 4 wird gezeigt, wie Revisionspunkte softwaregestützt definiert werden.

Das Ziel des Aktivitätstyps „Projekt verfolgen“ ist es, das Projekt kontinuierlich zu beobachten, zu geeigneten Zeitpunkten - nämlich beim Erreichen eines Revisionspunktes - zu bewerten und gegebenenfalls notwendige Korrektur-Maßnahmen zu ergreifen. Anhand der Bausteine und der entsprechenden Zyklen und Tätigkeiten kann der Projektmanager das gesamte Projekt systematisch verfolgen. Hierzu sollten Fragen geklärt werden, wie z.B. *In welcher Phase befindet sich der Baustein? In welchem Entwicklungszyklus befindet sich der Baustein? Bei welchen Bausteinen könnten Schwierigkeiten oder unvorhergesehene Risiken auftreten?*

Für die Bewertung des Projektsstands werden zuverlässige Angaben über geleistete Tätigkeiten, abgeschlossene Entwicklungsphasen, fertiggestellte Bausteine oder Ergebnisse von Reviews, Inspektionen und Revisionspunkt-Prüfungen benötigt. Dabei reicht die formale Vollzugsmeldung nicht aus. Die geforderte Qualität muß erbracht werden. Für den Plan/Ist-Vergleich ist es notwendig, die entstandenen Kosten zu ermitteln. Dazu müssen für jeden Baustein die bis zum jetzigen Zeitpunkt verbrauchten Zeit- und Budget-Aufwände berechnet und summiert werden. Um entscheiden zu können, ob Maßnahmen zur Planrevision notwendig sind, muß der gegenwärtige Projektstand mit der Planung verglichen werden. In der Regel werden Termine und Kosten der Bausteine mit den Planwerten verglichen. Übersteigen diese die geplanten Werte, so sind entsprechende korrigierende Maßnahmen zu veranlassen.

Im Rahmen des Aktivitätstyps „Risiken verfolgen“ werden mögliche Risiken im laufenden Projekt identifiziert, untersucht und Maßnahmen zu ihrer Abwehr eingeleitet. Die Grundlage für diesen Aktivitätstyp bildet der Risikoplan, der während der Projekt-Initialisierung und Planung erstellt wurde. Der aktuelle Projektstand wird hinsichtlich der eingetretenen Risiken und des Erfolgs der Abwehrmaßnahmen analysiert. Zu beantwortende Fragen sind: *Waren die eingeleiteten Abwehrmaßnahmen für die erkannten Risiken erfolgreich? Welche Risiken konnten erfolgreich abgewehrt werden? Sind neue Risiken aufgetreten?*

Für neu identifizierte Risiken müssen die geplanten Abwehrmaßnahmen eingeleitet und koordiniert werden. Falls ein Risiko trotz Abwehrmaßnahmen eingetreten ist, sollte alles unternommen werden, um den entstehenden Schaden zu minimieren. Hierzu sind die Maßnahmen zur Schadensbegrenzung einzuleiten, die im Risikoplan definiert worden sind. Wenn die definierten Maßnahmen nicht ausreichen, so müssen zusätzliche Abwehrmaßnahmen ermittelt und eingeleitet werden. Die Ursachen des eingetretenen Risikos werden analysiert, um daraus verbesserte Abwehrmaßnahmen abzuleiten und den Projektplan zu revidieren.

Mit der evolutionären Software-Entwicklung steht dem Anwender schon sehr früh eine erste Ausbaustufe des Anwendungssystems zur Verfügung. Sobald z.B. das Anwendungssystem genutzt wird, treten Fehler und Änderungswünsche auf. Gewisse Fehlermeldungen und Änderungswünsche können im Rahmen des operationellen Einsatzes berücksichtigt und abgearbeitet werden, andere werden erst im nächsten Entwicklungszyklus behandelt. Das Ziel des Aktivitätstyps „Änderungen verfolgen“ ist es, Änderungsanforderungen systematisch aufzunehmen, zu behandeln und gegebenenfalls den Projektplan zu revidieren.

3.1.3 Projekt-Abschluß

Ein Projekt hat nicht nur einen formalen Beginn, sondern auch einen formalen Abschluß. Dieser Vorgang dient u.a. dazu, die im Projekt gesammelten Erfahrungen zur systematischen Verbesserung des Software-Entwicklungsprozesses zu nutzen. Außerdem wird das Projekt gedanklich abgeschlossen, um an neue Aufgaben heranzugehen. Die nächste Abbildung faßt die von uns definierten Aktivitätstypen zusammen.

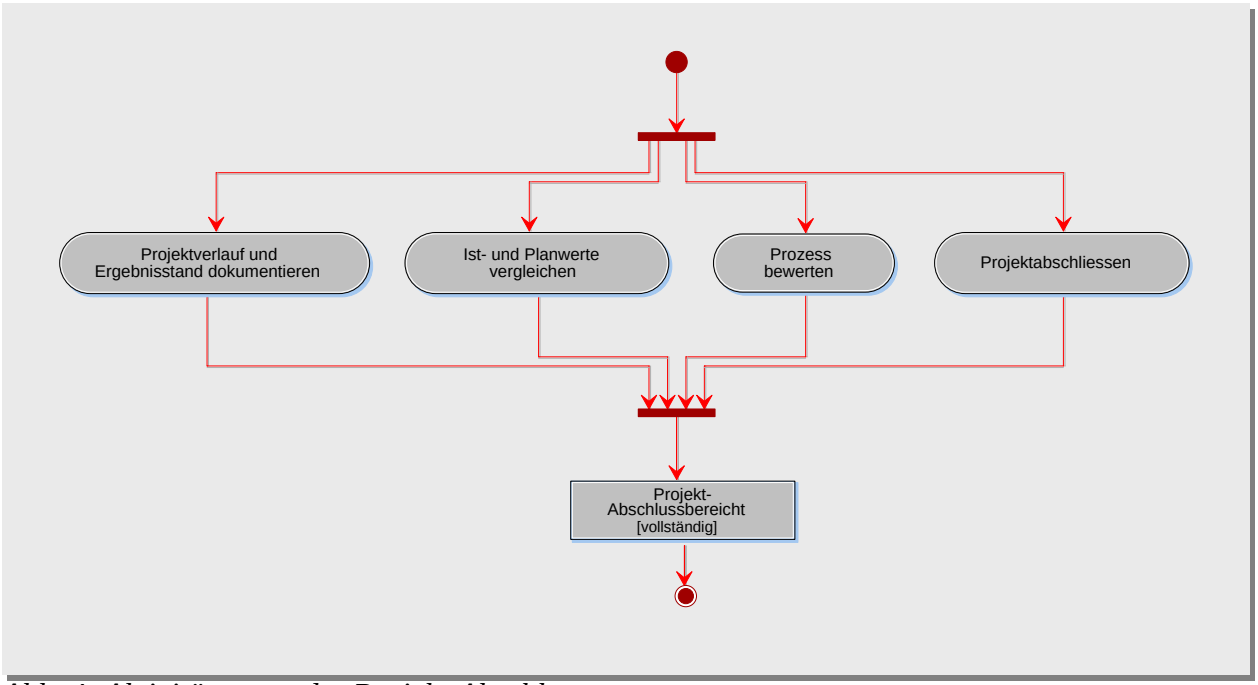


Abb. 4: Aktivitätstypen des Projekt-Abschlusses

Bei dem Aktivitätstyp „Projektverlauf und Ergebnisstand dokumentieren“ werden der Ablauf und die erzielten Ergebnisse des Projekts beschrieben. Dabei sollte insbesondere auf die Veränderungen eingegangen werden, die sich im Projektverlauf hinsichtlich der Aufgabenstellung und der technischen bzw. wirtschaftlichen Aspekte ergeben haben. Welche Schwierigkeiten gab es? Welche Ursachen hatten diese und wie wurden sie bewältigt? Welche positiven bzw. negativen Erfahrungen wurden gemacht?

Damit eine kontinuierliche Anpassung des CEOS-Verfahrens an die projekt- und unternehmensspezifischen Gegebenheiten möglich ist, sollten die tatsächlich erbrachten Aufwände für die einzelnen Bausteine erfaßt werden. Die Implementierung des CEOS-Verfahrens stellt entsprechende Benutzerfunktionen zur Daten-Erfassung und -Analyse an.

Im Rahmen der Aktivitätstypen „Ist- und Planwerte vergleichen“, „Prozeß bewerten“ und „Projekt abschließen“ werden Ist-Werte der Planung gegenübergestellt, um Abweichungen zu identifizieren. Änderungen, wie z.B. Termin-Verzug oder Budget-Überschreitung, werden auf ihre Ursachen hin analysiert. Der Umgang mit solchen Änderungen oder Problemen wird für künftige Projekte dokumentiert. Der geplante Prozeß wird mit dem tatsächlich durchgeführten verglichen. Abweichungen werden nach Ursachen, wie z.B. zu wenige Zyklen, analysiert. Alle relevanten Projektergebnisse und -unterlagen werden archiviert. Die Projektorganisation wird aufgelöst und die Mitarbeiter von ihren Projektpflichten entbunden.

3.2 Software-Entwicklung

Im EOS-Modell wird zwischen den drei Entwicklungsebenen System, Komponente/Subsystem und Klasse unterschieden. Jeder Baustein durchläuft – unabhängig von der ihm zugeordneten Ebene - die Phasen Analyse, Entwurf, Implementierung und operationeller Einsatz. Im folgenden definieren wir für die einzelnen Bausteine und ihre entsprechenden Phasen den Prozeß der Software-Entwicklung. Detaillierte Ausführungen können im Prozeß-Handbuch nachgelesen werden (vgl. Anhang A).

3.2.1 System-Analyse

Das Ziel der Analyse-Phase ist es, die Wünsche und die Bedürfnisse der Anwender an das zu entwickelnde System zu identifizieren und in einer möglichst einfachen Form zu dokumentieren. Die definierten Aktivitätstypen und die entsprechenden Produkttypen dieser Phase haben wir in der nächsten Abbildung zusammengefaßt.

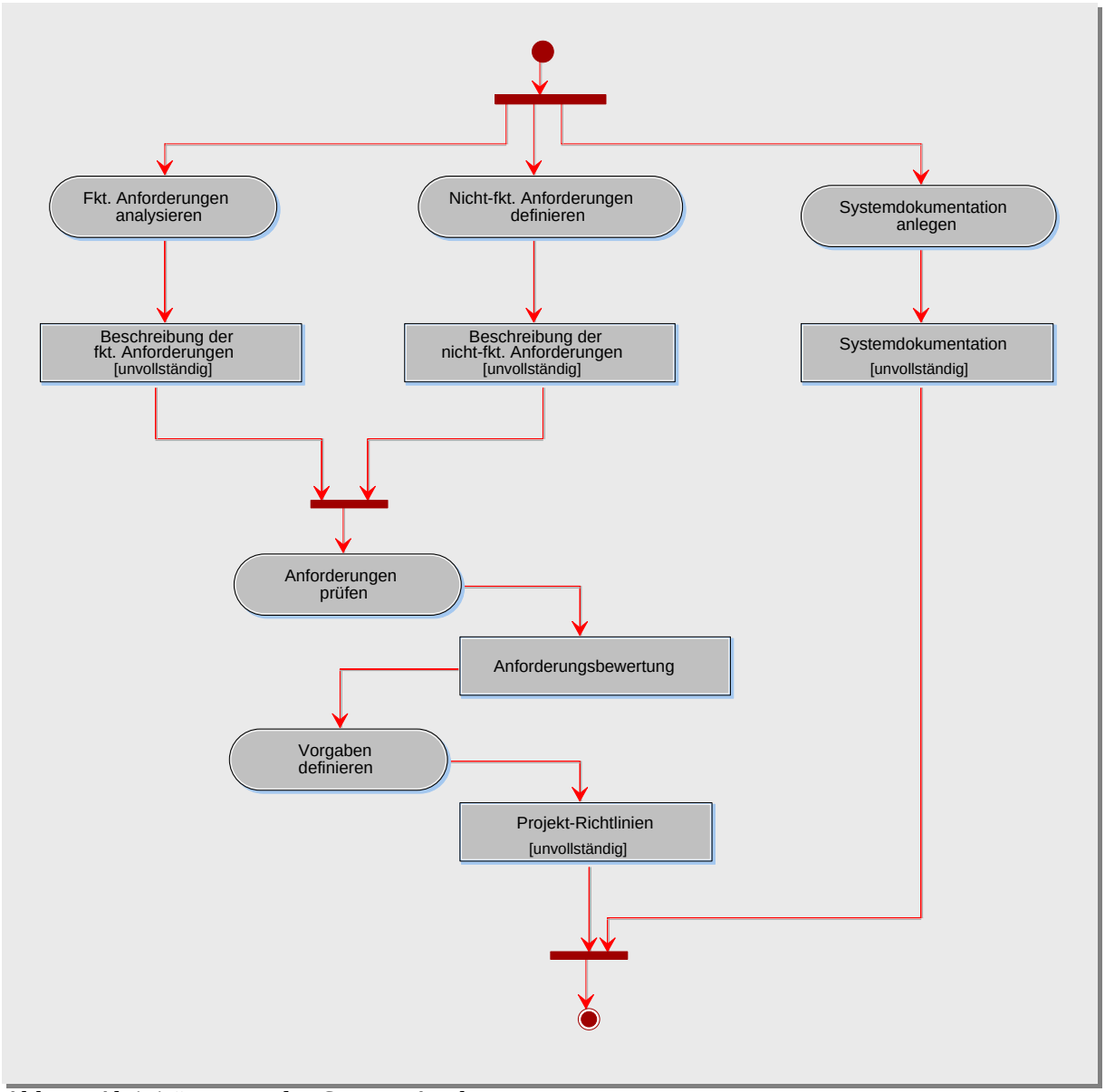


Abb. 5: Aktivitätstypen der System-Analyse

Der Aktivitätstyp „Funktionale Anforderungen analysieren“ dient dazu, eine klare, möglichst vollständige, konsistente und überprüfbare Definition der Anforderungen zu liefern, die an das Anwendungssystem gestellt werden. Die definierten Anforderungen müssen in einer verständlichen Form dokumentiert werden, damit eine gute Kommunikation mit dem künftigen Anwender möglich ist. Denn wenn Anwender bzw. Auftraggeber die Dokumentation der Anforderungen nicht verstehen, werden sie sich daran nicht gebunden fühlen. Daraus resultiert ein potenzieller Risikofaktor hinsichtlich des Projekterfolges. Eine mögliche Form der Dokumentation, die die obige Bedingung erfüllt, ist der Einsatz von Anwendungsfalldiagrammen. Die in einem Anwendungsfalldiagramm enthaltenen Anwendungsfälle repräsentieren die funktionalen Anforderungen. Neben der Beschreibung der Systemfunktionalität auf hoher Ebene können die Anwendungsfälle zur Definition von Entwicklungs- und Testeinheiten genutzt werden. Die Anwendungsfälle können auch verwendet werden, um den aktuellen Projektstatus zu ermitteln, indem geprüft wird, welche Anwendungsfälle schon umgesetzt worden sind und welche noch zu realisieren sind. Die Anwendungsfall-Modellierung ist eine weit verbreitete und bewährte Technik. Es gibt aber noch weitere Techniken zur Anforderungsanalyse, die in der Softwaretechnik im Rahmen des *Requirements-Engineering* behandelt werden.

Alle Anforderungen, die sich nicht auf die Funktionalität des zu realisierenden Anwendungssystems beziehen, werden in dem Schritt „Nicht-funktionale Anforderungen ermitteln“ analysiert. Zu den nicht-funktionalen Anforderungen eines Anwendungssystems zählen z.B. Vorgaben für den Erfüllungsgrad von Qualitätsmerkmalen oder Leistungsanforderungen, Hard- und Software-Restriktionen, Vorgaben an die Software-Architektur, Vorgaben zur Wiederverwendung oder Kauf von Fertigsoftware oder Vorgaben zu Software-Ergonomie.

Die funktionalen und nicht-funktionalen Anforderungen werden im Rahmen des Aktivitätstyps „Anforderungen prüfen“ hinsichtlich ihrer Erfüllbarkeit und Dringlichkeit mit dem Anwender bzw. Auftraggeber bewertet, um Mißerfolge und deren mögliche Konsequenzen zu vermeiden. Denn in der Regel können nicht alle Anforderungen im Rahmen des Zeit- bzw. Geld-Budgets realisiert werden. Daher ist es die Aufgabe des Projektmanagers, mit dem Anwender bzw. Auftraggeber einen Konsens bezüglich der Wirtschaftlichkeit und Realisierbarkeit der Anforderungen zu finden.

Damit eine einheitliche und leicht verständliche Lösung erarbeitet werden kann, werden verbindliche Vorgaben, wie z.B. Richtlinien für die Benutzeroberfläche, festgelegt (Aktivitätstyp „Vorgaben definieren“). Einheitliche Standards erleichtern auch eine schnelle Einarbeitung von später hinzukommenden Teammitgliedern.

Parallel zu den Aktivitätstypen der Analyse-Phase wird eine Systemdokumentation angelegt, in der die ermittelten Ergebnisse systematisch zusammengefaßt werden (Aktivitätstyp „Systemdokumentation anlegen“). Als praxisrelevant hat sich z.B. der Standard „IEEE STD 830-1993“ bewährt, so daß es sich empfiehlt, sich an diesem Standard zu orientieren (siehe Beschreibungsschema „Systemdokumentation“ im Prozeß-Handbuch).

3.2.2 System-Entwurf

Ziel der Entwurfs-Phase ist es, eine fachliche Lösung für das geplante Anwendungssystem zu entwickeln. Ausgangspunkt sind die in der Analyse-Phase ermittelten funktionalen und nicht-funktionalen Anforderungen. Diese werden zur Definition von Komponenten genutzt, um das System zu strukturieren. Weiterhin muß die Zielumgebung spezifiziert werden. Deren genaue

Kenntnis ist eine weitere notwendige Voraussetzung für die Systemarchitektur ist. Ausgehend von der Zielumgebung und der definierten Integrationsstrategie wird die Systemarchitektur entworfen. Parallel zu den genannten Aktivitätstypen muß die Systemdokumentation weiter fortgeschrieben werden.

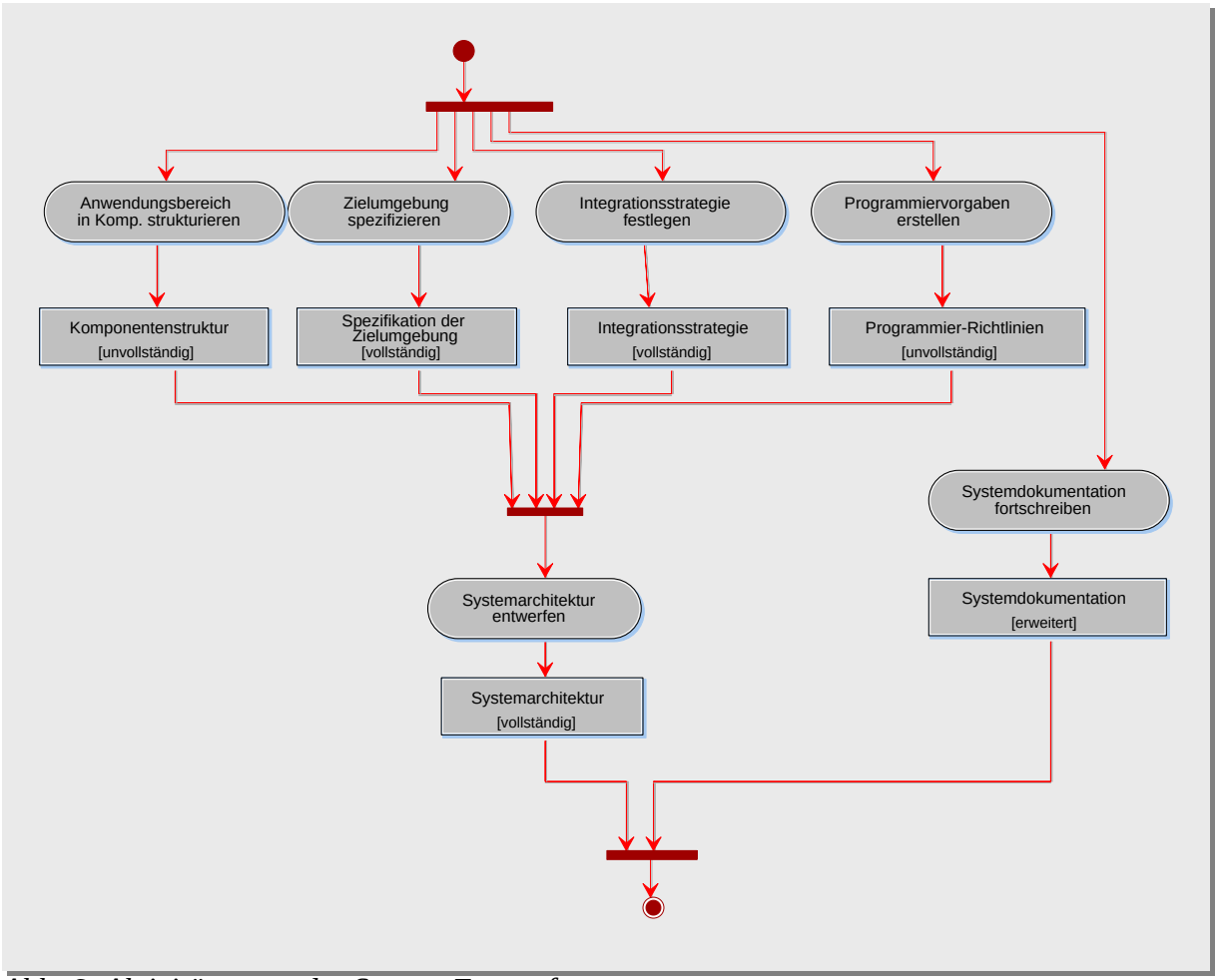


Abb. 6: Aktivitätstypen des System-Entwurfs

Damit die Komplexität der fachlichen Lösung beherrschbar bleibt, werden größere Modelleinheiten für die Klassen des Anwendungsbereichs gebildet. In der UML heißen diese Pakete (*Packages*). Ein Paket enthält jeweils eine Gruppe von Klassen, die semantisch zusammengehören, aber durchaus Beziehungen zu anderen Gruppen von Klassen haben können. Wir sprechen in diesem Zusammenhang von Komponenten und behalten in den nächsten Abschnitten diesen Sprachgebrauch bei (Komponente = Paket). Die Strukturierung des Systems in Komponenten hat neben der Reduktion von Komplexität organisatorische Vorteile, wie z.B. Handhabbarkeit der großen Anzahl von Klassen, Unterstützung der Wiederverwendung oder Erleichterung der Projektplanung. Im Rahmen des Aktivitätstyps „Anwendungsbereich in Komponenten strukturieren“ werden Komponenten identifiziert, abgegrenzt, ihre Beziehungen untereinander definiert und die Komponenten-Gliederung geprüft. Ausgangspunkt für diese Tätigkeiten sind die definierten funktionalen und nicht-funktionalen Anforderungen. Diese werden nach logischen Kriterien, wie z.B. fachliche Zusammengehörigkeit, gruppiert. Aus diesen Gruppen können Komponenten abgeleitet werden, die eine definierte Leistung erbringen. Zwischen den Anforderungen und den Komponenten besteht aber keine 1:1 Beziehung. Aus den bereits abgeleiteten Komponenten können noch weitere Komponenten abgeleitet werden, um z.B. gemeinsam genutzte Systemdienste bereit zu stellen oder um zu große Komponenten zu zerteilen.

Bevor die Systemarchitektur entworfen wird, wird die Zielumgebung des geplanten Anwendungssystems vollständig untersucht. Das ist der Zweck des Aktivitätstyps „Zielumgebung spezifizieren“, in der die Elemente der Zielumgebung, wie z.B. Hardware, Netzwerk, Betriebssysteme, festgelegt und spezifiziert werden.

Im Rahmen des Aktivitätstyps „Integrationsstrategie festlegen“ wird eine Strategie zur Integration des Systems aus den Komponenten bzw. Subsystemen ausgearbeitet. Die Systemintegration ist in der Regel ein Bottom-up-Prozeß, der von den Abhängigkeiten der Komponenten bzw. Subsystemen stark beeinflußt wird. Im Prozeß-Handbuch werden wir unterschiedliche Integrationsstrategien, wie z.B. *Bottom-up*, *Top-down*, *Inside-out*, *Hardest-first*, *Outside-in*, *Big-bang*, *Geschäftsprozeß-orientiert*, *Funktions-orientiert*, *Verfügbarkeits-orientiert* vorstellen. Die Auswahl einer Strategie hängt vom zu realisierenden Anwendungssystem ab. Oft ist aber eine Kombination solcher Strategien sinnvoll.

Damit eine vom jeweiligen Autor unabhängige Wartbarkeit und eine hohe Qualität des Quellcodes gewährleistet werden kann, sollten sich Softwareentwickler an einheitlichen Richtlinien orientieren. Falls solch ein Regelwerk als unternehmensweiter Standard nicht vorliegt, so sind entsprechende Programmierrichtlinien, wie z.B. Namenskonventionen, im Rahmen des Aktivitätstyps „Programmiervorgaben erstellen“ festzulegen.

Ziel des Aktivitätstyps „Systemarchitektur entwerfen“ ist es, eine Systemstruktur zu entwerfen, um die fachliche Lösung in der vorhandenen Zielumgebung technisch zu realisieren. Ein geeignetes Architekturkonzept auszuarbeiten, ist eine schwierige und folgenreiche Entscheidung im Prozeß der Software-Entwicklung. Denn von der Architektur hängt es z.B. ab, ob das System die Wiederverwendbarkeit, Erweiterbarkeit oder Wartbarkeit gut unterstützt. Damit ist die Wahl einer geeigneten Systemarchitektur eine langfristige und weitreichende Entscheidung.

Auf der Suche nach der geeigneten Systemarchitektur wird zunächst definiert, welche Ziele mit der Systemarchitektur erreicht werden sollen. Für das Anwendungssystem anzustrebende Architekturziele, wie z.B. Erweiterbarkeit oder Skalierbarkeit, werden gesammelt, dokumentiert und bewertet. Die Systemgliederung in Komponenten hängt von den festgelegten Architekturzielen ab. Beim Entwickeln der Systemstruktur werden folgende Schritte durchlaufen: *Schichten spezifizieren und konstruieren*, *Komponenten in Schichten gliedern*, *Schnittstellen der Schichten spezifizieren*, *Fehlerbehandlungsstrategie entwickeln* und *Systemarchitektur prüfen*. Im Prozeß-Handbuch werden diese Schritte näher behandelt, außerdem werden einige bewährte Entwurfsmuster vorgestellt.

3.2.3 System-Implementierung

Gegenstand der Implementierung auf der Systemebene ist im engeren Sinne die Systemintegration und der Systemtest. Im einzelnen sind die in der nächsten Abbildung definierten Aktivitäten durchzuführen. Die Systemdokumentation muß weiterhin gepflegt und verfeinert werden (Abb. 7).

Ein Subsystem ist eine nicht-disjunkte Zusammenfassung von Klassen, die zu Test- oder Integrationszwecken gemeinsam zum Ablauf gebracht werden. Im Rahmen des Aktivitätstyps „Subsysteme definieren“ werden die für das Anwendungssystem benötigten Subsysteme definiert. Bei der Definition von Subsystemen sollte beachtet werden, daß Subsysteme eine logische Einheit bilden. Das bedeutet:

- Ein Subssystem sollte nur Klassen enthalten die gemeinsam zum Ablauf gebracht werden sollen.
- Ein Subssystem sollte einen Themenbereich enthalten, der für sich allein betrachtet und verstanden werden kann.
- Ein Subssystem sollte Klassen enthalten, die logisch zusammengehören.
- Ein Subssystem sollte eine wohldefinierte Schnittstelle zu seiner Umgebung haben.
- Ein Subssystem sollte Vererbungsstrukturen nur in vertikaler Richtung schneiden, d.h. alle Oberklassen einer Unterklasse sollten in dem Subsystem enthalten sein.
- Die Durchtrennung von Aggregationen sollte vermieden werden.
- Ein Subsystem sollte so wenige Assoziationen enthalten wie möglich.

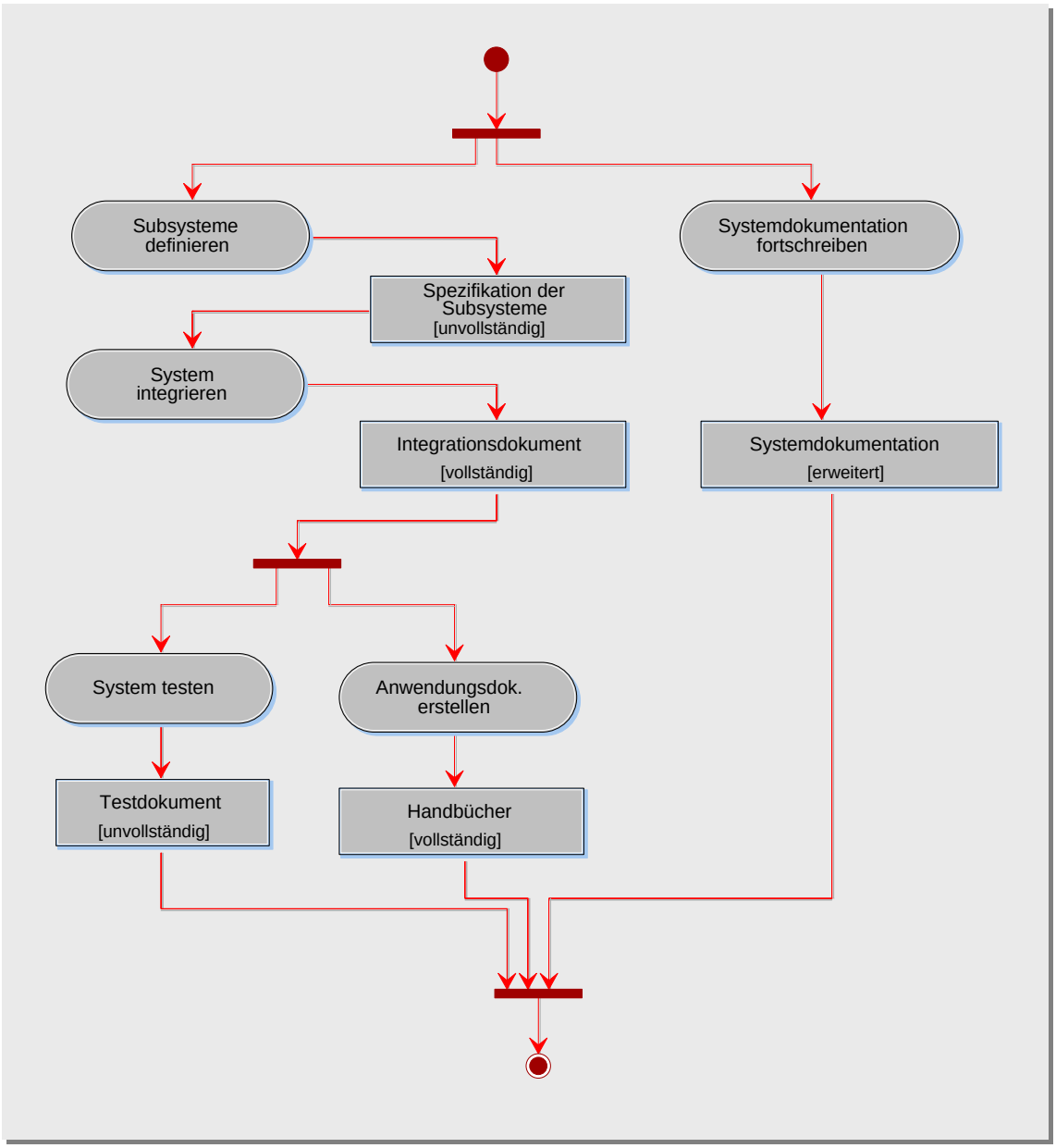


Abb. 7: Aktivitätstypen der System-Implementierung

Ein komponentenbasiertes Anwendungssystem setzt sich aus neuentwickelten, wiederverwendeten bzw. von fremder Seite hergestellten Komponenten zusammen. Ziel der Systemintegration ist es, diese Komponenten zu einem Anwendungssystem zusammenzuführen und in der Zielumgebung zum Laufen zu bringen (Aktivitätstyp „System

integrieren“). Hierbei ist der Nachweis zu erbringen, daß die Komponenten bzw. Subsysteme fehlerfrei über ihre Schnittstellen zusammenarbeiten. Dazu muß die Integration der Komponenten bzw. Subsysteme geplant werden, die Testfälle müssen für den Integrationstest definiert werden, die Integrationsumgebung muß erstellt werden, der Integrationstest der Komponenten bzw. Subsysteme durchgeführt werden und die Fehler analysiert und behoben werden.

Im Rahmen des Aktivitätstyps „System testen“ wird nachgewiesen, daß das Anwendungssystem in der Zielumgebung die Leistungen erbringt, die in der Analyse-Phase spezifiziert worden sind. Es werden eine Reihe von Black-Box-Tests ausgeführt, die zwar aus der Anwendersicht, aber noch ohne direkte Anwenderbeteiligung stattfinden. Mögliche zu überprüfende Qualitätsmerkmale sind: die funktionale Vollständigkeit bezüglich der definierten Anforderungen, die Leistungsfähigkeit (Antwortzeit, Durchsatz, Belastbarkeit, Robustheit, Korrektheit, Sicherheit und Interoperabilität), die Verständlichkeit und Bedienbarkeit, die Installations- und Recovery-Fähigkeit.

Damit das Anwendungssystem erfolgreich eingeführt werden kann, sind eine Reihe von Maßnahmen, wie z.B. „Anwender schulen“ oder „Infrastruktur beschaffen“, notwendig. Die Entwicklung und Umsetzung einer Einführungsstrategie ist aber in erster Linie die Aufgabe des Auftraggebers und nicht mehr als unmittelbarer Bestandteil des Softwareentwicklungs-prozesses anzusehen. Das Projektteam kann aber die Einführung durch die Entwicklung einer Anwendungsdokumentation und durch technische Hilfestellung bei der Inbetriebnahme unterstützen (Aktivitätstyp „Anwendungsdokumentation erstellen“).

3.2.4 System-Operationeller Einsatz

Nach dem erfolgreichen Systemtest beginnt die Phase des operationellen Einsatzes, bei dem die Aspekte der Erprobung, Nutzung und Revision betrachtet werden. In dieser Phase wird der Akzeptanztest durchgeführt und das Anwendungssystem in Betrieb genommen.

Nach der Inbetriebnahme des Anwendungssystems

- treten in der Regel Betriebsfehler auf,
- ändern sich die Umfeldbedingungen, wie z.B. durch den Einsatz neuer Hardware oder neuer Systemsoftware,
- entstehen neue Wünsche und Anforderungen, wie z.B. nach neuen Funktionen oder erhöhter Geschwindigkeit.

Um diese Korrekturen und Verbesserungen vorzunehmen, sind entsprechende Aktivitäten durchzuführen, die in der Abbildung 8 dargestellt sind.

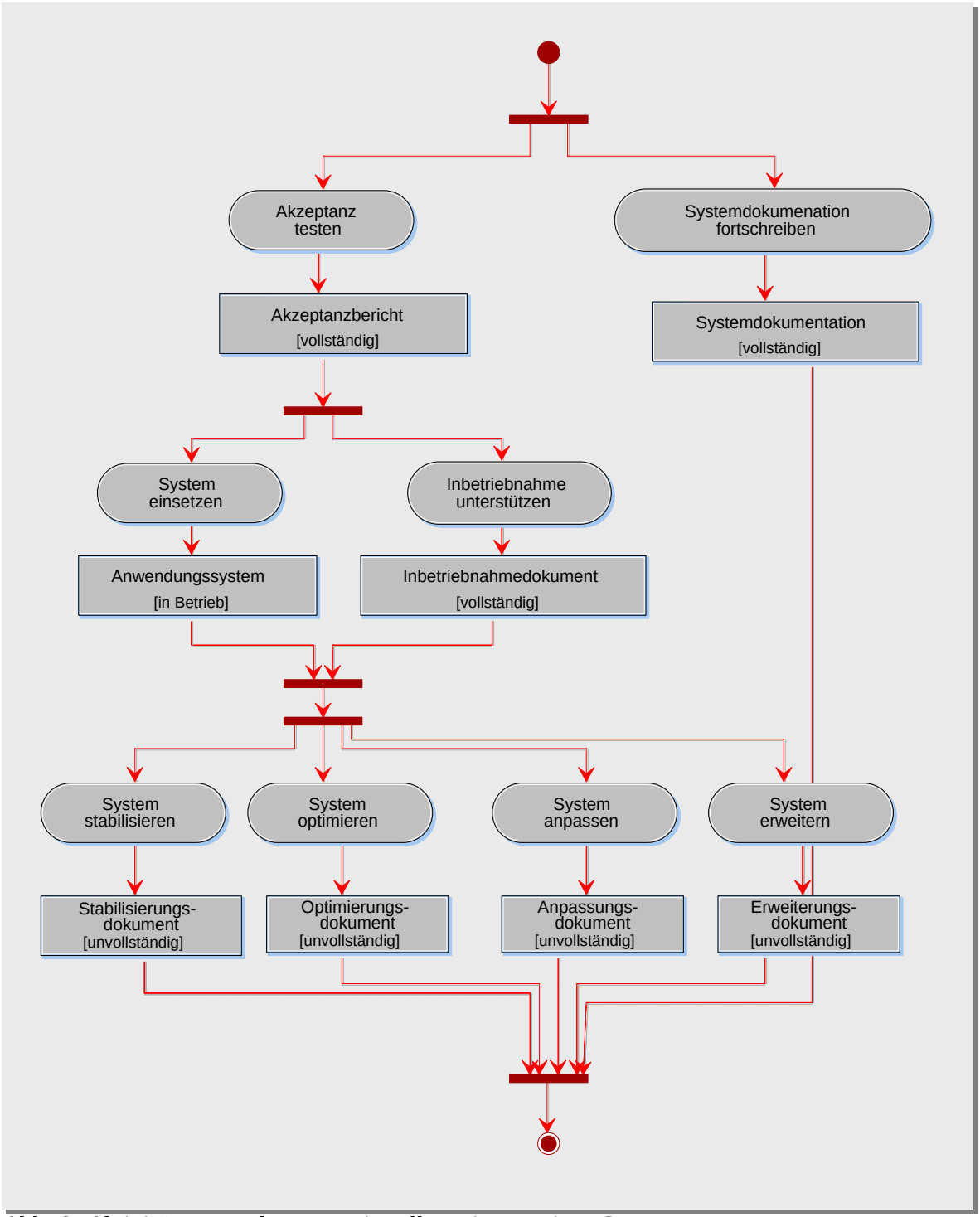


Abb. 8: Aktivitätstypen des operationellen Einsatz eines Systems

Der Akzeptanztest ist eine besondere Ausprägung des Systemtests, bei dem das System unter Beobachtung und Mitwirkung des Auftraggebers in der realen Einsatzumgebung getestet wird. Der Auftraggeber ist für die Planung, wie z.B. die Definition von Testfällen, und für die Durchführung verantwortlich. Beim Akzeptanztest sind folgende Schritte zu durchlaufen (Aktivitätstyp „Akzeptanz testen“): Akzeptanztest planen, Testfälle definieren, Akzeptanztest durchführen und System abnehmen.

Der Aktivitätstyp „Inbetriebnahme unterstützen“ dient dazu, den Endanwender bei der Übernahme des Systems in einen laufenden Betrieb zu unterstützen. Die Übergangsstrategie

und ein Stufenplan für die Systemeinführung werden endgültig festgelegt. Eine wichtige Aufgabe bei der Einführung ist die Umstellung bzw. die Migration von Datenbeständen. Manuelle Karteien müssen entsprechend aufbereitet werden, bevor sie im neuen Anwendungssystem erfaßt werden können. Digitale Datenbestände müssen gegebenenfalls in andere Formate überführt werden, damit sie in das neue Anwendungssystem übernommen werden können. Die eigentliche Inbetriebnahme kann auf drei Arten vorgenommen werden: *direkte Umstellung*, *Parallellauf* und *stufenweise Umstellung*.

Das Ziel des Aktivitätstyps „System stabilisieren“ ist es, auftretende Fehler zu beheben. Dabei kann es sich um Fehler handeln, die bereits bei der Entwicklung in das Anwendungssystem gelangt sind, oder um Fehler, die durch Korrektur-Maßnahmen neu entstanden sind. Neu eingesetzte Software ist nicht nur fehlerhaft, sondern verbraucht auch häufig mehr Zeit und Speicher als zur Erfüllung ihrer Aufgaben erforderlich ist. Denn oft hat die Implementierung der Funktionalität eine höhere Priorität als die Optimierungsroutinen. Das Ziel des Aktivitätstyps „System optimieren“ ist es, die Leistung des Anwendungssystems zu verbessern.

Durch Veränderungen im System-Umfeld werden Anpassungen des Anwendungssystems erforderlich (Aktivitätstyp „System anpassen“). Beispiele für solche Anpassungen sind: Ändern der technischen Umgebung (z.B. neue Systemsoftware), Ändern der Benutzeroberfläche (z.B. modifizierte Dialoge) oder Ändern der Funktionen (z.B. bedingt durch Gesetzesänderungen). Erweiterungen führen zu einer funktionalen Ergänzung des Anwendungssystems (Aktivitätstyp „System erweitern“). Das sind Funktionen, die beim ersten Zyklus vorgesehen waren, aber nicht implementiert wurden oder die sich aus den Erfordernissen des Betriebs ergaben. Je nach Priorität und Umfang der Anpassungen und Erweiterungen muß entschieden werden, ob diese im nächsten Entwicklungszyklus durchgeführt werden. Die betroffenen Personen müssen über diese Entscheidung informiert werden.

3.2.5 Komponenten-Analyse

Es ist nicht realistisch anzunehmen, daß die auf der System-Ebene entwickelte Anforderungsbeschreibung bereits lückenlos und genügend detailliert ist. Daher werden bei der Komponenten-Analyse die funktionalen und nicht-funktionalen Anforderungen, die die jeweilige Komponente technisch realisieren, überarbeitet und verfeinert. Während der Analyse wird die Bausteinbibliothek zwecks Wiederverwendung von Komponenten durchsucht. Außerdem wird für die zu analysierende Komponente eine Komponenten-dokumentation angelegt, die stets erweitert und verfeinert werden muß.

Wie bereits erwähnt, können bei der Anforderungsanalyse unterschiedliche Methoden eingesetzt werden. Wenn die Anwendungsfallmodellierung verwendet wurde, so werden die von einer Komponente zu realisierenden Anwendungsfälle vervollständigt, korrigiert und detailliert dokumentiert (Aktivitätstyp „Anwendungsfallmodell konkretisieren“). Gegebenen-falls werden Anwendungsfälle durch Aktivitätsdiagramme verfeinert oder bereits vorhandene Aktivitätsdiagramme aktualisiert. Die Hauptszenarien der Anwendungsfälle sind in der Regel schon auf der System-Ebene ausreichend spezifiziert. Problematisch sind aber die möglichen Erweiterungen oder Variationen der Hauptszenarien, die z.B. zur Abwicklung von Ausnahmen benötigt werden. Diese gilt es zu konkretisieren und zu dokumentieren.

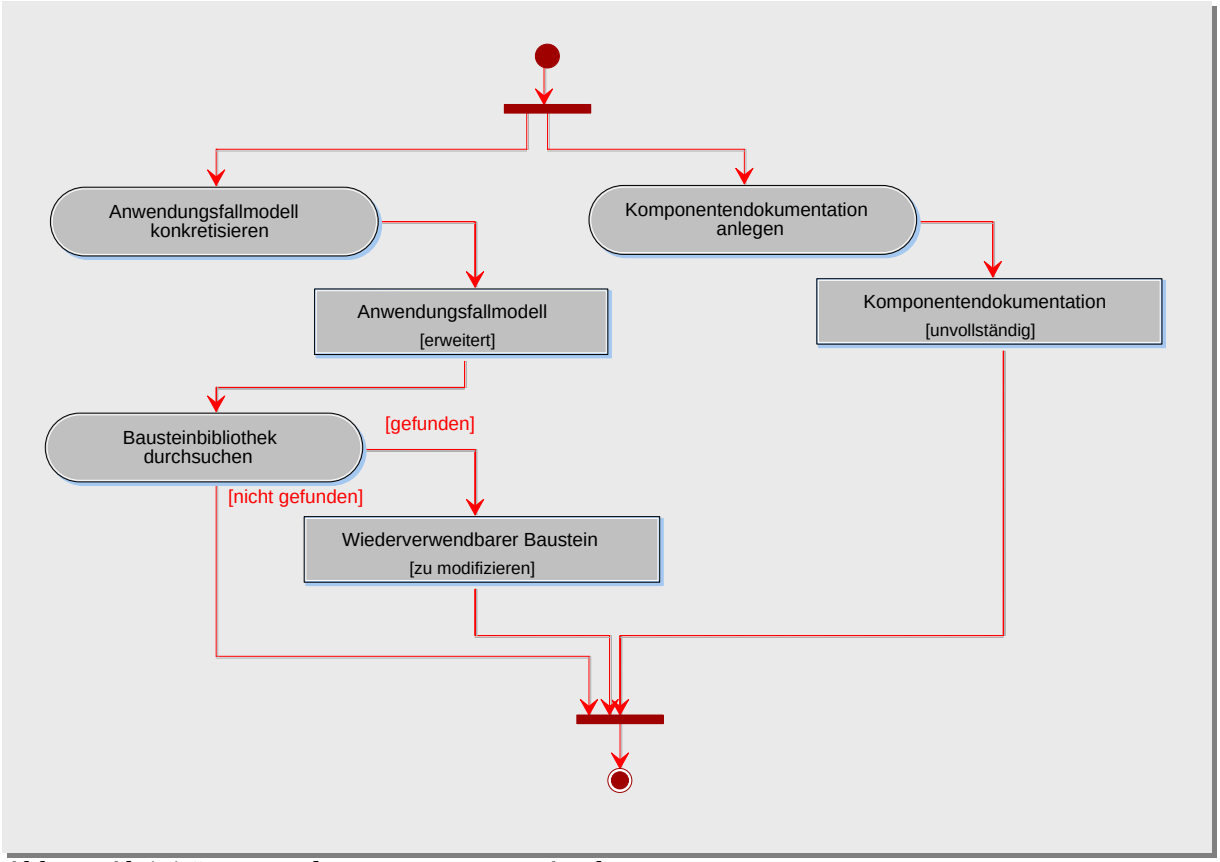


Abb. 9: Aktivitätstypen der Komponenten-Analyse

Im Rahmen des Aktivitätstyps „Bausteinbibliothek durchsuchen“ ist dafür zu sorgen, daß die Baussteinbibliothek nach Komponenten zwecks Wiederverwendung durchsucht wird. Neben der Bausteinbibliothek werden auch andere Quellen, wie z.B. Komponentenanbieter, bei der Suche berücksichtigt. Die Bausteinbibliothek wird auf eine Komponente hin durchsucht werden, die über die gleichen Dienste verfügt wie die zu analysierende Komponente. Ist die Suche erfolgreich, so kann die gefundene Komponente möglicherweise nach Modifikationen wiederverwendet werden. Folgende Schritte werden durchlaufen:

Die gefundene Komponente wird in die bereits vorliegende Komponentenstruktur eingefügt.
Die Beziehungen zu den bereits vorhandenen Komponenten müssen angelegt und geprüft werden.
Die Anforderungen an Modifikationen müssen spezifiziert werden. Diese Anforderungen müssen in den nachfolgenden Phasen realisiert werden.

3.2.6 Komponenten-Entwurf

Nachdem mögliche Subkomponenten einer Komponente identifiziert worden sind, werden für die einzelnen Subkomponenten die untergeordneten Klassen ermittelt. Für die gefundenen Klassen werden die statischen und dynamischen Aspekte modelliert. Ferner ist der Nachweis zu erbringen, daß die der Komponente zugeordneten, funktionalen und nicht-funktionalen Anforderungen durch die gefundenen Klassen erfüllt werden können. Die nächste Abbildung definiert die entsprechenden Aktivitätstypen.

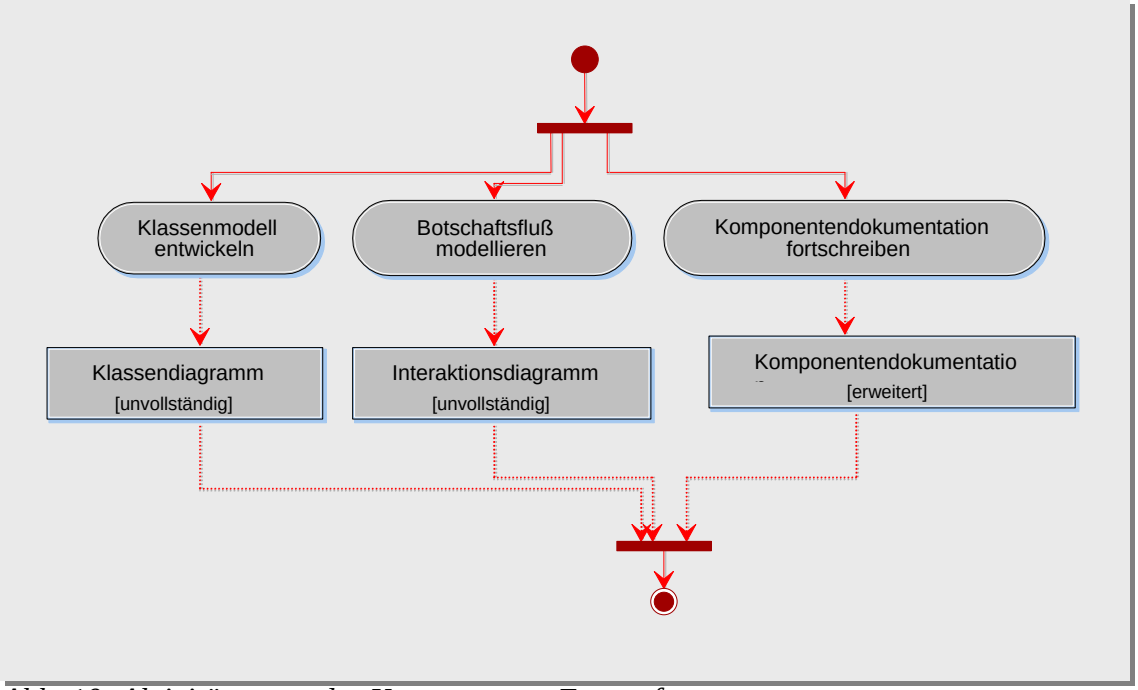


Abb. 10: Aktivitätstypen des Komponenten-Entwurfs

Der Aktivitätstyp „Klassenmodell entwickeln“ dient zum Auffinden von Klassen einer Komponente. Gefundene Klassen werden graphisch durch Klassendiagramme beschrieben. Im Klassendiagramm werden die Beziehungen *Assoziation*, *Aggregation*, *Komposition* und *Generalisierung* zwischen den beteiligten Klassen bzw. Objekten modelliert. Abschließend muß eine Prüfung der Klassendiagramme erfolgen. Dabei können folgende Fragen behilflich sein: *Wurden bei einer Klasse nicht benötigte Schnittstellen definiert? Sind die gefundenen Klassen vollständig?*

Zur Beschreibung von Botschaftsflüssen zwischen Objekten bietet UML die Sequenz- und Kollaborationsdiagramme an. Diese sind sehr nützlich, um Methoden zu finden und die modellierten Klassen zu stabilisieren. Da die Sequenz- und Kollaborationsdiagramme äquivalent sind (d.h. sie können ineinander transformiert werden) haben wir uns auf die in der Praxis häufiger benutzten Sequenzdiagramme beschränkt. Im Rahmen des Aktivitätstyps „Botschaftsfluß modellieren“ werden Haupt- und Nebenszenarien, an deren Durchführung die Objekte beteiligt sind, mit Sequenzdiagrammen modelliert (siehe Prozeß-Handbuch). Die entwickelten Botschaftsmodelle sind zu prüfen und zu dokumentieren. Bei den Prüfungen können folgende Fragestellungen nützlich sein: *Modelliert jedes Sequenzdiagramm wirklich ein fachliches Szenario? Gibt es Klassen, die kaum verwendet werden, so daß auf sie verzichtet werden könnte? Gibt es Methoden, die nicht verwendet werden und auf die verzichtet werden könnte?*

3.2.7 Komponenten-Implementierung

Beim Implementieren von Komponenten werden die entwickelten und bereits einzeln getesteten Klassen integriert. Wie auf der System-Ebene müssen auch für die Komponenten-Integration Subsysteme definiert werden (Aktivitätstyp „Subsysteme definieren“). Für die Integration muß zunächst eine Strategie festgelegt werden (Aktivitätstyp „Komponente integrieren“).

Grundsätzlich werden vier Typen von Integrationsstrategien unterschieden: *nicht-inkrementelle*, *inkrementelle*, *testzielorientierte* und *vorgehensorientierte* (vgl. Prozeß-

Handbuch). In der Regel ist eine Kombination solcher Integrationsstrategien erfolgversprechend. Nach der Integration muß im Rahmen des Komponententests der Nachweis erbracht werden, daß die integrierten Klassen fehlerfrei arbeiten und die von der Komponente erwartete Leistung erbringen (Aktivitätstyp „Komponente testen“). Die nächste Abbildung definiert die durchzuführenden Aktivitätstypen.

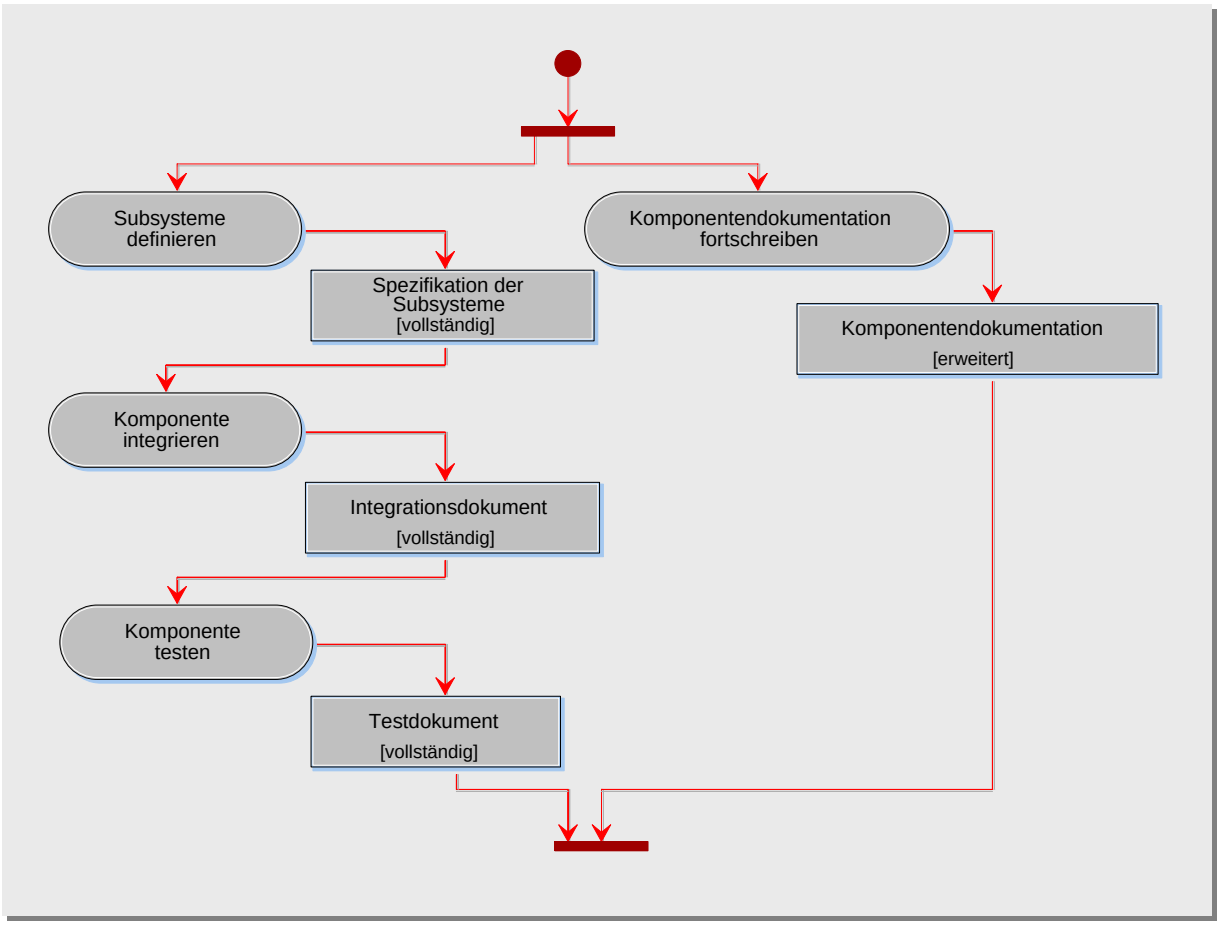


Abb. 11: Aktivitätstypen der Komponenten-Implementierung

Die Komponentendokumentation muß wieder weitergepflegt werden.

3.2.8 Komponenten-Operationeller Einsatz

Mit der Inbetriebnahme des Systems beginnt sowohl für das System als auch für die enthaltenen Komponenten der operationelle Einsatz, bei dem die Aspekte der Erprobung, Nutzung und Revision betrachtet werden.

Die nächste Abbildung zeigt eine Übersicht über die definierten Aktivitätstypen.

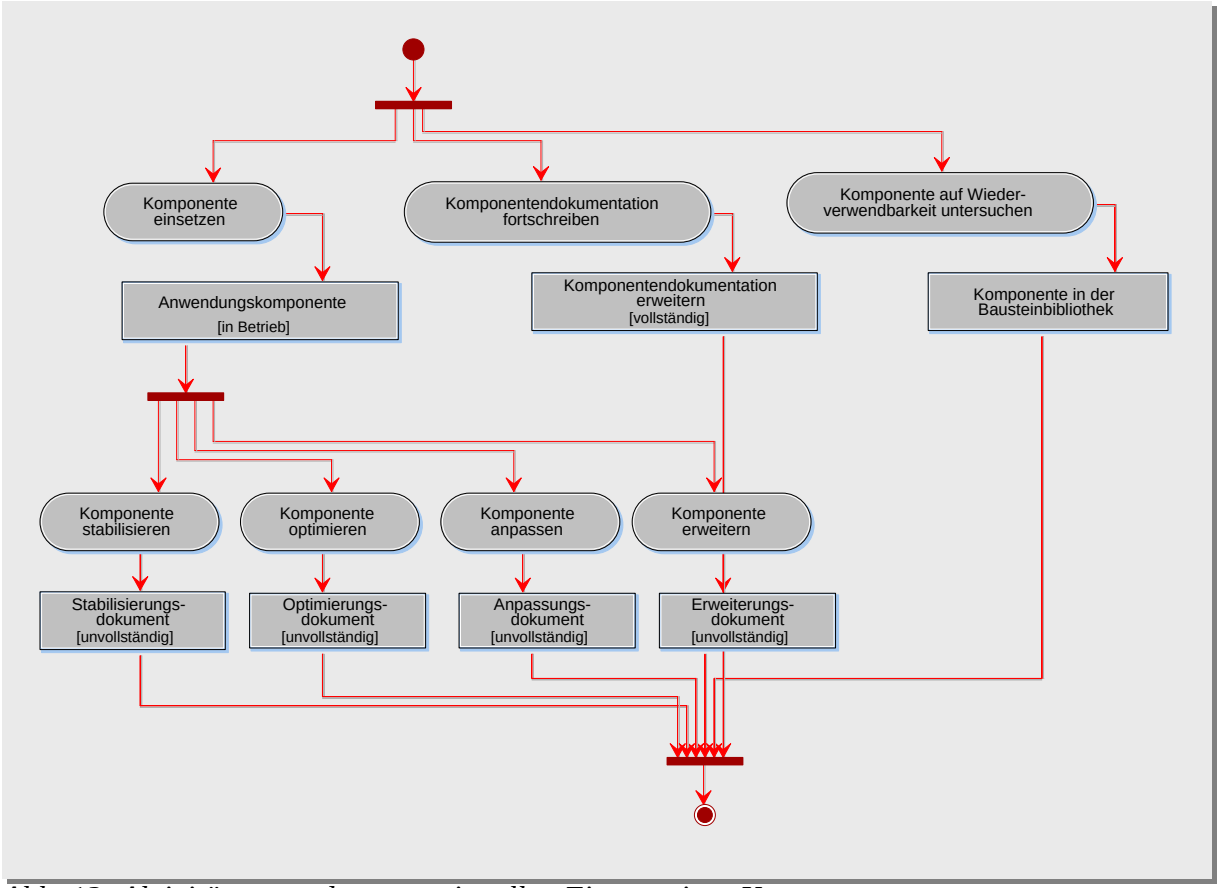


Abb. 12: Aktivitätstypen des operationellen Einsatz einer Komponente

Zu dem bekannten Aktivitätstyp „Komponentendokumentation fortschreiben“ kommt der Aktivitätstyp „Komponente auf Wiederverwendbarkeit untersuchen“ hinzu. Hierbei wird untersucht, ob die Komponente für die Wiederverwendung geeignet ist.

Das System wird in der Regel Fehler aufweisen. Die Anwender werden Optimierungs-, Anpassungs- und Erweiterungswünsche äußern. Die Betriebsfehler und die geäußerten Wünsche beziehen sich auf eine oder mehrere Komponenten des Systems. Damit setzen sich die auf der System-Ebene genannten Aktivitätstypen auf die entsprechenden Komponenten fort. Es sind folgende Aktivitätstypen durchzuführen, um die geforderten Korrekturen und Verbesserungen an einer Komponente vornehmen zu können:

- Komponente stabilisieren,
- Komponente optimieren,
- Komponente anpassen,
- Komponente erweitern.

Eine Komponente zu stabilisieren, heißt die aufgetretenen Fehler zu dokumentieren und zu beheben. Beim Optimieren wird die Systemleistung der Komponente verbessert. Die Anpassung des Systems an ein verändertes Umfeld und die Erweiterung der Systemfunktionalität erfolgt ebenfalls durch die Modifikation der entsprechenden Komponenten. Für jeden der genannten Aktivitätstypen ist zu entscheiden, ob die Änderungen zu einem neuen Komponentenzyklus führt.

3.2.9 Klassen-Analyse

Ein Klassen-Zyklus wird in der Regel auf der Komponenten-Ebene angestoßen. Die Analyse einer Klasse beginnt mit ihrer Identifikation während des Komponenten-Entwurfs. Im Rahmen der Analyse wird die Funktionalität der identifizierten Klasse beschrieben (Aktivitätstyp „Klassenfunktionalität beschreiben“). Außerdem wird die Bausteinbibliothek nach wiederverwendbaren Klassen durchsucht (Aktivitätstyp „Bausteinbibliothek durchsuchen“). Es wird eine Klassendokumentation angelegt, die alle im Rahmen der Klasse entstehenden Ergebnisse beinhaltet (Aktivitätstyp „Klassendokumentation anlegen“), wie z.B. Funktionsbeschreibung oder Zustandsdiagramm.

Die nächste Abbildung zeigt eine Übersicht über die durchzuführenden Aktivitätstypen.

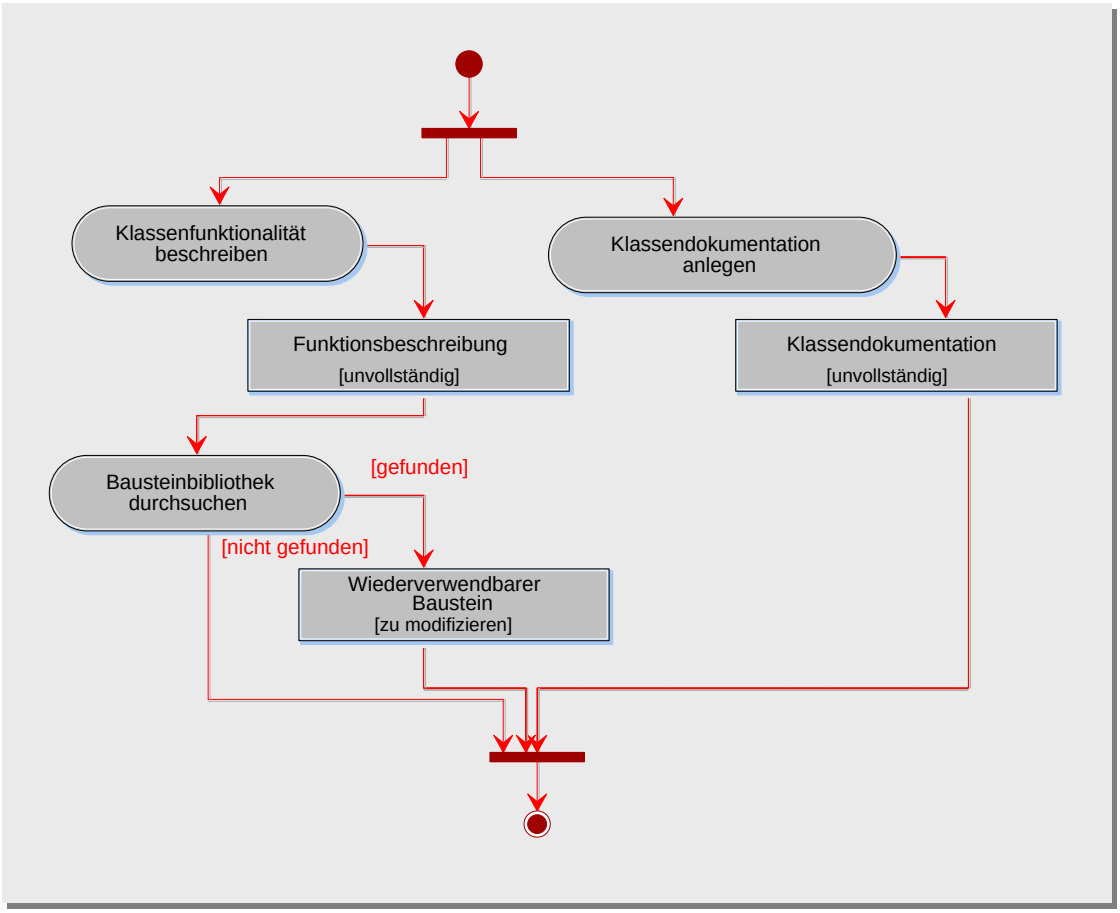


Abb. 13: Aktivitätstypen der Klassen-Analyse

3.2.10 Klassen-Entwurf

Beim Klassen-Entwurf wird die innere Struktur einer Klasse definiert (Aktivitätstyp „Attribute definieren“ und „Methoden definieren“). Für Klassen mit komplexen Lebenszyklen wird ein Zustandsmodell entwickelt (Aktivitätstyp „Zustandsmodell entwicklen“). Zustandsmodelle erleichtern die Analyse des dynamischen Verhaltens von Klassen. Um das dynamische Verhalten einer Klasse zu beschreiben, werden von UML die Zustandsdiagramme zur Verfügung gestellt. Ein Zustandsdiagramm beschreibt die Zustände, die die Objekte einer Klasse im Verlauf ihrer Existenz annehmen können. Es zeigt, welche Ereignisse (z.B. empfangene Botschaften) zu einem Zustandswechsel führen. Außerdem werden die Aktionen, d.h. gesendeten Botschaften, die mit einem Zustandswechsel verbunden

sind, dargestellt. Zustandsdiagramme sind auch für die Identifikation von Methoden sehr hilfreich. Zustandsdiagramme haben einen exemplarischen Charakter, d.h. Zustandsdiagramme nur dann, wenn es sich aufgrund der Komplexität lohnt. So wird z.B. für eine Klasse ein Zustandsdiagramm entwickelt, wenn z.B. ein Objekt der Klasse auf identische Botschaften, abhängig vom aktuellen Zustand, unterschiedlich reagiert.

Bei der Gestaltung des Zustandsdiagramms wird vom Lebenszyklus eines Objekts der betrachteten Klasse ausgegangen, und es werden „durchlebte“ Zustände gesammelt. Ausgehend von den gefundenen Zuständen wird nun nach Ereignissen und Aktionen gesucht, die zu einem Zustandswechsel beitragen. Hierbei werden womöglich neue Methoden gefunden.

Die nächste Abbildung zeigt die definierten Aktivitätstypen für den Klassen-Entwurf.

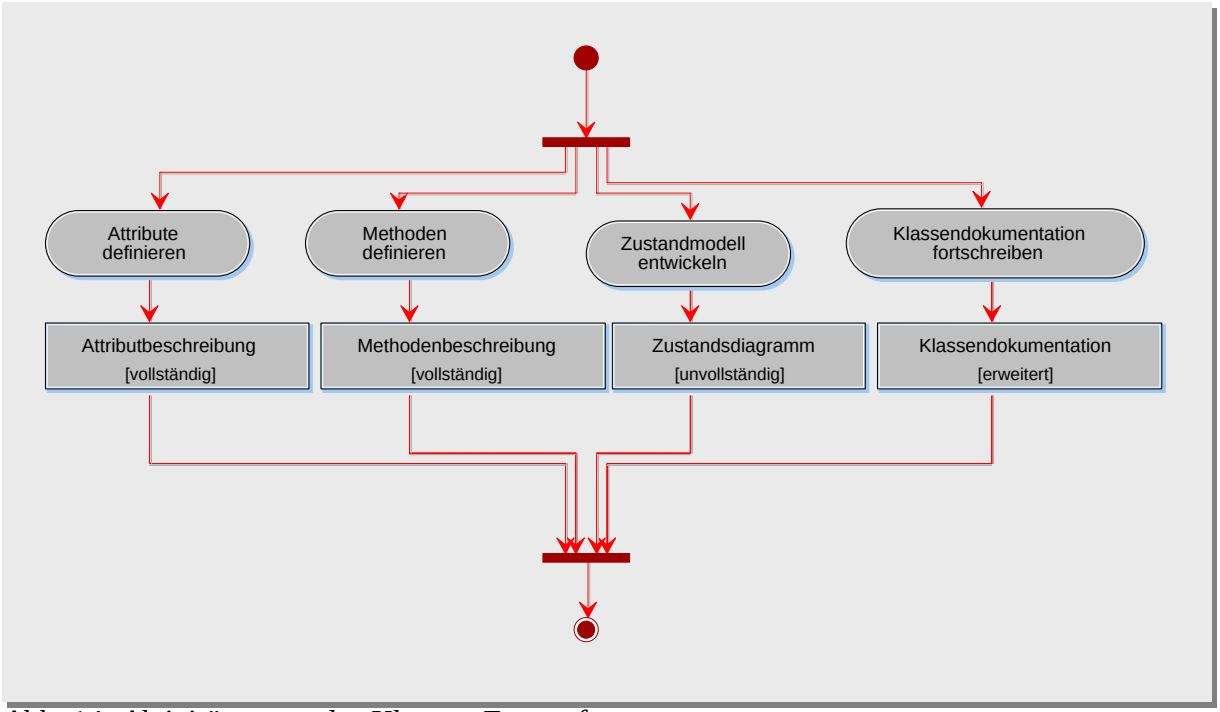


Abb. 14: Aktivitätstypen des Klassen-Entwurfs

Natürlich muß weiterhin die Klassendokumentation erweitert und verfeinert werden.

3.2.11 Klassen-Implementierung

Gegenstand der Implementierung ist es, eine spezifizierte Klasse (vgl. Prozeß-Handbuch, Produkttyp „Klassendokumentation“) zu kodieren, zu kompilieren und zu testen.

Grundlagen für den Aktivitätstyp „Klasse kodieren“ sind das Klassenmodell und die entsprechende Beschreibung der zu implementierenden Klasse. In der Regel wird im Verlauf der Implementierung das Klassenmodell im Detail modifiziert, da z.B. neue Modellelemente hinzu kommen. Bei der Programmierung sollten die Prinzipien *problemadäquate Datentypen*, *Verfeinerung*, *integrierte Dokumentation* und *Verbalisierung* eingehalten werden (vgl. Prozeß-Handbuch).

Das Ziel des Aktivitätstyps „Klasse testen“ ist es, alle implementierten Klassen einzeln zu testen. Das Testen von Klassen entspricht im Kern dem Testen von Modulen bei

konventioneller Entwicklungstechnik. Das bedeutet, daß klassische Testverfahren, wie Strukturtests (White-Box-Test) oder Funktionstests (Black-Box-Test), auf Klassen angewendet werden können (vgl. Prozeß-Handbuch, objektorientierte Testverfahren). Der Klassentest bildet die Grundlage für die schrittweise Integration von Klassen zu einer Komponente bzw. Subsystem. Der Entwickler eines Testfalls spezifiziert das erwartete Verhalten und den Zustand eines Objekts. Während und nach der Durchführung des Tests sind diese mit dem tatsächlich beobachtbaren Verhalten und dem eingetretenen Zustand zu vergleichen. Abweichungen weisen auf Fehler hin, die analysiert und behoben werden müssen.

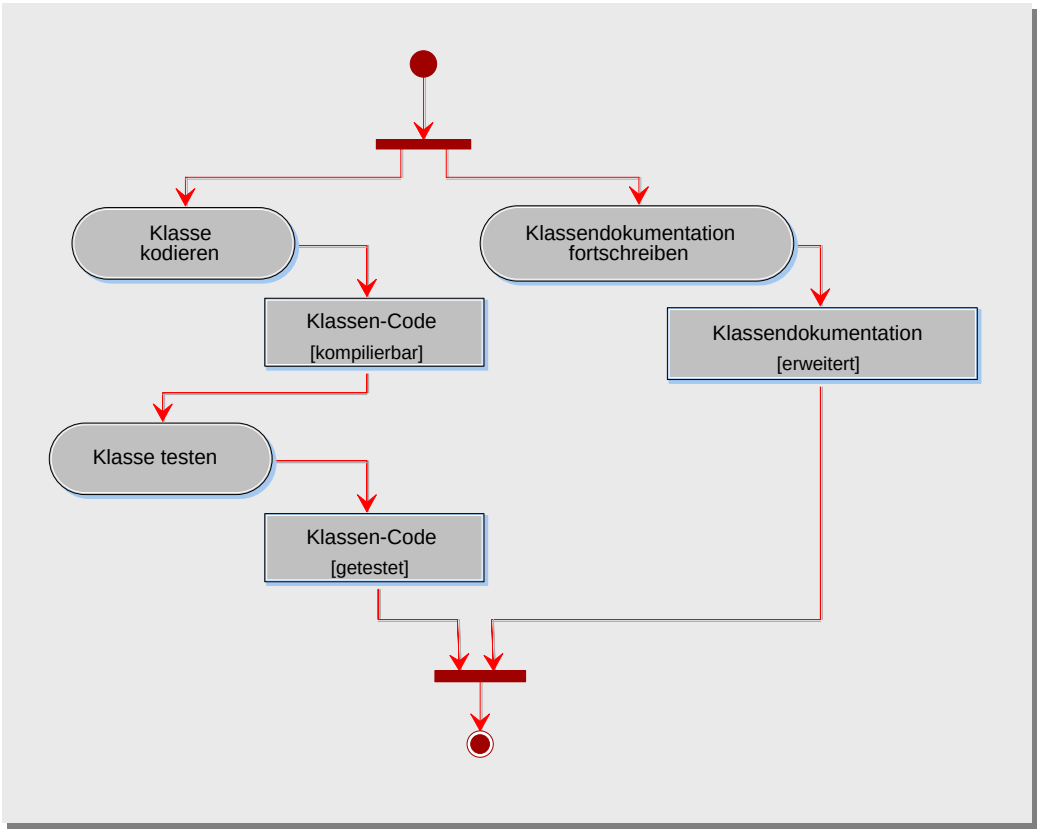


Abb. 15: Aktivitätstypen der Klassen-Implementierung

Die Klassendokumentation muß wieder fortgeschrieben werden. Die obige Abbildung faßt die definierten Aktivitätstypen zusammen.

3.2.12 Klassen-Operationeller Einsatz

Beim operationellen Einsatz muß die Klassendokumentation weiter gepflegt werden und die Klasse auf Wiederverwendbarkeit untersucht werden. Die nächste Abbildung faßt diese Aktivitätstypen zusammen (Abb. 16).

Die im Rahmen des operationellen Einsatzes notwendigen Optimierungen, Anpassungen, Erweiterungen und Fehlerkorrekturen einer Komponente betreffen im Detail ihre zugeordneten Klassen. Daher müssen während des operationellen Einsatzes einer Klasse folgende Schritte unternommen werden, die unter Umständen weitere Klassenzyklen veranlassen können: „Klasse stabilisieren“, „Klasse optimieren“, „Klasse anpassen“ und „Klasse erweitern“.

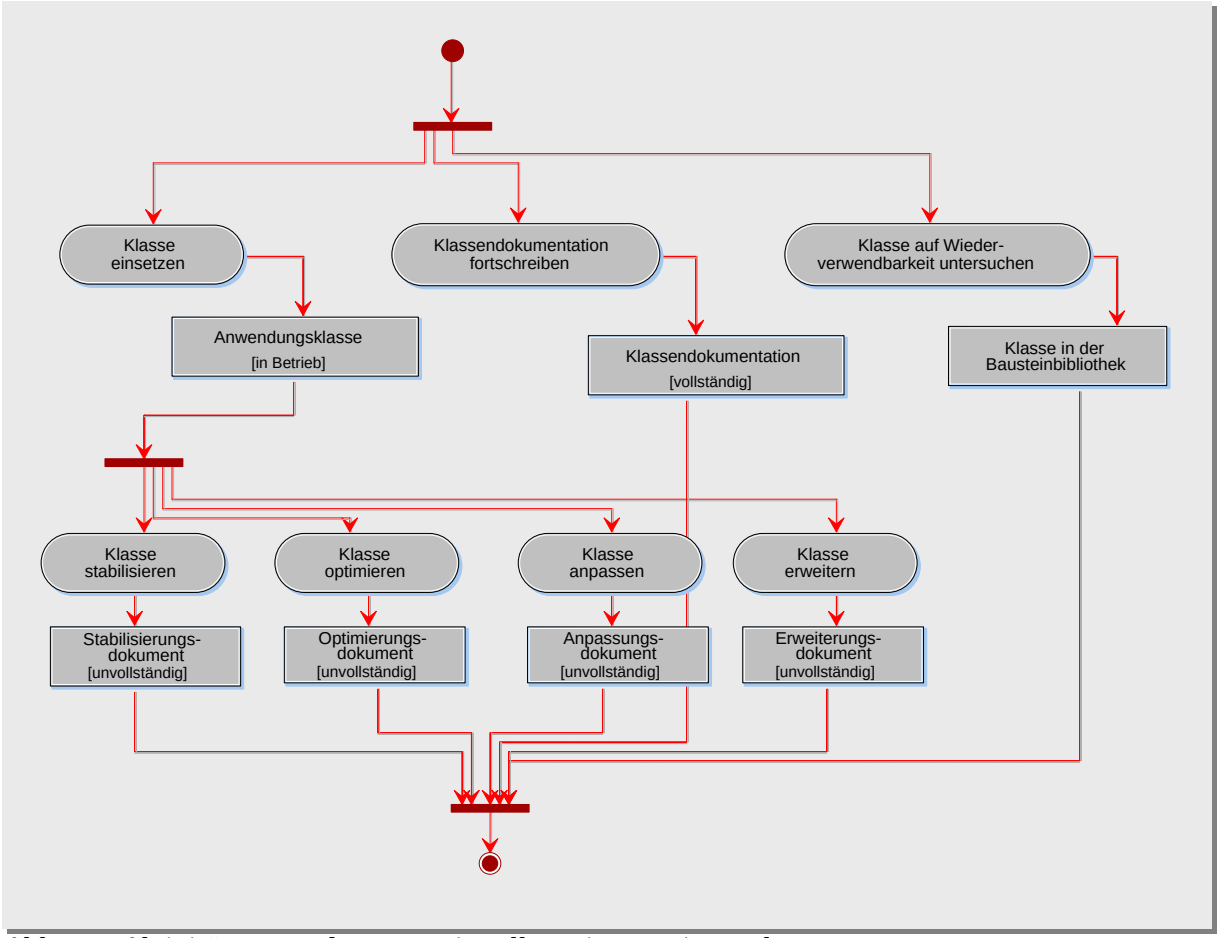


Abb. 16: Aktivitätstypen des operationellen Einsatz einer Klasse

3.3 Qualitätssicherung

In konventionellen Vorgehensmodellen werden analytische Qualitätssicherungs-Maßnahmen im allgemeinen erst am Ende des Entwicklungsprozesses eingesetzt. Das Beheben von Fehlern ist aber teuer, vor allem wenn sie im späteren Verlauf des Projekts entdeckt werden. Um die frühzeitige Fehlererkennung zu ermöglichen, ist im EOS-Modell die Qualitätssicherung im Entwicklungsprozeß integriert und begleitet die Software-Entwicklung. In dem Prozeßmodell EOS ist der primäre Gegenstand der Qualitätssicherung die entwickelten Bausteine. Sobald Analyse, Entwurf, Implementierung oder operationeller Einsatz an einem Baustein abgeschlossen sind, wird eine Prüfung der produzierten Ergebnisse vorgenommen werden. Hierbei kommen hauptsächlich *statische Prüfverfahren* zum Einsatz, wie z.B. Inspektionen oder Reviews. *Dynamische Prüfverfahren*, wie z.B. Funktionstests oder Strukturtests, sind von vornherein im Subprozeß „Software-Entwicklung“ integriert (siehe z.B. Klassentest oder Integrationstest). Zur Qualitätssicherung eines Produkts oder einer Aktivität werden drei Schritte durchlaufen, die wir im folgenden beschreiben.

3.3.1 Prüfplanung

Gegenstand der Prüfplanung ist es, den Prüfvorgang für ein Produkt oder eine Aktivität im Detail zu planen und die Ergebnisse in einem QS-Handbuch zu dokumentieren. Den Prozeß der Prüfplanung haben wir wie folgt definiert (Abb. 17).

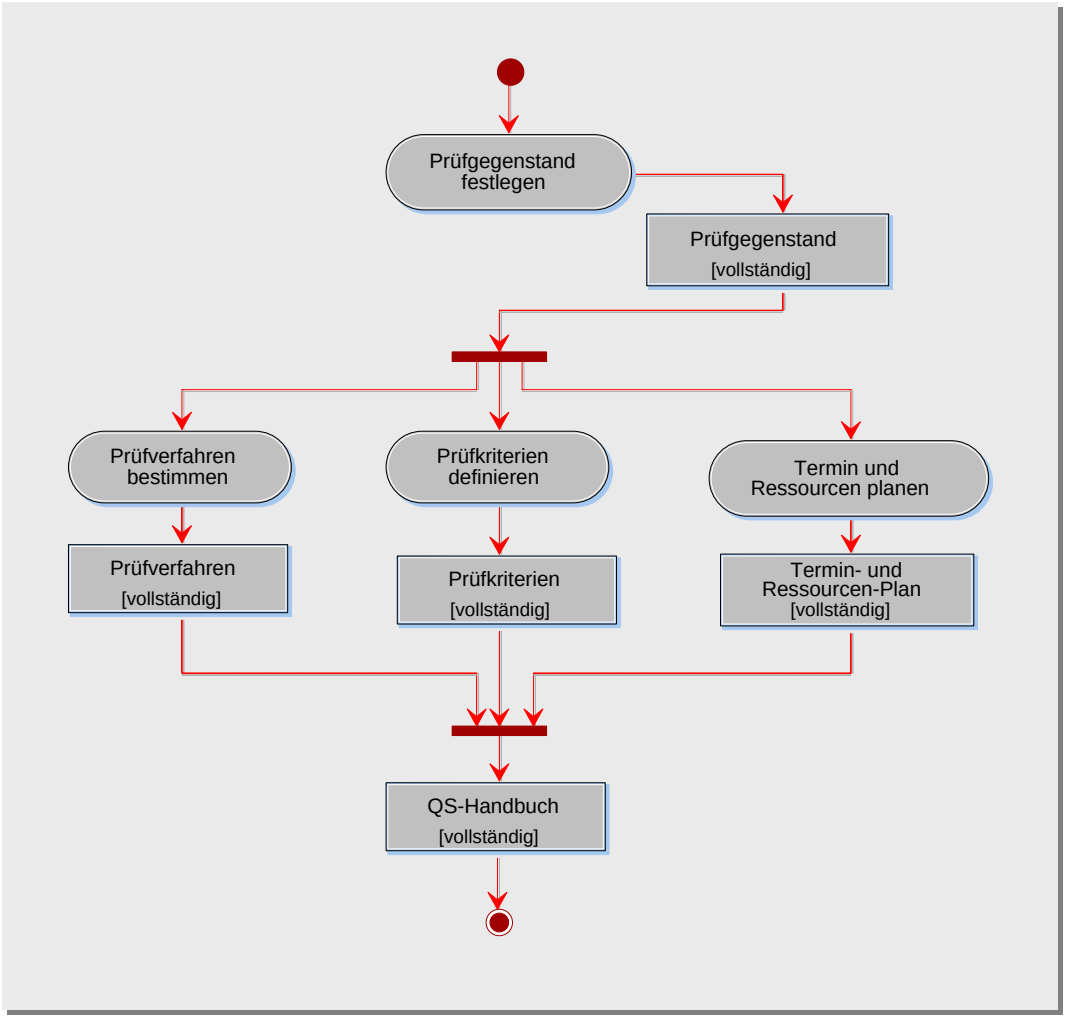


Abb. 17: Aktivitätstypen der Prüfplanung

Im QS-Plan (siehe Prozeß-Handbuch, Kapitel „Projektmanagement“, Abschnitt „Qualitätssicherung planen“) sind die zu prüfenden Produkt- und Aktivitätstypen definiert. Es ist nun zu entscheiden, ob wirklich alle Aktivitäten bzw. Produkte des festgelegten Typs geprüft werden (Aktivitätstyp „Prüfgegenstand festlegen“). In der Regel ist es sinnvoll, Ausnahmen zu machen, z.B. werden nur die Produkte von kritischen Bausteinen geprüft.

In Abhängigkeit von der Zielsetzung lassen sich *testende*, *verifizierende* und *analysierende Verfahren* unterscheiden, aus denen geeignete Prüfverfahren bestimmt werden (Aktivitätstyp „Prüfverfahren bestimmen“). Testende Verfahren dienen zur Erkennung von Fehlern. Verifizierende Verfahren beweisen die Korrektheit eines Bausteins. Analysierende Verfahren vermessen Eigenschaften eines Bausteins.

In dem Schritt „Prüfkriterien definieren“ werden Kriterien zur Prüfung eines Prüfgegenstands festgelegt. Zum Beispiel wäre eine ausreichende Kommentierung ein Kriterium für die Inspektion von Quellcodes. Dieser Schritt basiert auf dem QS-Plan. Für manuelle Prüfmethoden, wie Inspektionen, Reviews oder Walkthroughs, empfiehlt es sich, entsprechende Checklisten zu definieren.

Für die vorgesehenen Prüfungen werden im Rahmen des Aktivitätstyps „Termine und Ressourcen planen“ die Termine und der Ressourceneinsatz geplant. Das Ergebnis der Planung wird mit dem Projektplan abgeglichen. Die Planung wird nach und nach verfeinert,

indem zuerst für das System, dann für die Komponenten und zuletzt für die Klassen geplant wird. Bei der Planung von Terminen und dem Einsatz von Mitarbeitern kann wieder das Aufwandsschätz-Verfahren CEOS genutzt werden. Mit CEOS läßt sich der Aufwand für Analyse, Entwurf, Implementierung und operationeller Einsatz schätzen. Diese Schätzungen berücksichtigen den Aufwand für Prüfungen, die in der Regel nach einer Phase, wie Analyse, Entwurf, Implementierung und operationeller Einsatz stattfinden.

3.3.2 Prüfvorbereitung

Zur Durchführung von Qualitätssicherungs-Maßnahmen sind in der Regel Vorbereitungen nötig. Dazu gehört die Einrichtung der Prüfumgebung, die Definition von Prüffällen und die Entwicklung von Prüfprozeduren (Abb. 18).

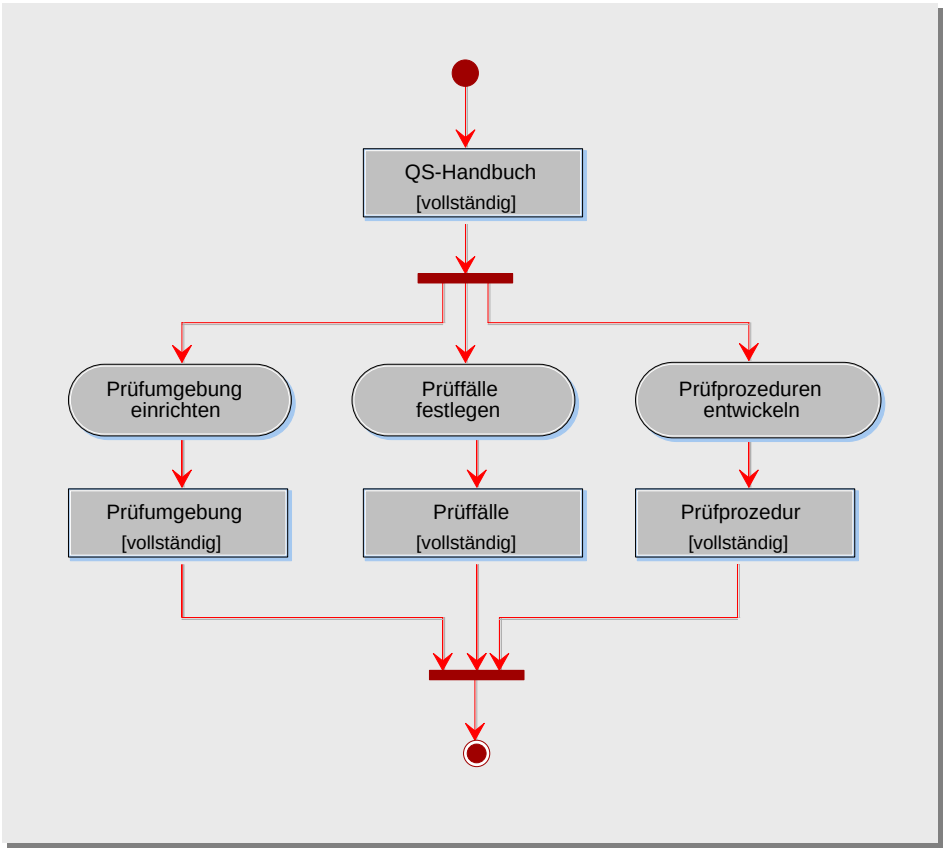


Abb. 18: Aktivitätstypen der Prüfvorbereitung

Zur Prüfung eines Produkts, außerhalb der Entwicklungsumgebung, wird in der Regel zunächst die software- und hardwaretechnische Umgebung eingerichtet (Aktivitätstyp „Prüfumgebung einrichten“). Das Bereitstellen des Arbeitsbereichs für die Qualitätsprüfer gehört zur Prüfumgebung. Zum Beispiel werden gegebenenfalls Werkzeuge installiert und eingerichtet oder benötigte Hardware-Komponenten beschafft.

Beim Entwickeln von Prüffällen können einige Grundsätze behilflich sein (Aktivitätstyp „Prüffälle festlegen“). Prüffälle sollten z.B. stets in einer nachvollziehbaren Weise auf Anforderungen zurück verfolgbar sein. Außerdem sollten Prüfungen kontinuierlich von der Analyse bis zum operationellen Einsatz von Bausteinen erfolgen. Damit wird vermieden, daß Fehler sich fortpflanzen und ihre Beseitigung erschwert wird.

3.3.3 Prüfdurchführung

Im Rahmen der Prüfdurchführung wird ein Produkt hinsichtlich der formalen und inhaltlichen Qualität geprüft (Abb. 19). Am Ende dieses Schrittes wird entschieden, ob das geprüfte Produkt eine akzeptable Qualität hat, oder ob es nachbearbeitet werden muß.

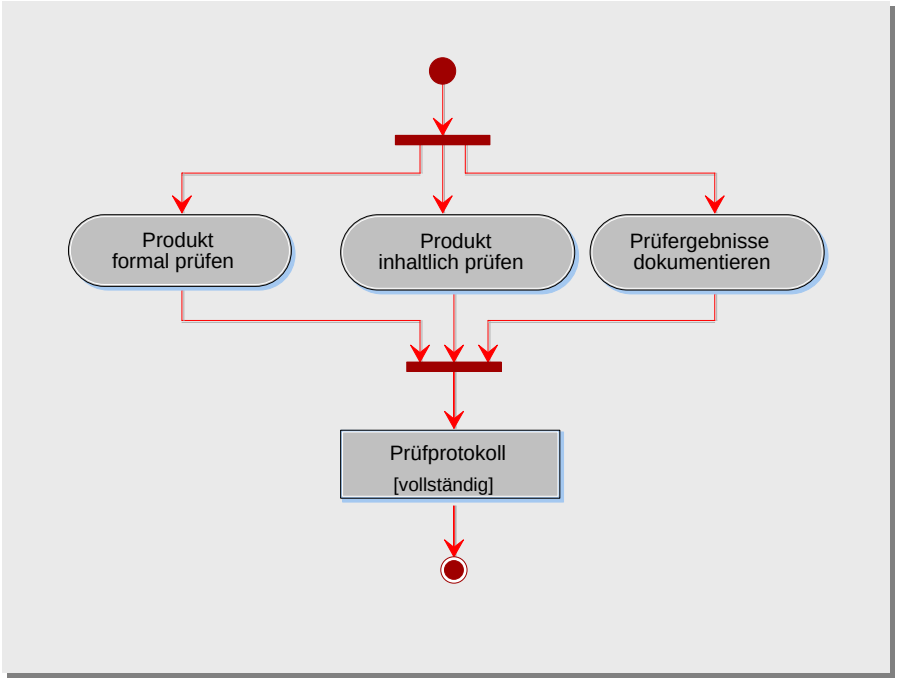


Abb. 19: Aktivitätstypen der Prüfdurchführung

Bevor eine inhaltliche Prüfung durchgeführt wird, sollte das Produkt formale Qualitätskriterien erfüllen. Es ist zu klären, ob das Produkt vollständig vorliegt und verständlich gestaltet ist. Außerdem ist zu überprüfen, ob vorgegebene Standards und Richtlinien eingehalten sind (Aktivitätstyp „Produkt formal prüfen“). Wenn die formale Prüfung erfolgreich abgeschlossen ist, folgt eine inhaltliche Prüfung. Dazu werden die entwickelten Prüffälle herangezogen und anhand der Prüfprozedur die Prüfung durchgeführt (Aktivitätstyp „Produkt inhaltlich prüfen“). Im Rahmen des Aktivitätstyps „Prüfergebnisse dokumentieren“ wird der Verlauf der Prüfung dokumentiert. In Abhängigkeit von den erzielten Ergebnissen ist zu entscheiden, ob eine Überarbeitung veranlaßt werden muß.

3.4 Konfigurationsmanagement

Eine *Konfiguration* ist eine benannte und zusammengehörende Menge von Produkten. Konfigurationen werden benötigt, um Probleme bei den Änderungen und Erweiterungen von Produkten besser zu regeln. Jedes Produkt durchläuft in seinem Lebenszyklus eine Reihe von Entwicklungsstadien. Ein Entwicklungsstand zu einem speziellen Zeitpunkt wird als eine *Version* des Produkts bezeichnet. In der Praxis reicht aber eine Versions-Bildung nicht aus. Denn oft werden weitere Ausprägungen eines Bausteins benötigt, z.B. für unterschiedliche Hardwareplattformen. Diese werden als *Varianten* bezeichnet.

Im EOS-Modell verstehen wir unter einer Konfiguration einen Behälter für alle Produkte, die beim Entwicklungsprozeß eines Bausteins entstehen. Solche Behälter werden von KM-Werkzeugen zur Verfügung gestellt (z.B. im Form eines Dateiverzeichnisses). Eine Konfiguration für eine Komponente X_1 enthält z.B. Klassen- und Sequenzdiagramme, die für

diese Komponente erstellt wurden. Falls X_1 aus Subkomponenten besteht, so sind die Konfigurationen der Subkomponenten in der Konfiguration von X_1 mitenthalten. Das bedeutet, daß die Konfigurationen im direkten Bezug zur Systemarchitektur stehen und sie widerspiegeln. Zur dauerhaften Dokumentation des Entwicklungsstands wird die Konfiguration „eingefroren“. Nach dem „Einfrieren“ wird eine neue Version der Konfiguration der Komponente erstellt. Einen typischen Zeitpunkt für das „Einfrieren“ einer Konfiguration eines Bausteins bildet z.B. der Abschluß der Analyse, des Entwurf, der Implementierung oder des operationellen Einsatzes. Das Konfigurationsmanagement koordiniert und kontrolliert Änderungen systematisch. Dies erfolgt im Allgemeinen durch das *Checkin/Checkout-Modell*. Den Subprozeß „Konfigurationsmanagement“ haben wir wie folgt definiert (Abb. 20):

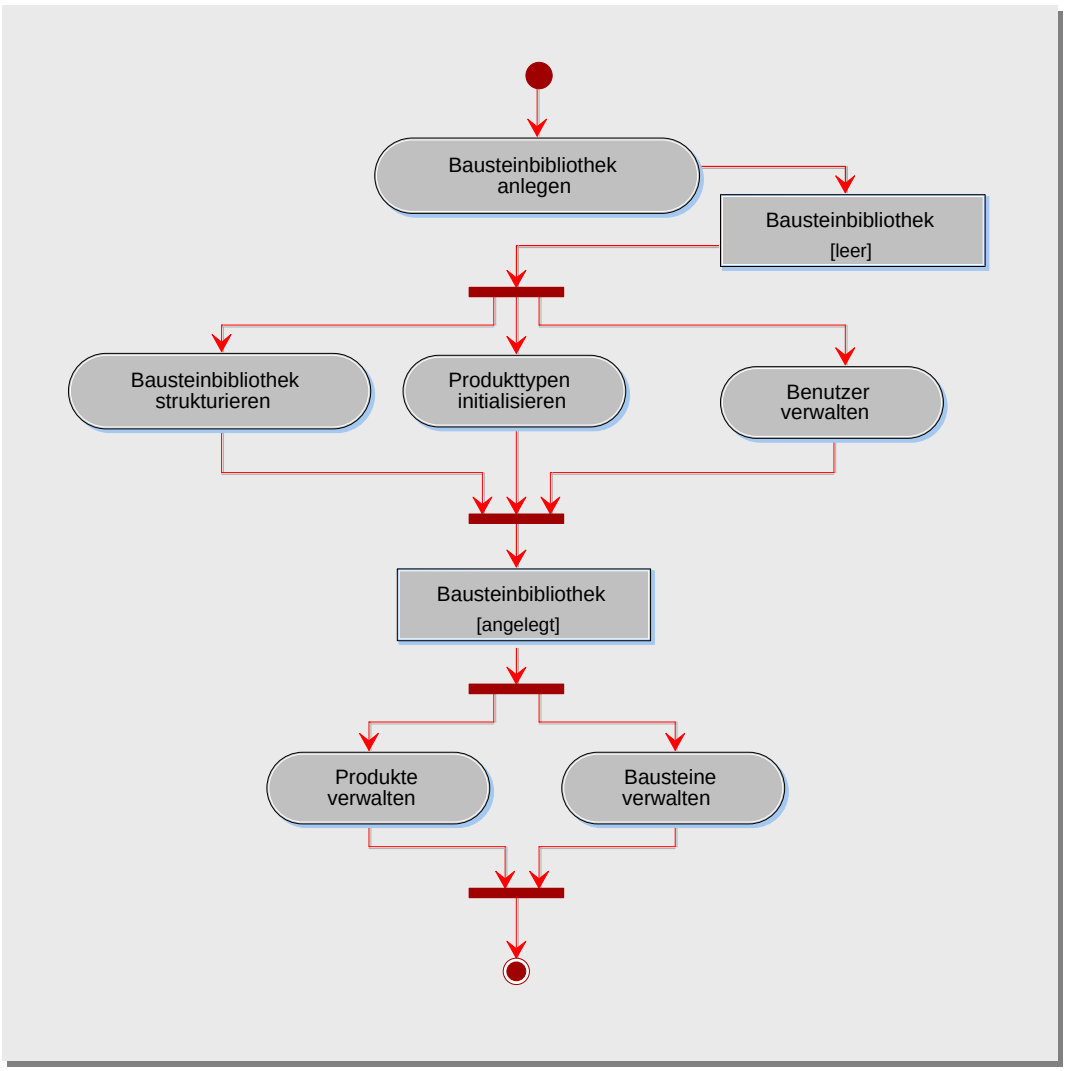


Abb. 20: Aktivitätstypen des Konfigurationsmanagements

Für ein neues Projekt wird die Bausteinbibliothek geeignet strukturiert. Diese Struktur spiegelt die Systemarchitektur wider und wird kontinuierlich an die Veränderungen der Systemarchitektur angepaßt. Im Rahmen des Aktivitätstyps „Bausteinbibliothek anlegen“ wird die Bausteinbibliothek samt der notwendigen Identifikationsmechanismen eingerichtet. Damit wird die Grundlage für die Verwaltung der Ergebnisse des Entwicklungsprozesses geschaffen. Denn die erzeugten Produkte werden mit Hilfe der Bausteinbibliothek systematisch koordiniert und verwaltet, so daß eine effektive Wiederverwendung ebenfalls unterstützt wird.

Das Ziel des Aktivitätstyps „Bausteinbibliothek strukturieren“ ist es, die Bausteinbibliothek so zu strukturieren, daß die Systemarchitektur wiedergegeben wird. Dazu müssen die bereits identifizierten Bausteine in der Bausteinbibliothek angelegt werden und entsprechende Benutzerrechte vergeben werden. Dieser Aktivitätstyp begleitet den Entwicklungsprozeß, da im Verlauf des Projekts die Systemarchitektur geändert und erweitert wird. Im vorherigen Schritt wurden Bausteine in der Bausteinbibliothek eingerichtet. In dem Schritt „Produkttypen initialisieren“ werden Produkttypen, die im Verlauf des Entwicklungsprozesses entstehen können, in der Bausteinbibliothek angelegt und vorbereitet. Nach diesem Schritt ist die Bausteinbibliothek zur Aufnahme von Produkten eingerichtet.

Im Verlauf des Projekts können sich die Rollen der Projektmitglieder ändern, es kommen neue Mitarbeiter hinzu oder Mitarbeiter scheiden aus. Der Aktivitätstyp „Benutzer verwalten“ sorgt dafür, daß jeder Projektmitarbeiter während des gesamten Projektverlaufs im Rahmen der ihm zugewiesenen Rechte Zugang zur Bausteinbibliothek hat.

Nachdem die Bausteinbibliothek strukturiert ist, werden die inhaltlichen Aspekte behandelt. Der Aktivitätstyp „Bausteine verwalten“ begeleitet alle Bausteine, die in der Bausteinbibliothek verwaltet werden. In der Bausteinbibliothek werden eine erste Version der Bausteine bzw. Behälter für Bausteine angelegt und inhaltlich schrittweise durch dazugehörige Produkte gefüllt. Neue Versionen und Varianten der Bausteine werden erzeugt, deren Entwicklung jederzeit zurückverfolgbar ist. Analog zur Verwaltung von Bausteinen begleitet der Aktivitätstyp „Produkte verwalten“ den gesamten Lebenszyklus jedes Produkts. Produkte werden in einer ersten Version in der Bausteinbibliothek angelegt und den Projektmitgliedern zur Weiterentwicklung zur Verfügung gestellt. Es werden neue Versionen und Varianten der Produkte erzeugt, die jeder Zeit zurückverfolgt werden können, so daß auf einer früheren Produktversion wieder aufgesetzt werden kann.

3.5 Nutzung und Bewertung

Im Rahmen dieses Subprozesses wird der Nutzer von Anfang bis zum Ende des Projekts begleitet und betreut. Die Einbindung der Anwender im Entwicklungsprozeß hat viele Fassaden. Im Subprozeß „Software-Entwicklung“ werden z.B. die Anwender von der Anforderungsanalyse bis hin zur Abwicklung von Schulungen soweit sinnvoll einbezogen. Der Abschluß eines Bausteinzyklusses bietet einen Einstiegspunkt, um die Anwender stets in die Entwicklung einzubeziehen. Der Funktionsumfang des Systems wird erweitert, wenn z.B. ein Komponenten-Zyklus abgeschlossen ist. Diese neuen Funktionen können den Anwendern präsentiert werden und mit ihnen diskutiert werden. Ein weiterer Aspekt sind Rückmeldungen und Bewertungen der Anwender. Auf diesen Aspekt wollen wir im folgenden exemplarisch näher eingehen.

Nutzer können positive Bewertungen in Form von Lob oder negative in Form von Beschwerden äußern. Diese Rückmeldungen sollten systematisch erfaßt, bewertet und zur Verbesserung des Anwendungssystems bzw. des Entwicklungsprozesses genutzt werden. Positive Rückmeldungen bestätigen eine erfolgreiche Zusammenarbeit und weisen auf einen zufriedenen Auftraggeber bzw. Nutzer hin. Der Umgang mit Beschwerden dagegen ist weitaus komplizierter und erfordert viel Fingerspitzengefühl. Daher konzentrieren wir uns im Folgenden auf das Beschwerdemanagement und schlagen dazu entsprechende Methoden vor (Abb. 21). Die vorgeschlagenen Methoden betonen zwar das Beschwerdemanagement, können aber für beliebige Arten von Bewertungen und Rückmeldungen genutzt werden.

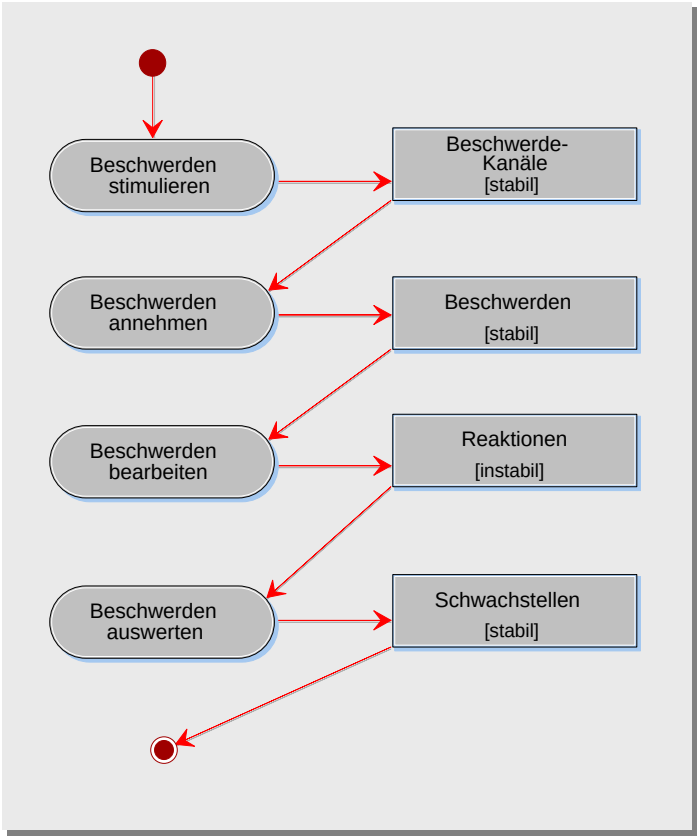


Abb. 21: Aktivitätstypen der Nutzung und Bewertung

In der Regel versuchen die Unternehmen die Anzahl der Beschwerden zu minimieren. Eine Beschwerdeminimierung ist aber nur dann sinnvoll, wenn davon ausgegangen werden kann, daß alle unzufriedenen Nutzer sich beschweren. So wäre eine geringe Beschwerdezahl ein eindeutiger Indikator für die Zufriedenheit der Nutzer. Empirische Untersuchungen belegen aber, daß ein Großteil der unzufriedenen Nutzer sich *nicht* beschweren. Sie reden über die negativen Erfahrungen in ihrem Umfeld und wechseln oft zu einem anderen Anbieter. Sowohl eine negative Mundpropaganda als auch die Abwanderung von Kunden zu anderen Unternehmen verursacht enorme finanzielle Schäden und ist daher zu vermeiden. Beschwerdestimulierung aktiviert Nutzer dahingehend, daß sie jegliche Art von Unzufriedenheit zum Zeitpunkt des Problemauftritts in Beschwerden umsetzen und dem Auftragnehmer mitteilen (Aktivitätstyp „Bewerden stimulieren“). Dazu werden Beschwerdewege (z.B. Formulare im Web oder Hotline) eingerichtet und gegenüber der Nutzer aktiv kommuniziert und aufgezeigt.

Das Ziel des Aktivitätstyps „Beschwerden annehmen“ ist es, den Erstkontakt mit dem sich beschwerenden Nutzer vorzubereiten und alle relevanten Informationen im Hinblick auf eine schnelle und unkomplizierte Bearbeitung des Beschwerdefalls zu erfassen. Nachdem eine Beschwerde eingegangen ist, müssen Maßnahmen unternommen werden, um sie zu behandeln. Dazu wird die Relevanz der Beschwerde geprüft, eine Lösung entwickelt und die Verantwortlichkeiten festgelegt (Aktivitätstyp „Beschwerden bearbeiten“). Bei der Bearbeitung von Beschwerden wird zwischen *internen* und *externen* Arbeitsschritten unterschieden. Zu den internen Schritten gehören alle Maßnahmen, die bei der Bearbeitung von Beschwerden nötig sind, aber von dem Nutzer nicht wahrgenommen werden (z.B. Zuständigkeiten klären oder Lösungswege entwickeln). Externe Schritte umfassen alles, was der Nutzer bei der Bearbeitung von Beschwerden wahrnimmt. Dazu gehört die

Eingangsbestätigung der Beschwerde, die Benachrichtigung über den Bearbeitungsstatus in Zwischenbescheiden und die Mitteilung der Problemlösung im Endbescheid.

Im Mittelpunkt des Aktivitätstyps „Beschwerden auswerten“ steht die aktive Nutzung der erfaßten Beschwerde-Informationen für Verbesserungsmaßnahmen, um künftige Kundenprobleme zu vermeiden und Nutzer durch Kundenzufriedenheit an sich zu binden. Da in den Beschwerden die „Stimme der Nutzer“ hinsichtlich Verbesserungsnotwendigkeit zum Ausdruck kommt, bietet die Beschwerdeauswertung eine Chance, um vom Kunden wahrgenommene Probleme systematisch und konsequent zu eliminieren. Die Beschwerdeauswertung bietet mit Hilfe entsprechender Analysetechniken Informationen, um von der Problemdiagnose zur wirksamen Problemprävention gelangen zu können. Grundsätzlich wird zwischen der quantitativen und qualitativen Beschwerdeauswertung unterschieden. Im Rahmen der quantitativen Beschwerdeauswertung wird das gesamte Beschwerde-Aufkommen mengenmäßig im Hinblick auf wichtige Merkmale analysiert. Dabei werden Verfahren statistischer Methoden genutzt, wie z.B. *Häufigkeitsanalyse* oder *Frequenz-Relavanz-Analyse*. In einem zweiten Schritt werden die Ergebnisse der quantitativen Beschwerdeanalyse interpretiert. Zunächst wird eine systematische *Ursache-Wirkungs-Analyse* vorgenommen, die im Rahmen der *Beschwerde-Problem-Deployment* genutzt wird, um die identifizierten Ursachen zu beseitigen.

In diesem Kapitel wurde ein möglicher Zuschnitt für die fünf EOS-Subprozesse vorgestellt. Der Subprozeß „Projektmanagement“ wurde in die Teilprozesse *Projekt-Initialisierung und Planung*, *Projekt-Steuerung* und *Projekt-Abschluß* gegliedert. Im Rahmen des Subprozesses „Software-Entwicklung“ wurden für die einzelnen Bausteine und ihre vier Phasen Methoden zur Software-Entwicklung angegeben. Der Subprozeß „Qualitätssicherung“ vollzieht sich in den Schritten *Prüfung planen*, *Prüfung vorbereiten* und *Prüfung durchführen*. Das „Konfigurationsmanagement“ wurde in die *Strukturierung* und *Fortschreibung der Bausteinbibliothek* unterteilt. Der Subprozeß „Nutzung und Bewertung“ wurde in die Schritte *Stimulierung, Annahme*, *Bearbeitung und Auswertung von Nutzer-Rückmeldungen* gegliedert.

4 Prototyp eines Prozeßmanagement-Systems

In diesem Kapitel stellen wir einen Prototypen vor, der das EOS-Modell abbildet und somit eine durchgängige Werkzeugunterstützung ermöglicht. Exemplarisch wurde ein Kernmodul des Prototypen vollständig implementiert. Dieses Modul zur Aufwandsschätzung von EOS-Projekten wird im zweiten Teil des Kapitels vorgestellt.

☑ Das Kapitel 3 wird als bekannt vorausgesetzt. Kenntnisse in Java und COM sind für das Verständnis hilfreich.

4 Prototyp eines Prozeßmanagement-Systems

Für den erfolgreichen Praxiseinsatz eines Prozeßmodells ist eine schriftliche Dokumentation allein nicht ausreichend. Entscheidend ist eine Werkzeugunterstützung. Die Anforderung an ein unterstützendes System ist leicht formuliert, aber schwer umzusetzen. Das System muß nämlich die Subprozesse des EOS-Modells mit den jeweiligen Methoden systematisch unterstützen. Jede der definierten Methoden im vorherigen Kapitel bzw. im Prozeß-Handbuch stellt eine Systemfunktion dar.

Wie könnte aber solch ein System gebaut sein? Ein erster Ansatz könnte aus einem System bestehen, das die definierten Methoden durch entsprechende Menüpunkte zur Verfügung stellt. Im letzten Kapitel wurden die Aktivitäts- und Produkttypen des EOS-Modells zusammenfassend vorgestellt. Jede der zahlreichen Aktivitäten repräsentiert eine Methode. Das vorgeschlagene System müßte somit über hundert Menüpunkte bereitstellen. Eine solche Menge von Menüpunkten wäre weder bedienbar noch pflegbar. Hinzu kommt noch, daß die Methoden eng verzahnt sind und ihre Ausführung bestimmte Produkte voraussetzt. Solch ein Ansatz wäre zu unflexibel, um die Dynamik eines Prozesses widerzuspiegeln und würde daher nicht zum Erfolg führen.

Im nächsten Abschnitt wird eine andere Lösung vorgeschlagen, die anhand eines Prototypen dargestellt wird. Diese Lösung nutzt den hierarchischen Aufbau von EOS-Projekten und die Beschreibung der einzelnen Subprozesse durch Aktivitätsdiagramme. Die zugrundeliegenden Aktivitätsdiagramme wurden im vorherigen Kapitel beschrieben. Zunächst wird die Komponentenstruktur des Prototypen beschrieben.

4.1 PMS-Komponentenstruktur

PMS besteht aus einer Reihe von kooperierenden Komponenten. Die nächste Abbildung faßt die logische Struktur der wichtigsten Komponenten zusammen (vgl. Abb. 1).

Das Hauptprogramm von PMS befindet sich in der Komponente „HauptFenster“. Sie stellt grundlegende Oberflächen-Elemente (z.B. Menüpunkte) zur Verfügung und koordiniert die anderen Komponenten. Die Komponente „SysStruktur“ bildet die Systemstruktur als einen Baum ab (vgl. Abb. 5). Die Unterstützungsfunktionen hängen (wie die Aktivitäten) an den Bausteinen der Systemstruktur. Diese Funktionen sind generisch (d.h. für die jeweiligen Bausteintypen immer gleich), können aber pro spez. Baustein immer wieder eingeblendet und gegebenenfalls modifiziert werden. „ProzAuswahl“ ermöglicht den Wechsel zwischen den einzelnen Subprozessen, die in den Komponenten „PM“ (Projektmanagement), „SWE“ (Software-Entwicklung), „QS“ (Qualitätssicherung), „KM“ (Konfigurationsmanagement) und „NB“ (Nutzung und Bewertung) gekapselt sind. Die Komponenten für die fünf Subprozesse verwalten unter anderem die jeweilige Prozeßstruktur (vgl. Abb. 8) in Form von Aktivitätsdiagrammen. Die Aktivitäts- und Zustandsdiagramme mit der zugehörigen Funktionalität werden von der Komponente „UML-Diag“ zur Verfügung gestellt.

Die Komponenten „HauptFenster“, „SysStruktur“, „ProzAuswahl“ und „UML-Diag“ sind vollständig implementiert und können wiederverwendet werden. Bei den Komponenten der Subprozesse wurden exemplarisch einige Kernfunktionen der Komponente „PM“ implementiert. Hierzu gehören EOS-spezifische Funktionen, wie z.B. die Aufwandsschätzung von EOS-Projekten und die Definition von Revisionspunkten.

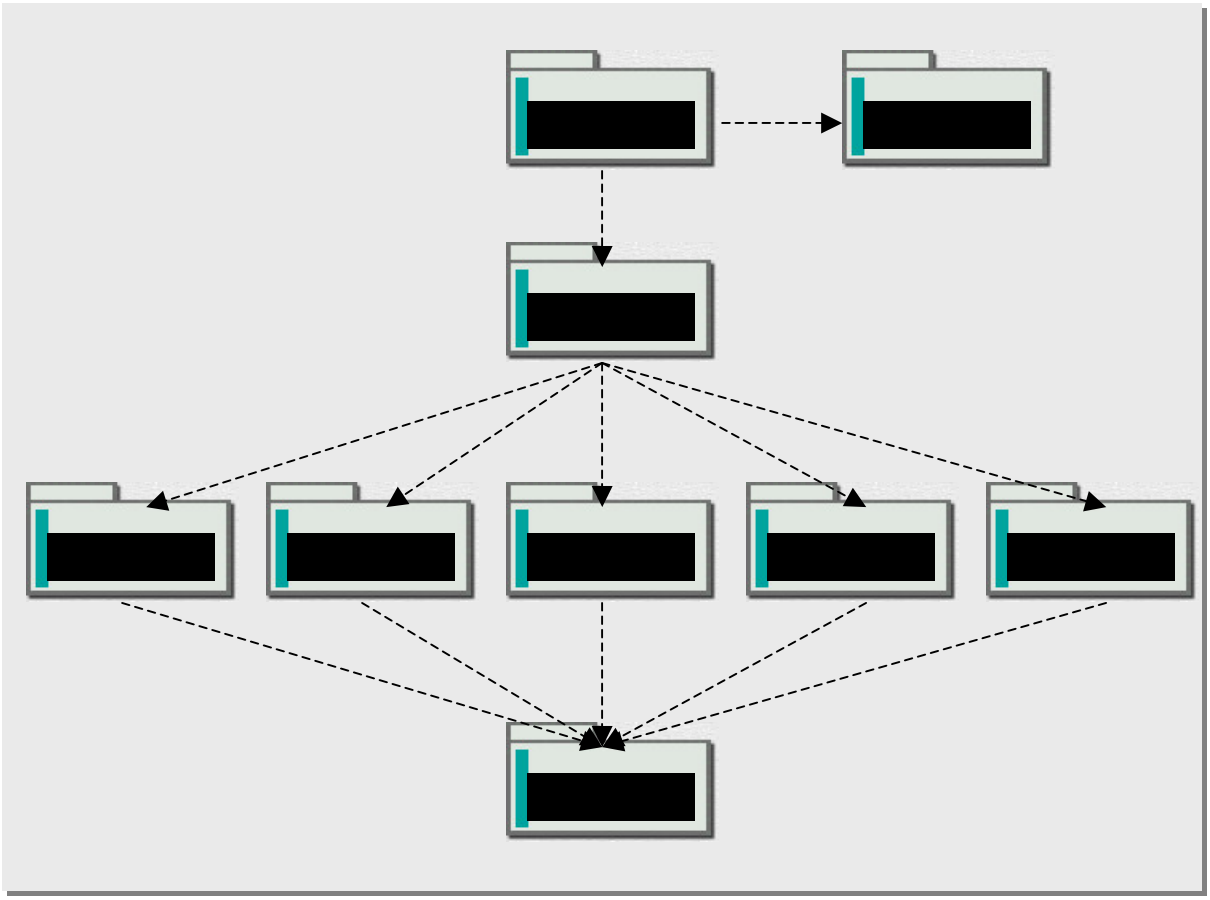


Abb. 1: PMS-Komponentenstruktur (Pfeil-Semantik: X_1 benutzt X_2)

Exemplarisch wird in der nächsten Abbildung die interne Struktur der Komponente „SysStruktur“ dargestellt.

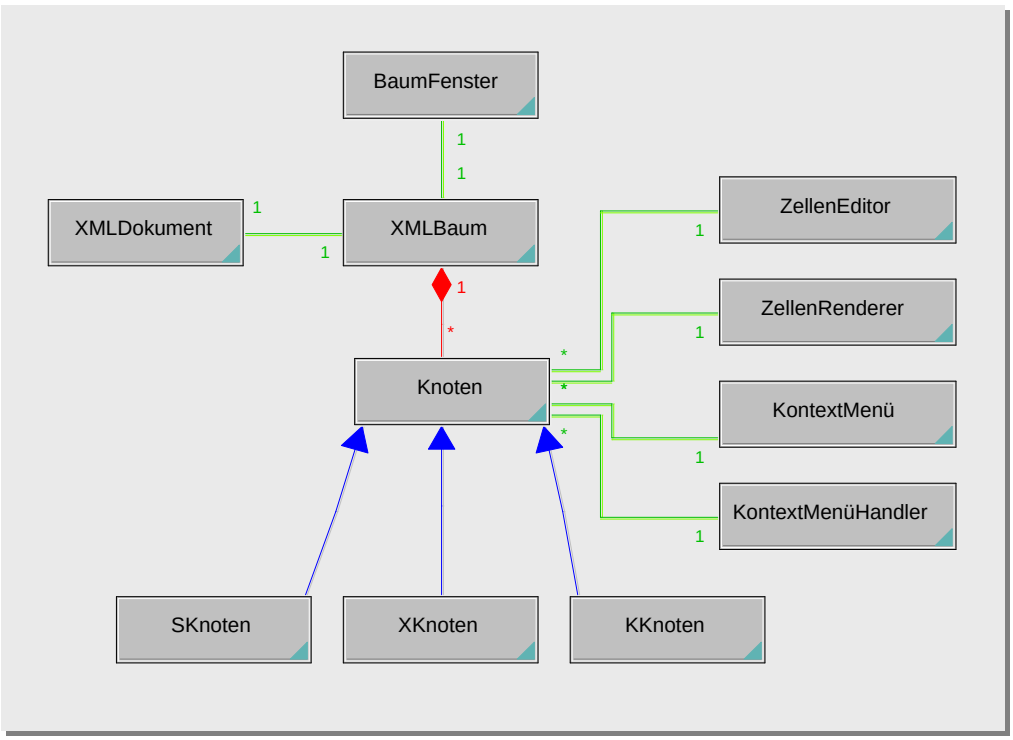


Abb. 2: Klassendiagramm für die Komponente „SysStruktur“

Der Kern der Komponente „SysStruktur“ besteht aus der Klasse „XMLBaum“, die die Systemstruktur als XML-Baum abbildet. Diese Klasse kann die Systemstruktur aus einer XML-Datei einlesen bzw. in einer XML-Datei ablegen (Klasse „XMLDokument“). Der XML-Baum besteht aus Knoten (Klasse „Knoten“), wobei zwischen System-, Komponenten- und Klassen-Knoten unterschieden wird (Klasse „SKnoten“, „XKnoten“, „KKnoten“). Die graphische Repräsentation der Knoten wird mit Hilfe der Klassen „ZellenEditor“ und „ZellenRenderer“ realisiert. Die Kontextmenüs zu den Knoten und die dazugehörigen Aktionen werden durch die Klassen „KontextMenü“ und „KontextMenüHandler“ abgewickelt. Die graphische Darstellung des XML-Baums im Hauptfenster von PMS (vgl. Abb. 5) wird durch die Klasse „BaumFenster“ ermöglicht.

Weitere Details könnten der Dokumentation auf dem beigelegten CD-ROM entnommen werden.

4.2 PMS-Implementierung

Der Prototyp des Prozeßmanagement-Systems (PMS) wurde in der Programmiersprache „Java“ geschrieben und beinhaltet eine Reihe von abgeschlossenen Komponenten, die für eine vollständige Implementierung von PMS wiederverwendet werden können. Im folgenden werden wir PMS anhand seiner Benutzeroberfläche vorstellen.

Nach einem erfolgreichen „Login“ mit gültigem „Usernamen“ und „Passwort“, kann ein neues Projekt mit Hilfe des folgenden Dialogs angelegt werden (Abb. 3).

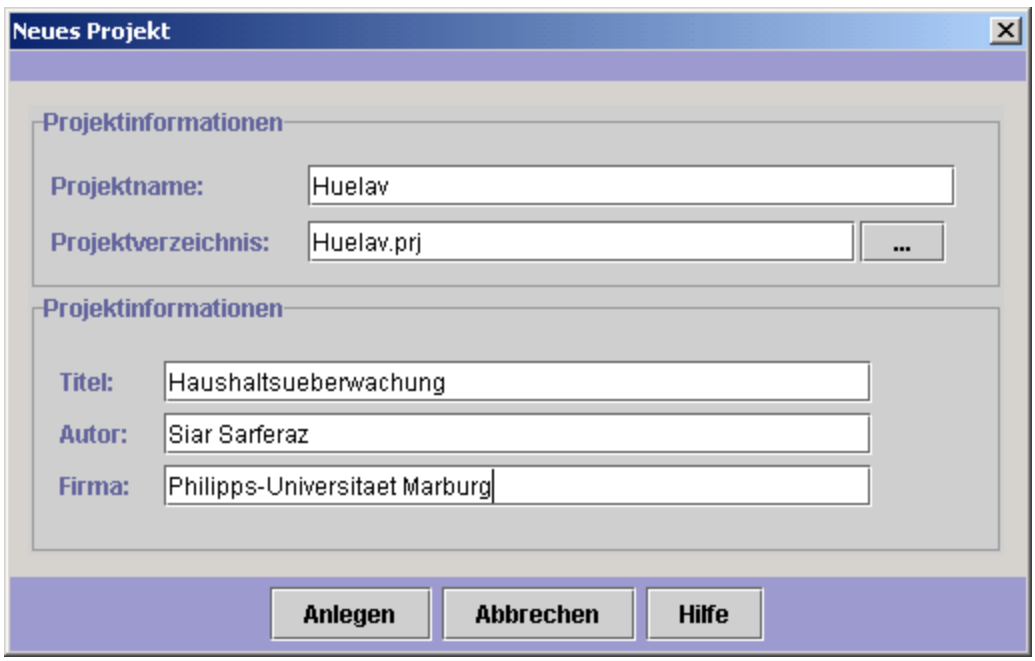


Abb. 3: Dialog für neue Projekte

In dem Dialog können Informationen, wie z.B. Projektname oder Projektverzeichnis, eingegeben werden. Nachdem das Projekt angelegt wurde, erscheint das Hauptfenster, aus dem die Anwendung gesteuert werden kann.

Für das angelegte Projekt „Huelav“ (Haushaltsüberwachung) haben wir bereits einige Daten eingegeben, so daß das Hauptfenster wie folgt aussieht (Abb. 4).

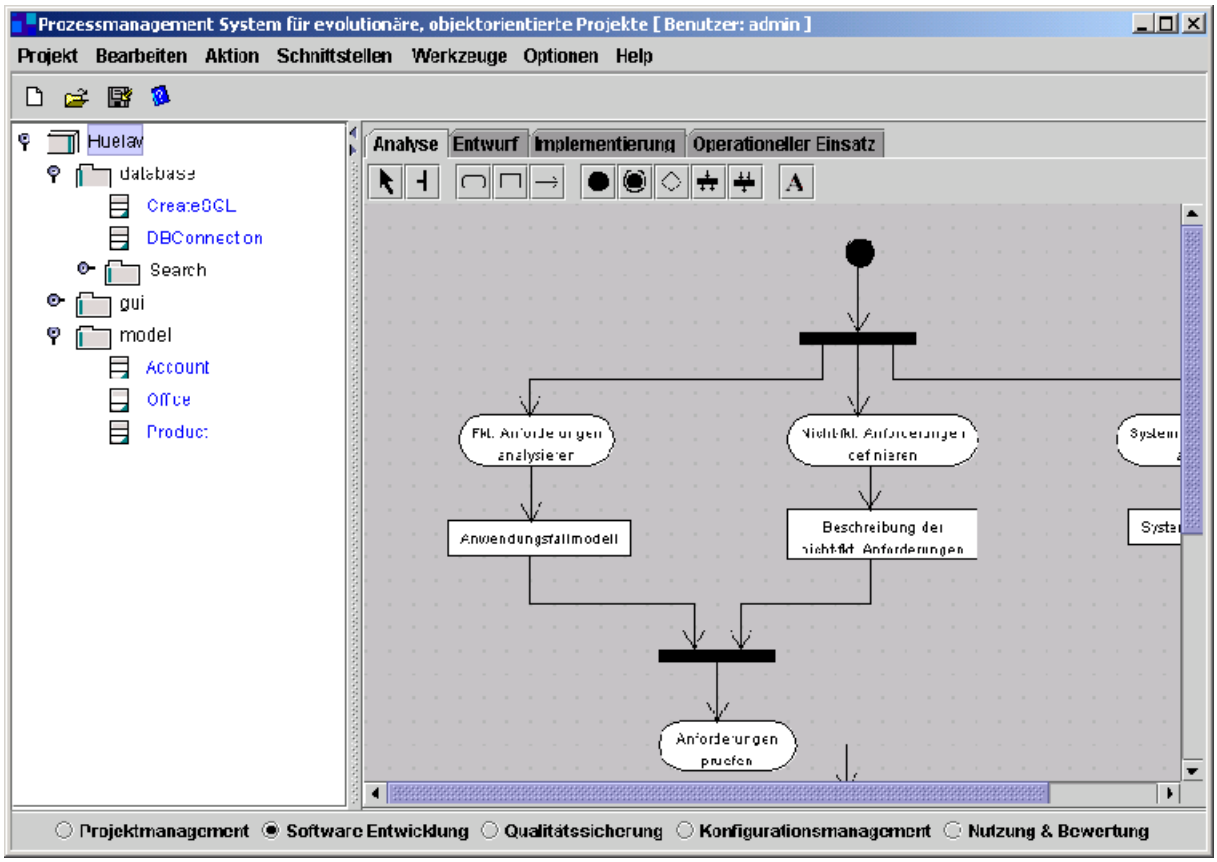


Abb. 4: Hauptfenster der Anwendung

Das Hauptfenster besteht aus den vier Bereichen „Menüführung“ (Menüleiste), „Systemstruktur“ (linkes Teilfenster), „Prozeßstruktur“ (rechtes Teilfenster) und „Prozeßauswahl“ (Statusbar), die im folgenden näher beschrieben werden.

4.2.1 Systemstruktur

Die Systemstruktur gehört zu den Kernkomponenten von PMS, mit deren Erläuterung wir aus didaktischen Gründen als erstes beginnen wollen. Diese Komponente ist vollständig programmiert und kann wiederverwendet werden. Mit ihr läßt sich die Bausteinstruktur eines Systems abbilden. Die Systemstruktur ist XML-fähig, d.h. sie kann aus einer XML-Datei gelesen und in einer XML-Datei geschrieben werden. Das bedeutet, daß die Systemstruktur sowohl importiert als auch exportiert werden kann, so daß eine Kommunikation mit anderen Anwendungssystemen ermöglicht wird.

Für die einzelnen Bausteine existieren Kontextmenüs, die durch die Betätigung der rechten Maustaste aktiviert werden können. Die Kontextmenüs weisen eine untereinander ähnliche Struktur auf. Daher beschränken wir uns darauf, in der nächsten Abbildung exemplarisch ein Kontextmenü für die Komponenten-Ebene zu betrachten (Abb. 5).

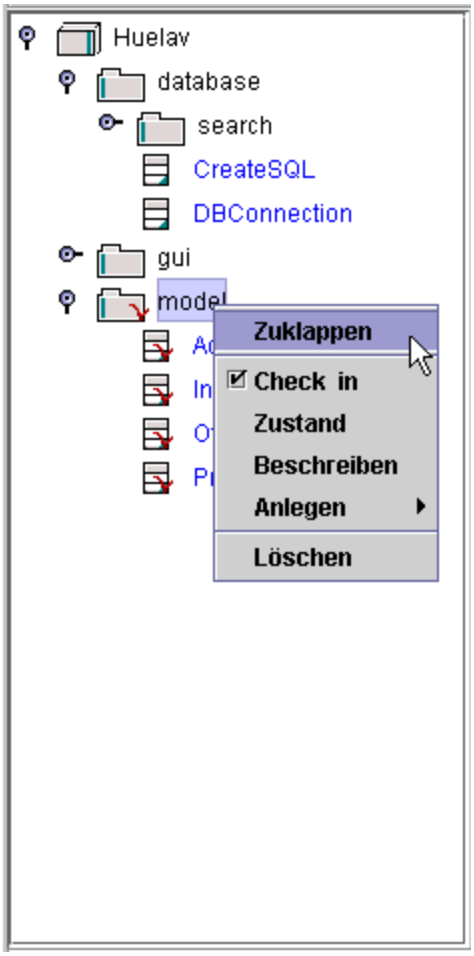


Abb. 5: Systemstruktur

Der Menüpunkt „Aufklappen“ bzw. „Zuklappen“ bewirkt, daß die Nachfolger eines Knoten angezeigt bzw. ausgeblendet werden.

„Check in“ bzw. „Check out“ simulieren nur die entsprechenden Funktionen eines Konfigurations-management-Systems (siehe KM-Kapitel, im Prozeß-Handbuch). Ein „Check out“ durch einen Akteur A bewirkt, daß dieser Schreibrechte auf die Daten eines Bausteins hat, während alle anderen nur Lesezugriffe haben. Durch den „Check out“ wird der entsprechende Baustein mit einem roten Häkchen versehen. Ein „Check in“ schreibt die modifizierten Daten eines Bausteins in die entsprechende Quelle und gibt die Daten für ein erneutes „Check out“ frei.

Mit Hilfe des Menüpunkts „Anlegen“ können weitere Bausteine angelegt werden. In unserem Fall wurde eine Komponente gewählt, so daß als weitere Bausteine eine Subkomponente oder untergeordnete Klassen angelegt werden können.

Mit dem Menüpunkt „Löschen“ kann ein Baustein aus der Systemstruktur entfernt werden. Das Umbenennen eines Bausteins erfolgt durch das einfache Klicken auf den Namen des Bausteins oder durch das Betätigen der „F2-Taste“.

Pro Zyklus durchläuft jeder Baustein die Zustände „Analyse“, „Entwurf“, „Implementierung“ und „operationeller Einsatz“. Der aktuelle Zustand und die Anzahl der bereits durchlaufenen Zyklen eines Bausteins können mit Hilfe des Menüpunkts „Zustand“ abgefragt werden (Abb. 6). Nach dem Betätigen des Menüpunkts erscheint ein Dialog, der neben der Anzahl der bereits durchlaufenen Zyklen ein Zustandsdiagramm enthält. Das Zustandsdiagramm beschreibt mit Hilfe von unterschiedlich gefärbten Zuständen die bereits durchlaufenen und die noch zu durchlaufenden Zustände des Bausteins.

Damit die vier definierten Zustände verfeinert werden können, wurde ein „Toolbar“ (vgl. Abb. 6, oben) in dem Dialog integriert, mit dem sich beliebige Zustandsdiagramme nach dem UML-Standard einbinden und entwickeln lassen. Die einzelnen Elemente des Zustandsdiagramms beinhalten Kontextmenüs, die in erster Linie zur Layout-Modifikation dienen. Das Zustandsdiagramm gehört ebenfalls zu den vollständig implementierten Komponenten, die in zukünftigen Projekten wiederverwendet werden können.

Mit dem Menüpunkt „Beschreiben“ im Kontextmenü eines Bausteins läßt sich ein Baustein dokumentieren. Nach Anklicken dieses Menüpunkts wird MS-Word mit einem Beschreibungsschema zum Ausfüllen gestartet (Abb. 7). Die entsprechenden Beschreibungsschemas für die einzelnen Bausteine wurden bereits im vierten Kapitel behandelt. Durch die Integration von MS-Word steht dem Anwender der gesamte Funktionsumfang eines Textverarbeitungssystems zur Verfügung, so daß die Bearbeitung von Beschreibungsschemata erleichtert wird.

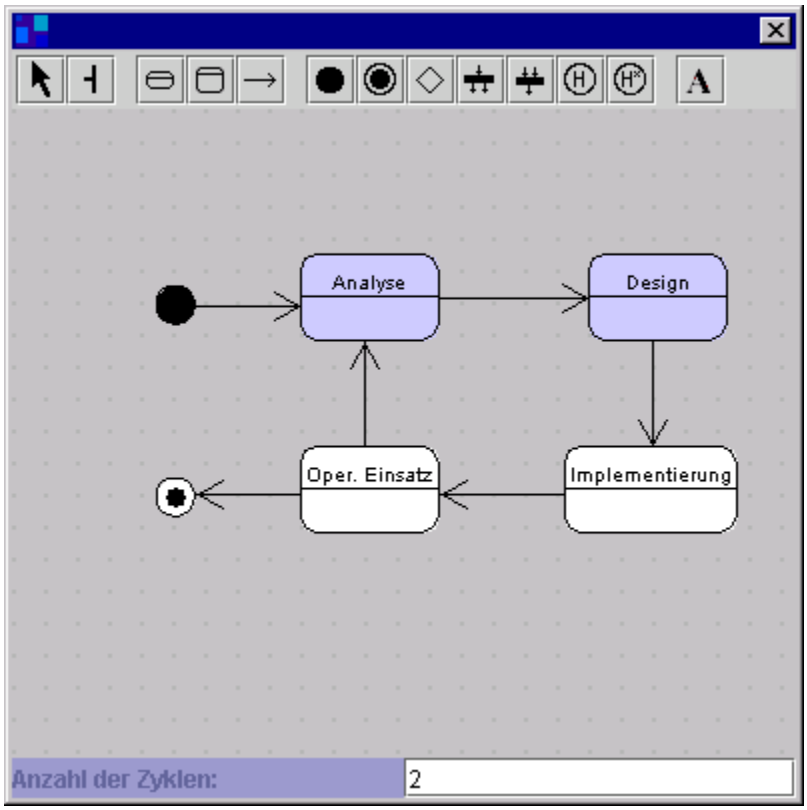


Abb. 6: Zustände und Anzahl der Zyklen eines Bausteins

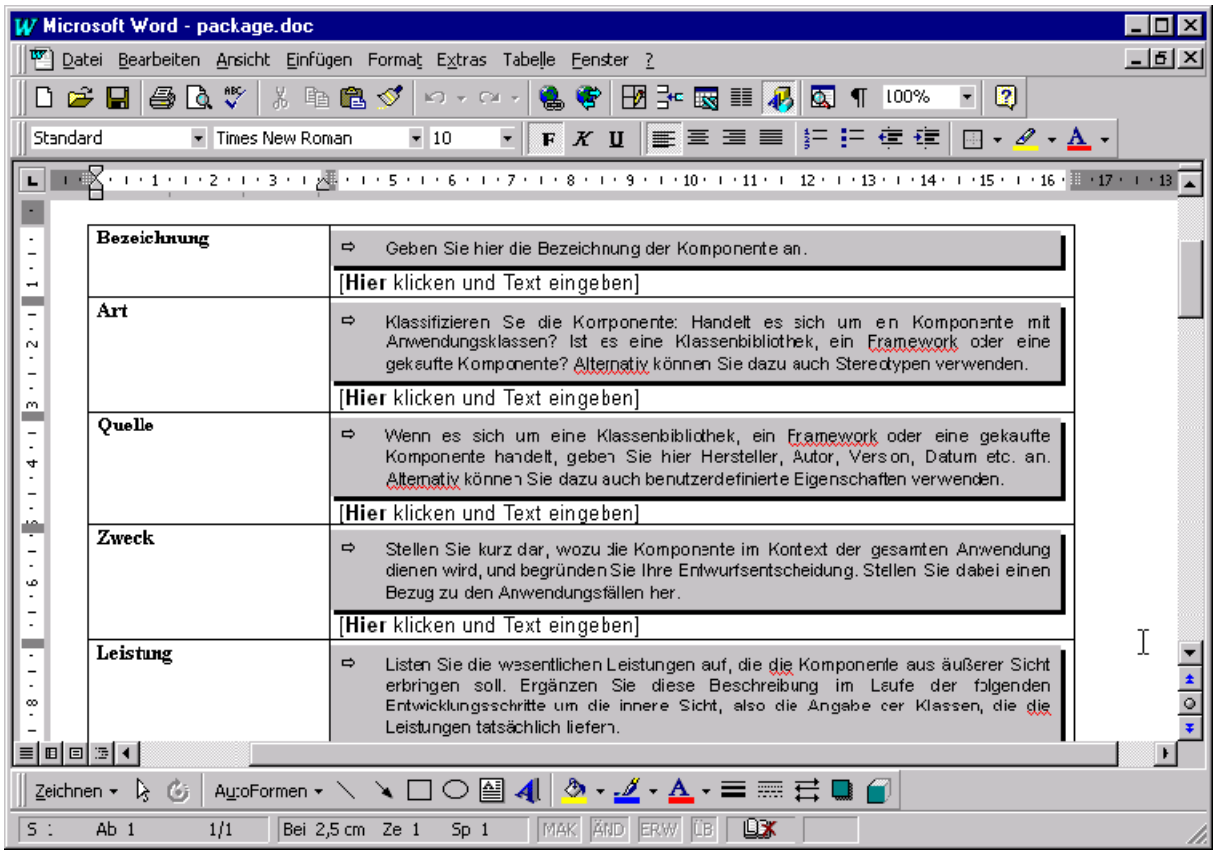


Abb. 7: Beschreibung einer Komponente

4.2.2 Prozeßstruktur

Das Prozeßstruktur-Fenster erlaubt es, die auszuführenden Aktivitäten und die zu produzierenden Produkte in Form von Aktivitätsdiagrammen zu beschreiben. Je nach gewähltem Baustein (in der Systemstruktur) und ausgewählter Sicht (z.B. Projektmanagement oder Software-Entwicklung), erscheint im rechten Teilfenster des Hauptfensters ein entsprechendes Aktivitätsdiagramm (Abb. 8: Aus Platz Gründen worden zwei offene Kontextmenüs abgebildet).

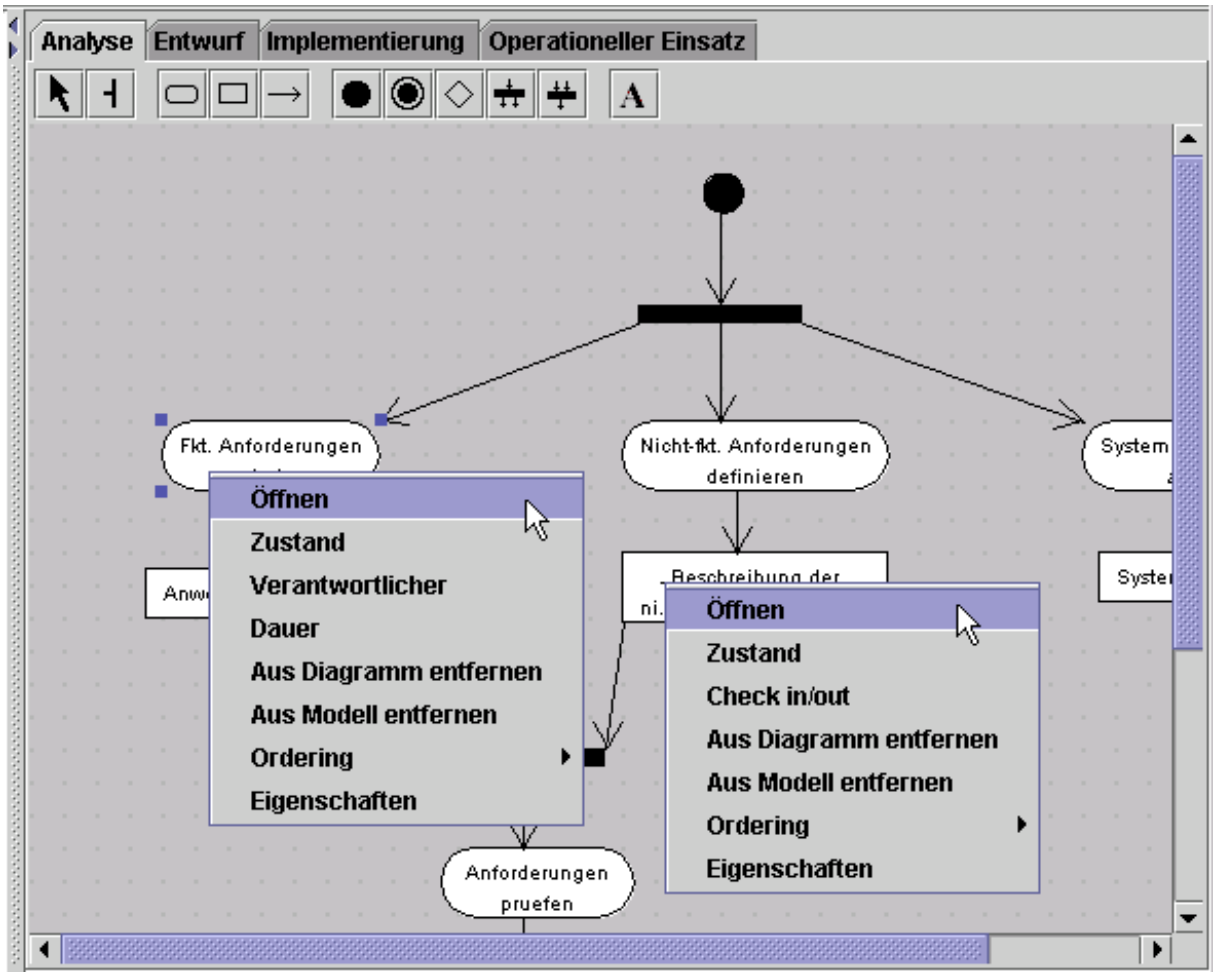


Abb. 8: Prozeßstruktur (Software-Entwicklung: System-Analyse)

Wie man der Abbildung entnehmen kann, wurde wieder ein „Toolbar“ mit integriert, so daß die Aktivitätsdiagramme jeder Zeit erweitert bzw. modifiziert werden können. Damit ist ein projektspezifischer Zuschnitt des EOS-Prozesses für die einzelnen Bausteine und Teilprozesse möglich. Für kleine Projekte kann man viele der Aktivitäts- und Produkttypen löschen bzw. zusammenfassen. Für große Projekte dagegen können weitere Aktivitäts- und Produkttypen eingefügt bzw. vorhandene verfeinert werden.

Alle Elemente der Aktivitätsdiagramme verfügen über Kontextmenüs, die durch das Betätigen der rechten Maustaste aktiviert werden können. Entscheidend sind jedoch die Kontextmenüs für Aktivitäten und Produkte, die in der Abbildung 8 gezeigt werden.

Der Menüpunkt „Öffnen“ soll das Starten eines Werkzeugs symbolisieren, das zur Unterstützung bzw. Verarbeitung einer Aktivität oder eines Produkts benötigt wird. Zum Beispiel wird für die Aktivität „Fkt. Anforderungen analysieren“ ein UML-Werkzeug

benötigt, während für das Betrachten des erzeugten Produkts „Anwendungsfallmodell“ möglicherweise auch ein Word-Viewer ausreichen könnte.

Mit Hilfe des Menüpunkts „Zustand“ kann der aktuelle Zustand einer Aktivität oder eines Produkts ermittelt werden. Für Produkte erscheint der folgende Dialog nach dem Betätigen des Menüpunkts:

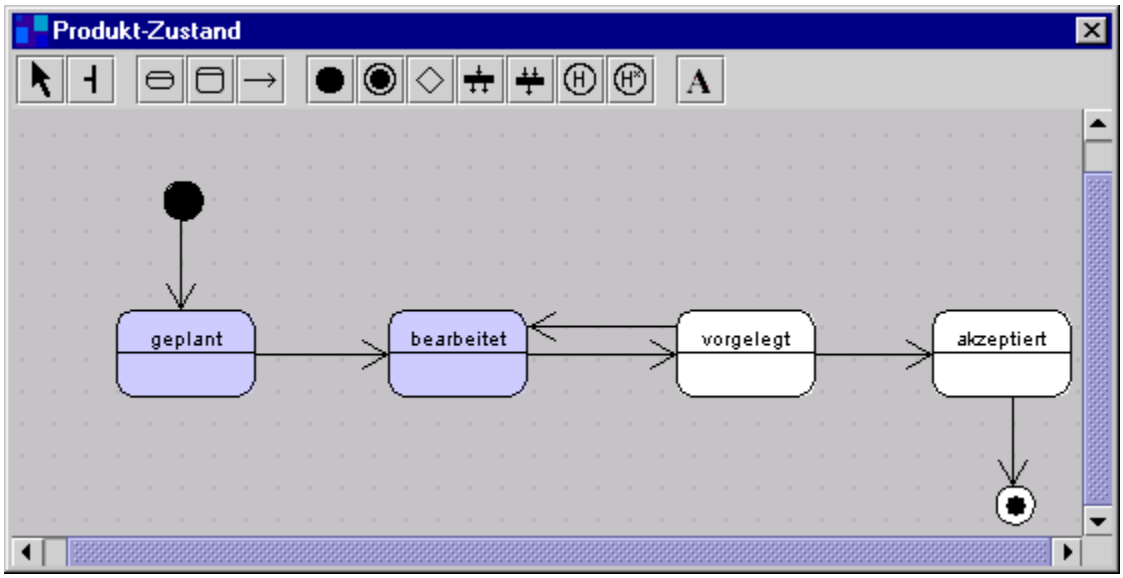


Abb. 9: Produkt-Zustand

Wieder werden durch unterschiedliche Farben die bereits durchlaufenen und die noch zu durchlaufenden Zustände hervorgehoben. Der integrierte „Toolbar“ kann zur Modifikation des Zustanddiagramms genutzt werden.

Die Menüpunkte „Aus Diagramm entfernen“ und „Aus Modell entfernen“ dienen zum Löschen von Aktivitäten und Produkten. Im ersten Fall wird das Element nur aus dem Diagramm gelöscht, kann aber wieder in ein anderes oder in dasselbe Diagramm eingefügt werden. Wird aber „Aus Modell entfernen“ gewählt, so wird das Element permanent gelöscht.

Hinter dem Menüpunkt „Reihenfolge“ (Ordering) verbergen sich weitere Menüpunkte zum Einrichten des Layouts. Zum Beispiel kann ein Element im Vorder- oder Hintergrund eines anderen Elements angezeigt werden.

Durch das Betätigen des Menüpunkts „Eigenschaften“ öffnet sich eine Dialog-Box, in der eine Reihe von Einstellungen vorgenommen werden kann, wie z.B. das Ändern von Farben oder die Definition von Constrains. In der nächsten Abbildung wird ein entsprechender Eigenschafts-Dialog für eine Aktivität gezeigt.

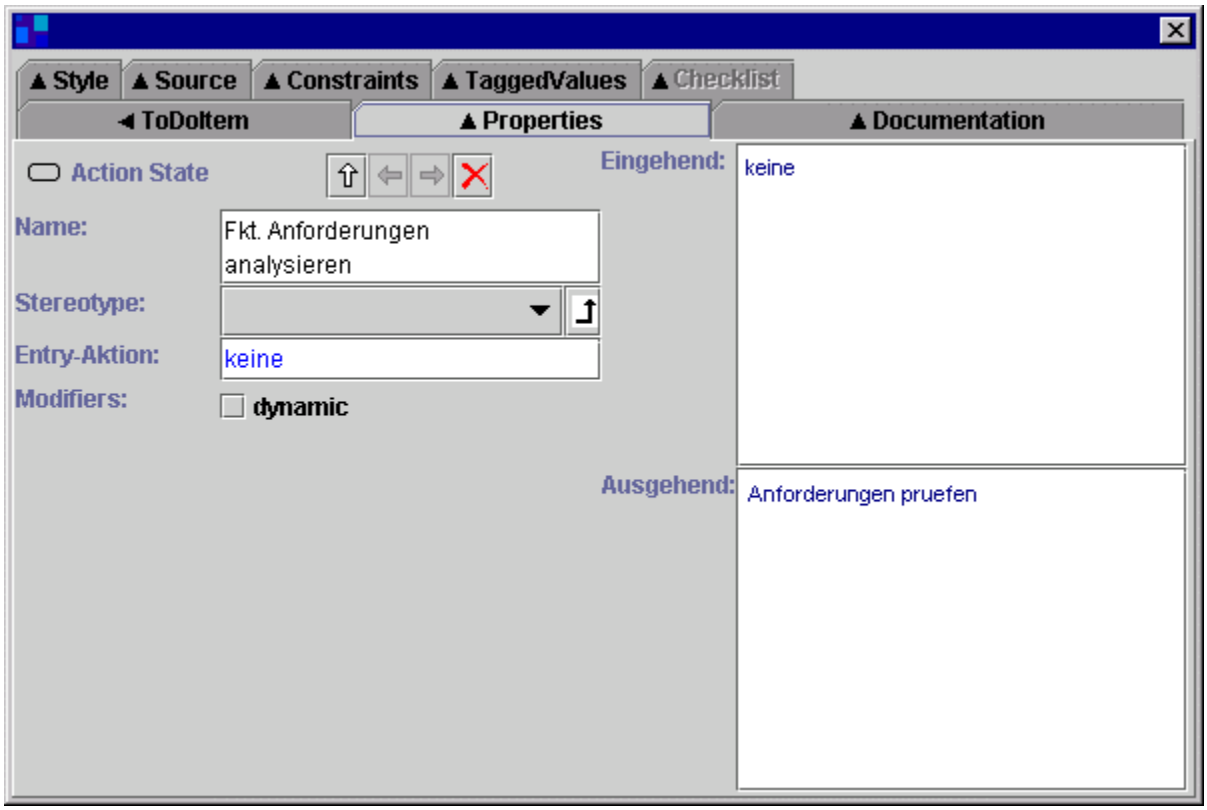


Abb. 10: Eigenschafts-Dialog für die Aktivität „Fkt. Anforderungen analysieren“

Neben den genannten Menüpunkten verfügt eine Aktivität über die Menüpunkte „Verantwortlicher“ und „Dauer“ (siehe Kontextmenü einer Aktivität in der Abbildung 8) . Mit „Dauer“ soll der Zeitaufwand für eine Aktivität angegeben bzw. abgelesen werden können. Nach dem Betätigen des Menüpunkts „Verantwortlicher“ erscheint der Dialog in der Abbildung 11. Dieser Dialog kann verwendet werden, um eine Aktivität einem Projektmitglied zuzuweisen. Da ein Projektmitglied verschiedene Rollen einnehmen kann, wird seine Rolle bezogen auf die gewählte Aktivität festgelegt.

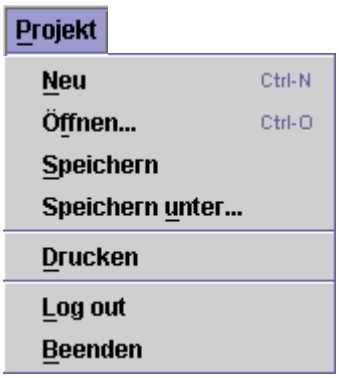
Abb. 11: Dialog zur Zuweisung eines Verantwortlichen

Das Kontextmenü eines Produkts beinhaltet den Menüpunkt „Check in/out“, der entsprechende Funktionen eines Konfigurationsmanagement-Systems simulieren soll. In PMS wird dies durch farbliche Markierung von Produkten dargestellt.

Zur Implementierung der Aktivitätsdiagramme und der oben vorgestellten Zustands-diagramme haben wir Open-Source-Bibliotheken /ArgoUML 0.9.3/ herangezogen, die wir für unsere Zwecke erweitert und modifiziert haben.

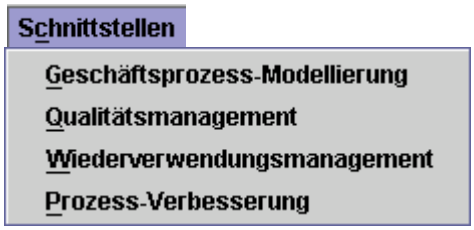
4.2.3 Menüführung

Im folgenden beschreiben wir kurz die einzelnen Menüpunkte von PMS (siehe Abb. 4).

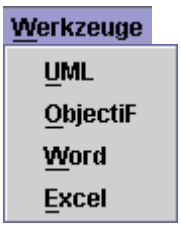


Dieser Menüpunkt dient zum Anlegen, Lesen und Schreiben von Projekten. Die Projektdaten können aus XML-Dateien gelesen bzw. in solche geschrieben. Während für den Systembaum ein eigenes XML-Format definiert wurde, wurde für die UML-Diagramme das standardisierte XML-Format „XMI“ herangezogen. Die Anwendung von „XMI“ ermöglicht die Kompatibilität der Diagramme mit UML-Werkzeugen.

Der Menüpunkt „Bearbeiten“ ermöglicht das Ausführen von Standard-Funktionen, wie z.B. das Kopieren und Wiedereinfügen von Texten oder Diagramm-Elementen.



Wie bereits im zweiten Kapitel beschrieben, verfügt das EOS-Modell über eine Reihe von unternehmensweiten Schnittstellen, die mit diesem Menüpunkt angedeutet werden. Hinter diesen Menüpunkten könnten sich z.B. Werkzeuge oder Richtlinien verbergen.



Dieser Menüpunkt stellt einige Standard-Anwendungen, wie z.B. Excel, Word oder UML-Werkzeuge, zur Verfügung. Durch das Betätigen der Menüpunkte werden die entsprechenden Anwendungen auch tatsächlich gestartet. Die notwendigen Konfigurations-Informationen müssen in der Datei „Config.xml“ im Root-Verzeichnis von PMS definiert werden.

Mit dem Menüpunkt „Optionen“ kann der aktuelle „Look & Feel“ (z.B. Windows- oder Unix-spezifische Oberflächen Darstellung) eingestellt werden. Außerdem können einige administrative Einstellungen vorgenommen werden. Dazu gehört z.B. das Anlegen von Benutzern (vgl. Abb. 12) und die Vergabe von Zugriffsrechten.

Dieser Menüpunkt ruft ein Hilfesystem auf. Da aber kein Hilfesystem hinterlegt ist, erscheint ein Informationsdialog.

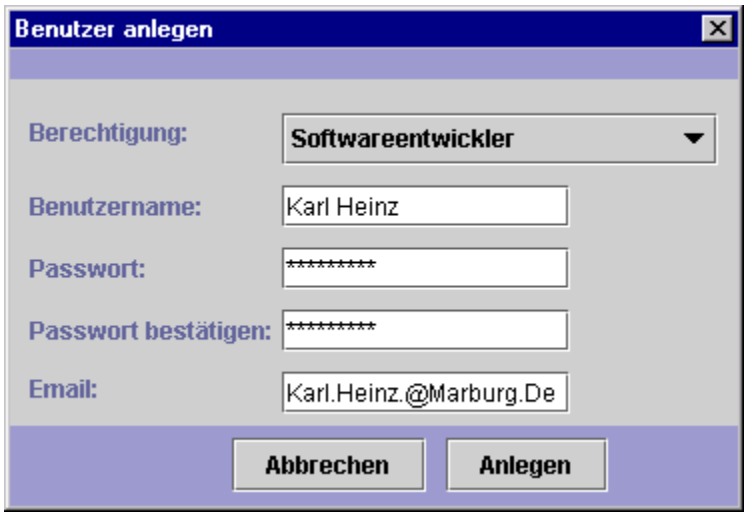
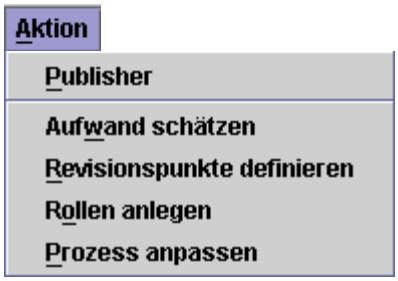


Abb. 12: Dialog zum Anlegen von Benutzern



Der Menüpunkt „Aktion“ wird in Abhängigkeit von der gewählten Prozeßauswahl (z.B. Qualitätssicherung oder Software-Entwicklung) immer neu generiert. Er stellt allgemeine Dienste bereit. In der Abbildung rechts haben wir exemplarisch die Dienste für den Teilprozeß „Projekt-management“ dargestellt, die wir im folgenden näher betrachten wollen.

Menüpunkt: „Aktion/Publisher“

Das Betätigen des Menüpunkts „Aktion/Publisher“ eröffnet den Publisher-Dialog, der in der nächsten Abbildung gezeigt wird.

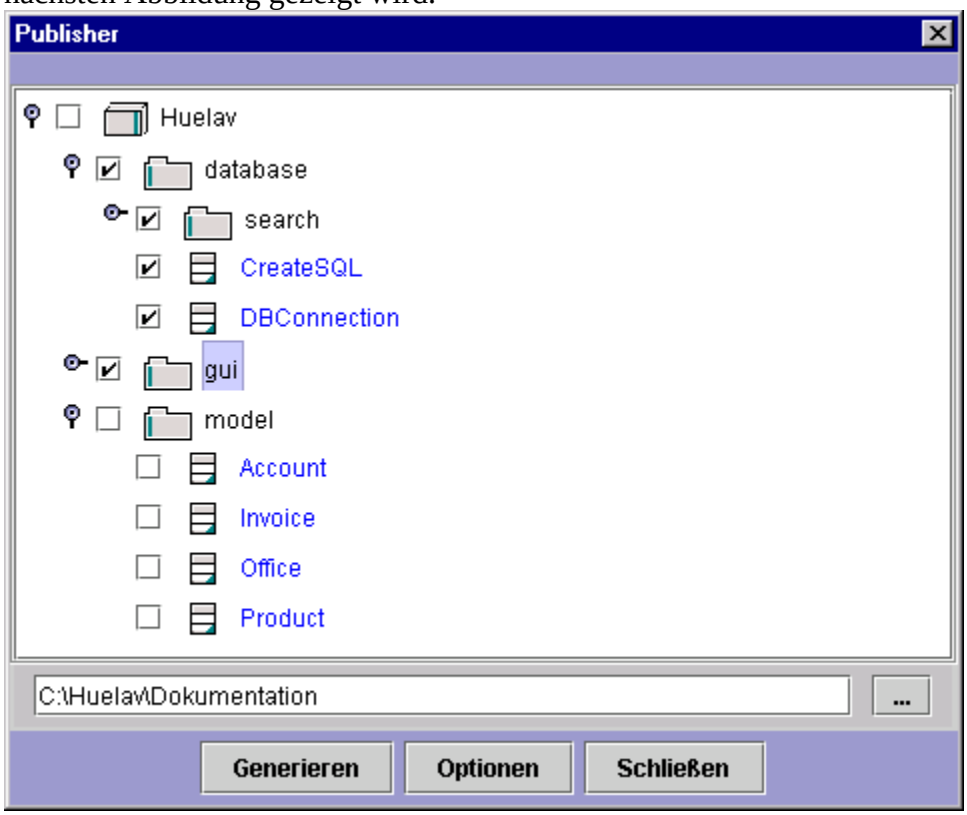


Abb. 13: Publisher-Dialog

Mit Hilfe dieses Dialogs soll das Generieren von Dokumentationen ermöglicht werden. Für die vom Anwender gewählten Bausteine soll aus den vorhandenen Daten (z.B. ausgefüllten Beschreibungsschemata) eine Dokumentation zusammengestellt.

Menüpunkt: „Aktion/Aufwand schätzen“

Zur Aufwandsschätzung von EOS-Projekten haben wir das CEOS-Verfahren vorgeschlagen. Das CEOS-Verfahren und die Integration seiner Implementierung in das UML-Werkzeug „objectiF“ wird in den Abschnitten weiter unten detailliert behandelt. Wird „Aktion/Aufwand schätzen“ betätigt, so wird „objectiF“ mit einem leeren Projekt gestartet. Das Kontextmenü von „objectiF“ haben wir mit dem Menüpunkt „Systembaum aktualisieren“ erweitert (Abb. 14a). Hiermit läßt sich die Systemstruktur von PMS via XML in „objectiF“ übertragen (Abb. 14b). Durch den Einsatz von XML kann die Systemstruktur prinzipiell aber auch in anderen Werkzeugen übertragen werden. Danach kann aus „objectiF“ eine Aufwandsschätzung mit dem von uns implementierten Verfahren CEOS (siehe Abschnitt 4.3) vorgenommen werden.

Abb. 14a: Leeres objectiF-Projekt

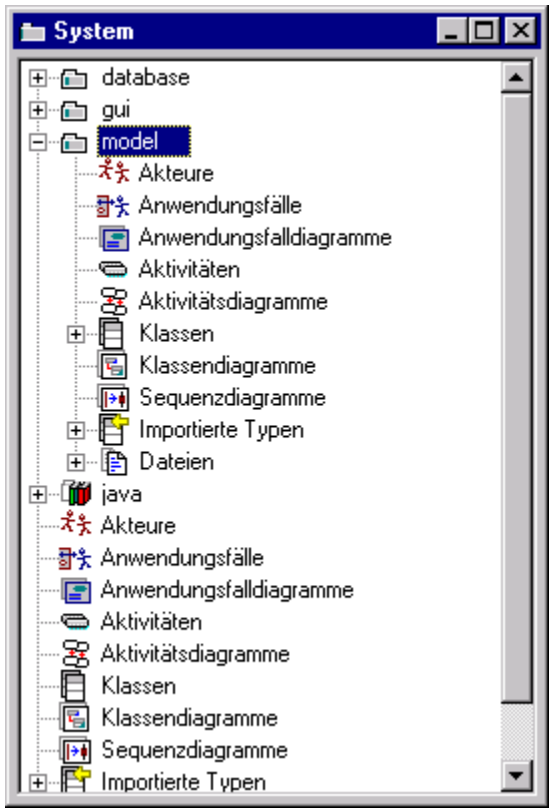


Abb. 14b: Aktualisierter Systembaum

Menüpunkt: „Aktion/Revisionspunkte definieren“

Damit die Ergebnisse der Aufwandsschätzung in PMS weiter verwendet werden können, muß die XML-Export-Funktion von CEOS angestoßen werden. Die Ergebnisse können nämlich so zur Definition von Revisionpunkten verwendet werden (vgl. Subprozeß „Projektmanagement“ im Prozeß-Handbuch). Dazu muß der Menüpunkt „Aktion/Revisionspunkte definieren“ betätigt werden. Es erscheint der in der nächsten Abbildung dargestellte Dialog (Abb. 15). Der Dialog ist als ein Beispiel für die Darstellung und Definition von Revisionspunkten zu verstehen (andere grafisch anspruchsvollere Varianten sind natürlich denkbar).

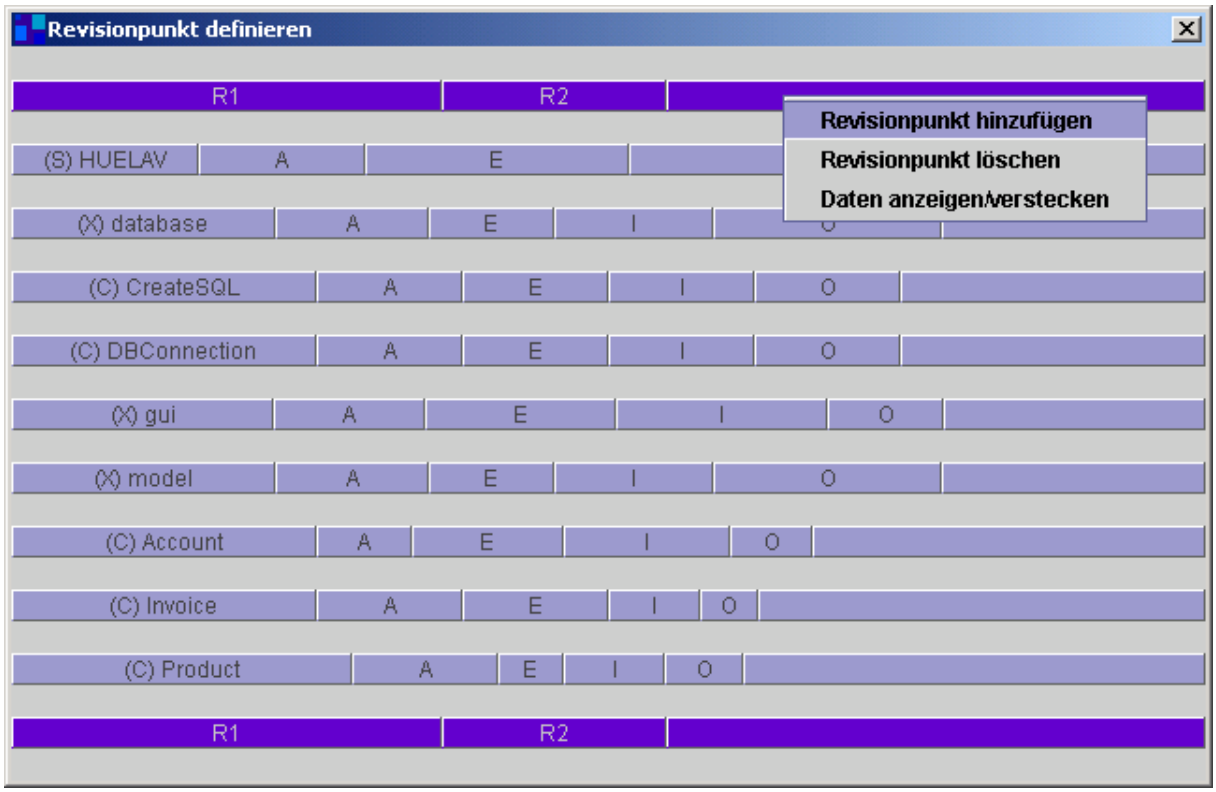


Abb. 15: Definition von Revisionspunkten

Der Aufwand für die einzelnen Phasen der Bausteine wird mit Hilfe von Balken dargestellt. Die Balken sind nicht statisch, so daß eine Korrektur der Aufwandsschätzung durch Verlängern bzw. Verkürzen von Teilbalken vorgenommen werden kann. Der erste und der letzte Balken dienen zur Definition von Revisionspunkten. Diese synchronen „Revisionsbalken“ beinhalten ein Kontextmenü zum Hinzufügen bzw. Löschen von verschiebbaren Revisionspunkten. Zusätzlich können auch die Schätzwerte angezeigt werden („Daten anzeigen/verstecken“).

Menüpunkt: „Aktion/Rollen anlegen“ und „Aktion/Prozeß anpassen“

Der Menüpunkt „Aktion/Rollen anlegen“ soll das Anlegen von Rollen ermöglichen, zusätzlich zu den bereits vordefinierten Rollen, wie z.B. Softwareentwickler oder Projektmanager. Mit Hilfe des Menüpunkts „Aktion/Prozeß anpassen“ kann das Zuschneiden des Prozesses an- bzw. ausgeschaltet werden. Der Menüpunkt bewirkt nämlich, daß die „Toolbars“ der Aktivitätsdiagramme angezeigt bzw. versteckt werden.

4.2.4 Prozeßauswahl

Das EOS-Modell besteht aus den Teilprozessen „Projektmanagement“, „Software-Entwicklung“, „Qualitätssicherung“, „Konfigurationsmanagement“, „Nutzung und Bewertung“. Jeder dieser Teilprozesse erfordert spezielle Methoden- und Werkzeugunterstützung.

Mit der Prozeßauswahl im Statusbalken kann zwischen den Teilprozessen gewechselt werden. Ein Wechsel bewirkt, daß die Prozeßstruktur für den gewählten Teilprozeß in Form eines

Aktivitätsdiagramms angezeigt wird. Außerdem wird der Menüpunkt „Aktion“ neu generiert, so daß die für den entsprechenden Teilprozeß definierten Dienste als Menüpunkte zur Auswahl stehen. Für den Teilprozeß „Projektmanagement“ steht z.B. „Aufwand schätzen“ oder „Rollen anlegen“ zur Verfügung.

Weiter oben wurde gezeigt, wie in PMS mittels Aufwandsschätzung Revisionspunkte definiert werden können (vgl. Abb.15). Die Aufwandsschätzung ist ein Kernmodul von PMS und ein grundlegendes Hilfsmittel für das Projektmanagement. Daher wurde exemplarisch dieses Modul vollständig implementiert. Im folgenden wird zunächst das CEOS-Verfahren zur Aufwandsschätzung von EOS-Projekten vorgestellt /Sarferaz 2000a/, /Sarferaz 2000b/, /Sarferaz, Hesse 2000/. Ausgehend von dem Verfahren wird dann anhand von Benutzeroberflächen die entsprechende Realisierung präsentiert.

4.3 CEOS-Verfahren

Im vorliegenden Abschnitt wird das CEOS-Verfahren (*Cost estimation for EOS-Projects*) zur Aufwandsschätzung von evolutionären, objektorientierten Software-Projekten vorgestellt. Die Grundidee besteht darin, *Komplexitätsmaße* zu definieren, so daß für einzelne Bausteine Komplexitätswerte berechnet werden können. Aus den ermittelten Komplexitätswerten wird mit dem statistischen Verfahren der Regression der Entwicklungsaufwand bestimmt. Die definierten Komplexitätsmaße enthalten eine Reihe von *Koeffizienten*, die detaillierte Aufwandsschätzungen, wie z.B. für einzelne Bausteine, Phasen oder Zyklen, ermöglichen. Die Koeffizienten werden anhand von abgeschlossenen Projekten mit Verfahren aus der Statistik berechnet /Rousseeuw, Aelst 1999/, so daß sie ebenfalls prozeßspezifische Einflußfaktoren berücksichtigen. Das CEOS-Verfahren berücksichtigt objektorientierte Prinzipien und den evolutionären Charakter der Software-Entwicklung. Damit sind Aspekte der Revision, Nutzung und Wartung in dem Verfahren mitverankert.

Es wird von einer linearen Beziehung zwischen der Komplexität von objektorientierter Software und dem zu ihrer Herstellung benötigten Aufwand ausgegangen /Nesi, Querci 1998/. Diese Annahme basiert auf Erfahrungswerten und ist keine Einschränkung, da sie hier zur Berechnung von Koeffizienten benötigt wird, wofür nicht notwendigerweise Linearität vorausgesetzt werden muß. Somit wird das Problem der Aufwandsschätzung auf das Problem der Komplexitätsschätzung reduziert. Der Einflußfaktor Programmumfang fließt indirekt in die Software-Komplexität mit ein, da z.B. die Anzahl der Attribute, Methoden, Klassen und Komponenten berücksichtigt werden.

Neben den Koeffizienten sind die Maße im CEOS-Verfahren von einem Grundmaß m abhängig. Damit wird erreicht, daß die definierten Maße allgemein gehalten werden. Für das Grundmaß m kann ein Komplexitätsmaß, wie z.B. das McCabe-Maß, oder ein Umfangsmaß, wie z.B. LOC (Lines Of Code), eingesetzt werden Es wird zwischen Maßen für die *Anwendungs*- und *Definitionskomplexität* von Bausteinen unterschieden. Mit Hilfe der Maße für die Anwendungskomplexität werden Vererbungs-, Assoziations- und Aggregations-beziehungen bewertet. Die Komplexität von Bausteinen wird anhand der Maße für die Definitionskomplexität berechnet. Die Tabelle in der Abbildung 16 gibt eine Übersicht über die im CEOS-Verfahren definierten Maße.

Maß	Bezeichnung
Attribut – Ebene	
AC_m^{App}	Attribute Application Complexity
AC_m^{Def}	Attribute Definition Complexity
Methoden – Ebene	
MC_m^{App}	Method Application Complexity
MC_m^{Def}	Method Definition Complexity
MBC_m^{App}	Method Body Complexity (Application)
MBC_m^{Def}	Method Body Complexity (Definition)
MIC_m^{App}	Method Interface Complexity (Application)
MIC_m^{Def}	Method Interface Complexity (Definition)
NMLA	Number of Method Local Attributes
Klassen – Ebene	
CC_m^{App}	Class Application Complexity
CC_m^{Def}	Class Definition Complexity
CCA_m^{App}	Class Application Complexity during Analysis activity
CCA_m^{Def}	Class Definition Complexity during Analysis activity
CCD_m^{App}	Class Application Complexity during Design activity
CCD_m^{Def}	Class Definition Complexity during Design activity
CCI_m^{App}	Class Application Complexity during Implementation activity
CCI_m^{Def}	Class Definition Complexity during Implementation activity
CU_m	Class Usability
IC_m	Inherited Class Complexity (analysis)
ICC_m	Inherited Class Complexity (design)
$InCC_m$	Inherited Class Complexity (implementation)
LC_m	Local Class Complexity (analysis)
LCC_m	Local Class Complexity (design)
$LoCC_m$	Local Class Complexity (implementation)
NDP	Number of Direct Parents
NIC	Number of Inherited Classes
NLA	Number of Local Attributes
NLM	Number of Local Methods
NPuCA	Number of Public Class Attributes
NPuCM	Number of Public Class Methods
Komponenten – Ebene	
NXC	Number of Component Classes
XC_m	Component Complexity
XCA_m	Component Complexity during Analysis activity
XCA_r	Component Complexity coarse Analysis
XCD_m	Component Complexity during Design activity
XCI_m	Component Complexity during Implementation activity
XU_m	Component Usability
System – Ebene	
NSX	Number of System Components
SCA_m	System Complexity during Analysis phase
SCA_r	System Complexity coarse Analysis
SCD_m	System Complexity during Design phase
SCI_m	System Complexity during Implementation phase

Abb. 16: Übersicht über die definierten Maße

4.3.1 Komplexitätsmaße

4.3.1.1 Attribut-Ebene

In diesem Abschnitt werden die Maße AC_m^{App} und AC_m^{Def} definiert. Mit diesen Maßen kann die Komplexität eines Attributes bzgl. eines Grundmaßes m gemessen werden.

Definition I (Attribute Application Complexity): $AC_m^{App} = K$

Sei *A : ClassA* ein Attribut vom Typ „*ClassA*“, dann berechnet sich AC_m^{App} wie folgt:
$AC_m^{App}(A) = K$, wobei $K \in IR^+$ eine Konstante ist.

Definition II (Attribute Definition Complexity): $AC_m^{Def} = CC_m^{App}$ (ClassA)

Sei *A : ClassA* ein Attribut vom Typ „*ClassA*“, dann berechnet sich AC_m^{Def} wie folgt:
$AC_m^{Def}(A) = CC_m^{App}(ClassA)$.

CC_m^{App} ist ein Maß für die Klassen-Anwendungskomplexität, die weiter unten definiert wird.

Eine Klasse kann sehr einfach oder sehr komplex und umfangreich sein, was sich auf die Komplexitätsschätzung auswirken sollte. Daher dürfen Attribute, die sich auf unterschiedlich komplexe Klassen beziehen, nicht alle gleich behandelt werden. Dieser Sachverhalt wird mit dem Maß AC_m^{Def} berücksichtigt.

Für die Berechnung des Maßes CC_m^{App} muß natürlich die Komplexität der Attribute dieser Klasse berücksichtigt werden, so daß prinzipiell ein rekursiver Zusammenhang vorliegen kann. Eine möglicherweise zirkuläre Berechnung wird aber vermieden, indem bei der Definition von CC_m^{App} auf das Maß AC_m^{App} zurückgegriffen wird.

4.3.1.2 Methoden-Ebene

Für die Komplexität einer Methode werden die folgenden Maße definiert:

$MC_m^{App} = MIC_m^{App} + MBC_m^{App}$ und
$MC_m^{Def} = w_{MICm}\, MIC_m^{Def} + w_{MBCm}\, MBC_m^{Def}$.

Mit MIC_m^{App} bzw. MIC_m^{Def} läßt sich die Komplexität der Signatur einer Methode messen. Das Maß MBC_m^{App} bzw. MBC_m^{Def} berechnet die Komplexität für die Implementierung einer Methode. Die Koeffizienten w_{MICm} und w_{MBCm} sind Gewichtungen, die u.a. vom Grundmaß m abhängen. Die Koeffizienten werden mit Hilfe der statistischen Technik der *Regression* /Rousseeuw, Aelst 1999/ aus abgeschlossenen Projekten ermittelt.

Definition I (*Method Application Complexity*): $MC_m^{App} = MIC_m^{App} + MBC_m^{App}$

(1) MIC_m^{App} (Method *Interface* Complexity (*Application*))

Sei $F(A_1{:}C_1, ..., A_n{:}C_n) : C_{n+1}$ eine Methoden-Signatur, wobei F einen Methoden-Bezeichner entspricht, A_i einen Attributbezeichner und C_i eine Klasse repräsentiert. A_{n+1} sei ein Pseudoattribut vom Typ C_{n+1}, das für das Methoden-Ergebnis steht. MIC_m^{App} läßt sich dann wie folgt berechnen:

$$MIC_m^{App} = \sum_{i=1}^{n+1} AC_m^{App}(A_i)$$

(2) MBC_m^{App} (Method *Body* Complexity (*Application*))

Sei NMLA die Anzahl der Variablen, die in der Methode lokal definiert sind und m ein Grundmaß, dann läßt sich MBC_m^{App} wie folgt berechnen (A_i als lokale Variable):

$$MBC_m^{App} = \sum_{i=1}^{NMLA} AC_m^{App}(A_i) + m$$

Definition II (*Method Definition Complexity*): $MC_m^{Def} = w_{MICm}\, MIC_m^{Def} + w_{MBCm}\, MBC_m^{Def}$

Die Maße MIC_m^{Def} und MBC_m^{Def} lassen sich analog wie die Maße MIC_m^{App} und MBC_m^{App} definieren, indem statt AC_m^{App} das Maß AC_m^{Def} verwendet wird.

4.3.1.3 Klassen-Ebene

An einer Klasse werden die Tätigkeiten Analyse, Entwurf, Implementierung und operationeller Einsatz ausgeführt. Da von Tätigkeit zu Tätigkeit mehr Daten über eine Klasse zur Verfügung stehen, werden mehrere Maße definiert, so daß unterschiedliche Schätz-Zeitpunkte berücksichtigt werden können. Die Definitionen der Maße von der Analyse bis zur Implementierung werden stets verfeinert, um die jeweils vorliegenden Daten auch ausschöpfen zu können.

Phase Analyse

Es wird nun das Maß CCA_m^{Def} definiert, mit dem eine erste Gesamtschätzung während der Analysetätigkeit vorgenommen werden kann.

Definition I (Class *Application* Complexity during *Analysis* activity):

$CCA_m^{App} = NLA + NLM$

Mit NLA ist die Anzahl der Klassenattribute und mit NLM die Anzahl der Klassenmethoden gemeint.

Definition II (Class *Definition* Complexity during *Analysis* activity): $CCA_m^{Def} = LC_m + IC_m$

(1) LC_m (*Local* Class Complexity (analysis))

$LC_m = w_{NLAm}\, NLA + w_{NLMm}\, NLM$

(2) IC_m (*Inherited* Class Complexity (analysis))

$IC_m = w_{NICm}\, NIC,$

wobei mit NIC die Anzahl der beerbten Klassen gemeint ist.

Phase Entwurf

Beim Entwurf einer Klasse wird die Klassenstruktur grob festgelegt, d.h. es werden zumindest die Signatur der Methoden und die Typen der Klassenattribute bekannt sein. Außerdem werden auch die Beziehungen zu anderen Klassen feststehen. Diese Informationen werden genutzt, um eine genauere Komplexitätsschätzung für eine Klasse vorzunehmen.

Es wird wieder zwischen der „inneren" Klassenkomplexität LCC_m und der „beerbten" Komplexität ICC_m unterschieden. Mit LCC_m wird die Anzahl der Klassenattribute- und methoden sowie deren Komplexität betrachtet. Diese sind nämlich Indikatoren für den Entwicklungs- und Wartungsaufwand. Zudem beeinflussen diese Faktoren auch die Systemgröße. Die Vererbungskomplexität wird mit dem Maß ICC_m berücksichtigt. Je größer die Vererbungskomplexität ist, desto schwieriger gestaltet sich der Klassentest und die Klassen-Integration.

Da die Komplexität von Attributen und Methoden in das Maß eingeht, wird auch die Kopplungs- bzw. Schnittstellenkomplexität beachtet. Je höher diese Komplexität ist, desto größer wird der Test-, Integrations- und Wartungsaufwand. Durch die Kopplungskomplexität werden Assoziations- und Aggregationsaspekte berücksichtigt.

Definition I (Class *Application* Complexity during *Design* activity): CCD_m^{App}

Sei A_i und M_i das i-te Attribut bzw. Methode.

$$CCD_m^{App} = \sum_{i=1}^{NLA} AC_m^{App}(A_i) + \sum_{i=1}^{NLM} MIC_m^{App}(M_i)$$

Das Maß NLA ist die Anzahl der Klassenattribute, während NLM die Anzahl der Klassenmethoden beschreibt. Die Maße AC_m^{App} und MIC_m^{App} sind bereits definiert worden.

Definition II (Class Definition Complexity during *Design* activity): $CCD_m^{Def} = LCC_m + ICC_m$

Sei A_i, M_i und C_i das i-te Attribut, Methode und Klasse.

(1) LCC_m (*Local* Class Complexity (design))

$$LCC_m = w_{LACm} \sum_{i=1}^{NLA} AC_m^{Def}(A_i) + w_{LMCm} \sum_{i=1}^{NLM} MIC_m^{Def}(M_i)$$

(2) ICC_m (*Inherited* Class Complexity (design))

$$ICC_m = w_{ICCm} \sum_{i=1}^{NDP} CCD_m^{App}(C_i) + NIC(C_i)$$

Das Maß NDP steht für die Anzahl der direkten Superklassen.

Phase Implementierung

Während der Implementierung einer Klasse stehen weitere Informationen zur Verfügung, anhand derer noch detailliertere Komplexitätsschätzungen vorgenommen werden können. Zum Beispiel steht die Implementierung der Methoden zur Verfügung, so daß das McCabe

Maß berechnet werden kann. Die innere Struktur der Klassen und die Beziehung zu anderen Klassen sind inzwischen stabil, was zu differenzierteren Komplexitätsschätzungen führt.

Definition I (Class *Application* Complexity during *Implementation* activity): CCI_m^{App}

Sei A_i und M_i das i-te Attribut und Methode einer Klasse.

$$CCI_m^{App} = \sum_{i=1}^{NLA} AC_m^{App}(A_i) + \sum_{i=1}^{NLM} MC_m^{App}(M_i)$$

Definition II (Class *Definition* Complexity during *Implementation* activity):

$CCI_m^{Def} = LoCC_m + InCC_m$

Sei A_i, M_i und C_i das i-te Attribut, Methode und Klasse.

(1) $LoCC_m$ (*Local* Class Complexity (implementation))

$$LoCC_m = w_{LoACm} \sum_{i=1}^{NLA} AC_m^{Def}(A_i) + w_{LoMCm} \sum_{i=1}^{NLM} MC_m^{Def}(M_i)$$

(2) $InCC_m$ (*Inherited* Class Complexity (implementation))

$$InCC_m = w_{InCCm} \sum_{i=1}^{NDP} CCI_m^{App}(C_i) + NIC(C_i)$$

Phase Operationeller Einsatz

Bei dieser Tätigkeit werden Aspekte der Erprobung, Nutzung und Revision behandelt. Die Komplexität der Tätigkeit des operationellen Einsatzes wird neben der Quellcode-Komplexität durch die Koeffizienten berücksichtigt. Wie oben erwähnt, werden die Koeffizienten aus abgeschlossenen Projekten gewonnen, so daß die Faktoren, die die Tätigkeit des operationellen Einsatzes beeinflussen, einkalkuliert werden. Die für die Komplexitätsberechnung notwendigen Messtätigkeiten bestehen hier aus dem Korrigieren der Werte für Koeffizienten, aus dem Protokollieren von Schätzergebnissen und möglichen Neukalkulationen auf Attribut- oder Methodenebene, die sich u.a. aufgrund der Wartung und Pflege ergeben.

Wiederverwendung auf der Klassenebene

Eine Klasse kann entweder neu implementiert werden, oder sie wird einer Bausteinsbibliothek entnommen und wiederverwendet. Sie kann aber auch durch Spezialisierung von Klassen aus der Bausteinbibliothek entstehen. Wie die Klassenkomplexität bei einer *Neuimplementierung* ermittelt wird, ist bereits weiter oben beschrieben worden. Im folgenden werden noch die Aspekte der *vollen Wiederverwendung* und der *Wiederverwendung durch Spezialisierung* dargestellt.

Sowohl bei der vollen Wiederverwendung als auch bei der Wiederverwendung durch Spezialisierung muß vorerst verstanden werden, wie die Klassen zu benutzen sind. Daher wird der Grad der Verständlichkeit und Benutzbarkeit einer Klasse durch das Maß CU_m ermittelt.

Das Maß CU_m berechnet den Grad der Benutzbarkeit und Verständlichkeit einer Klasse, indem die Komplexität des *öffentlichen Teils* einer Klasse mit einem Koeffizienten multipliziert wird. Denn um eine Klasse benutzen zu können, müssen seine Dienste, d.h. seine öffentlichen Teile, verstanden worden sein.

Definition (*Class Usability*): CU_m

Sei A_i und M_i das i-te Attribut bzw. Methode.

$$CU_m = w_{CU} \left(\sum_{i=1}^{NPuCA} AC_m^{App}(A_i) + \sum_{i=1}^{NPuCM} MC_m^{App}(M_i) \right)$$

Mit den Maßen NPuCA und NPuCM sind die Anzahl der öffentlichen (z.B. „public“ und „protected“ deklarierte) Klassenattribute und –methoden gemeint. w_{CU} ist wieder ein Koeffizient.

Je niedriger der Wert von CU_m ist, desto leichter ist die entsprechende Klasse zu benutzen. Der Koeffizient w_{CU} wird aus alten Projekten einer Firma berechnet. So werden Aspekte, wie z.B. der Suchaufwand in der Bausteinbibliothek oder die Qualität der Dokumentation, berücksichtigt.

Mit Hilfe von CU_m kann jetzt die Komplexität einer Klasse CC_m^{Def} allgemein betrachtet werden (CC_m^{App} läßt sich analog definieren, indem „Def“ durch „App“ ersetzt wird) . Sei *C* eine beliebige Klasse, dann gilt:

$$CC_m^{Def}(C) = \begin{cases} \text{Konstant}, & \text{falls } C \text{ eine Systemklasse ist} \\ CCA_m^{Def}(C), & \text{falls } C \text{ analysiert wird} \\ CCD_m^{Def}(C), & \text{falls } C \text{ entworfen wird} \\ CCI_m^{Def}(C), & \text{falls } C \text{ implementiert wird} \\ CU_m(C), & \text{falls } C \text{ vollständig wiederverwendet wird} \end{cases}$$

Falls eine Klasse *C* durch Spezialisierung aus einer Klasse *S* (*S* sei aus der Bausteinbibliothek entnommen) entsteht, wird ihr CU_m (*S*) als Vererbungskomplexität zugewiesen. Denn die Superklasse *S* muß zunächst studiert und verstanden werden. Bei der Assoziations- und Aggregationsbeziehung mit einer wiederverwendeten Klasse *C* geht statt der Anwendungskomplexität der Wert $CU_m(C)$ in die Berechnung ein.

Anwendung des Maßes CCD_m^{Def}

Die Anwendung des Maßes CCD_m^{Def} wird nun anhand eines Beispiels illustriert. Beim Entwurf von Klassen wird die statische Struktur modelliert und mit Hilfe von Klassendiagrammen beschrieben. Diese Informationen werden genutzt, um die Komplexität von Klassen zu berechnen. Als Beispiel betrachten wir ein vereinfachtes Klassendiagramm für ein System zur Kontoverwaltung.

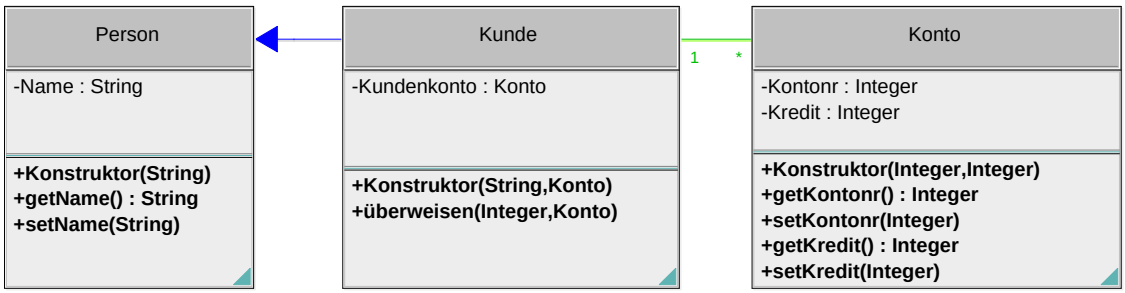


Abb. 17: Ein einfaches Klassendiagramm

Es wird nun exemplarisch die Definitionskomplexität der Klasse „Kunde“ berechnet. Den Elementartypen „String“ und „Integer“ wird die Komplexität K = 1 zugewiesen.

CCD_m^{Def} (Kunde) = LCC_m (Kunde) + ICC_m (Kunde)
LCC_m (Kunde)
= w_{LACm} AC_m^{Def} (Konto) + w_{LMCm} (MIC_m^{Def} (Konstruktor) + MIC_m^{Def} (überweisen))
= w_{LACm} CC_m^{App} (Konto) + w_{LMCm} (AC_m^{Def} (String) + AC_m^{Def} (Konto) +
AC_m^{Def} (Integer) + AC_m^{Def} (Konto))
= w_{LACm} * 8 + w_{LMC} *(1 + 8 + 1 + 8) = w_{LACm} * 8 + w_{LMC} * 18
ICC_m (Kunde)
= w_{ICCm} (CCD_m^{App} (Person) + NIC (Person))
= w_{ICCm} (AC_m^{App} (Name) + MIC_m^{App} (Konstruktor) + MIC_m^{App} (getName) +
MIC_m^{App} (setName) + 0)
= w_{ICCm} (CC_m^{App} (String) + AC_m^{App} (String)) + AC_m^{App} (String))
= w_{ICCm} (1 + 1 + 1 + 1) = w_{ICCm} * 4

CCD_m^{Def} (Kunde) = w_{LACm} * 8 + w_{LMCm} * 18 + w_{ICCm} * 4

4.3.1.4 Komponenten-Ebene

Für die einzelnen Phasen einer Komponenten werden ebenfalls Komplexitätsmaße definiert. Da diese Maße eine ähnliche Struktur haben, wird im folgenden eine zusammenfassende Definition angegeben.

Definition (Component Complexity during *Analysis/Design/Implementation* activity): XCY_m

Sei C_i die i-te Klasse einer Komponente.

$XCY_m = w_{XYm} \sum_{i=1}^{NXC} CCY_m^{Def}(C_i)$, wobei Y = A(nalysis), D(esign), I(mplementation)

Ist nur die ungefähre Anzahl NXC der Klassen bekannt, kann die Komponentenkomplexität durch die Multiplikation eines Faktors mit NXC geschätzt werden. Dies wird mit dem Maß XCA_r geleistet (*grobe Schätzung*).

Definition (Component Complexity *coarse Analysis*): XCA_r

$XCA_r = w_X$ NXC

Für wiederverwendete Komponenten wird das folgende Maß definiert.

Definition (Component Usability): XU_m

Sei *X* ein beliebige Komponente und C_i die i-te Klasse der Komponente *X*.

$$XU_m(X) = w_{XU} \sum_{i=1}^{NXC} CU_m(C_i)$$

NXC steht für die Anzahl der Klassen einer Komponente und w_{XU} ist ein Koeffizient.

4.3.1.5 System-Ebene

Zuletzt wird das gesamte System betrachtet und hierfür Komplexitätsmaße definiert. Auf der System-Ebene wird noch ein *Qualitäts-Einflußfaktor* eingeführt, mit der die Qualitäts-Anforderungen an das System berücksichtigt werden.

Ist am Anfang nur die ungefähre Anzahl der Komponenten NSX (Number of System Components) eines Systems bekannt, kann diese als Grundlage für eine grobe Komplexitätsschätzung genutzt werden. Daher wird zunächst das Maß SCA_r definiert, das solche erste Schätzungen ermöglicht. Ist später über die einzelnen Komponenten mehr bekannt, so kann das Maß XCY_m genutzt werden.

Definition (System Complexity during *Analysis/Design/Implementation* phase): SCY_m

Sei X_i die i-te Komponente eines Systems.

$$SCY_m = w_{SYm}\, w_{Qualität} \sum_{i=1}^{NSX} XCY_m(X_i)\text{, wobei Y = A(nalysis), D(esign), I(mplementation)}$$

w_{SA} ist wieder ein Koeffizient und $w_{Qualität}$ ist ein Qualitäts-Einflußfaktor, der ähnlich wie in /Sneed 1996/ berechnet wird.

Definition (System Complexity *coarse Analysis*): SCA_r

$$SCA_r = w_S\, w_{Qualität}\, NSX$$

NSX steht hierbei für die Anzahl der Komponenten eines Systems und w_S ist ein Koeffizient, was der durchschnittlichen Komplexität einer Komponente entspricht.

4.3.1.6 Koeffizienten-Sätze für Zyklen und Tätigkeiten

Für alle Bausteine wurden eine Reihe von Koeffizienten eingeführt, die mit der statistischen Technik der Regression bestimmt werden. Für jede der Tätigkeiten Analyse, Entwurf und Implementierung wurden unterschiedliche Koeffizienten gewählt. Die Koeffizienten berücksichtigen *Prozeß-Einflußfaktoren* und federn die Komplexitätsschätzung ab.
Bis jetzt wurde von einem einzigen Klassenzyklus ausgegangen und es wurden Koeffizienten vorausgesetzt, mit denen jeweils die Gesamtkomplexität eines Bausteins abgeschätzt werden konnte. Mit Hilfe der Koeffizienten können neben Komplexitätsschätzungen für den gesamten Entwicklungszyklus eines Bausteins auch Schätzungen für die einzelnen Tätigkeiten vorgenommen werden. Die Datenerfassung aus abgeschlossenen Projekten muß lediglich auf die einzelnen Tätigkeiten bezogen sein und nicht auf den Gesamtaufwand. Die Regression

braucht dann nur mit solchen Daten ausgeführt zu werden und man erhält einen Koeffizientensatz für den Aufwand der einzelnen Tätigkeiten.

Wie schon oben erwähnt, wurde zunächst ein einziger Zyklus betrachtet. Eine Beschränkung auf einen Zyklus ist aber nicht zwingend, da auch p Zyklen (p > 0 ist ein Erfahrungswert) betrachtet werden können. In diesem Fall müssen aber die Koeffizientensätze der einzelnen Maße für die jeweiligen Zyklen entsprechend abgestimmt sein. Eine einfache Variante für die Wahl der Koeffizientensätze wäre, daß der p-te Anteil des Koeffizientensatzes für einen Zyklus betrachtet wird. Eine verfeinerte, aber aufwendigere Methode wäre, daß für jede der p-Zyklen durch die Regression ein Koeffizientensatz bestimmt wird.

4.3.2 Bestimmen der Koeffizienten im CEOS-Verfahren

Um das CEOS-Verfahren flexibel zu halten, wurde eine Reihe von Koeffizienten in die Maße eingeführt. Die Koeffizienten werden anhand der statistischen Methode der *robusten Regression* bestimmt. Für CEOS wurde das Verfahren LMS (Least Median Squares) gewählt, das aus theoretischer Sicht eines der bestmöglichen Verfahren ist. Im folgenden werden die mathematischen Grundlagen der Regression vorausgesetzt. Wir beschränken uns hier auf die Darstellung des LMS-Verfahren.

4.3.2.1 Regressionsschätzer

Gegenwärtig ist die Methode der kleinsten Quadrate (LS = Least Squares) das am häufigsten verwendete Verfahren zur Regressionsanalyse (Details zum LS-Verfahren können z.B. bei /Bosch 1997/ nachgelesen werden). Der Grund hierfür ist, daß das Verfahren sehr einfache Berechnungen erlaubt und eine lange Tradition hat. Reale Daten enthalten aber sehr oft *Ausreißer*. Werden nun Verfahren wie LS eingesetzt, obwohl Ausreißer vorliegen, kann die Schätzung verfälscht werden. Außerdem bleiben die Ausreißer verborgen und können nicht identifiziert werden. Um diese Probleme zu vermeiden, wurden robuste Techniken entwickelt. Bei der robusten Regression werden Regressionsschätzer entworfen, die resistent gegenüber Ausreißern sind. So wie bei der linearen Regression, gibt es bei der robusten Regression auch sehr viele Regressionsschätzer, so daß eine Auswahl getroffen werden muß. Für unsere Zwecke werden wir den Regressionsschätzer LMS (Least Median of Squares) verwenden, der wie folgt definiert ist:

Least Median of Squares (LMS)

Sei n die Anzahl der Datensätze.

(1.1) $\min \underset{i=1,\ldots,n}{med} \varepsilon_i^2$, wobei $\varepsilon_i = y_i - y_i^*$, $y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}$,

$y_i^* = \beta_1^* x_{i1} + \beta_2^* x_{i2} + \ldots + \beta_p^* x_{ip}$,

$\beta_k, \beta^*_{k,} x_{ik} \in IR$ und $k, p \in IN^+$

med bezeichnet den Median.

Der LMS-Regressionsschätzer wurde von P. J. Rousseeuw vorgestellt und ist beweisbar einer der besten Regressionsschätzer /Rousseeuw 1984/. Die Idee beim LMS ist einfach: Statt die

Residuen zu summieren, wird deren Median berechnet. Der LMS-Regressionsschätzer ist gegenüber von Ausreißern „resistenter“ als LS und kann unabhängig von der Anzahl der Einflußgrößen x_{ik} (k = 1,..., p) bis zu n/2 Ausreißer korrigieren. Damit ist die geschätzte Regressionsgerade selbst dann brauchbar, wenn fast 50% der Daten „verschmutzt“ sind. Das ist aber zugleich die theoretische Obergrenze. Denn wenn mehr als 50% der Daten „verschmutzt“ sind, kann nicht mehr zwischen „guten“ und „schlechten“ Daten unterschieden werden.

4.3.2.2 LMS-Algorithmus

Der Regressionsschätzer LMS bringt eine ganze Reihe von Vorteilen, dafür gestaltet sich aber die Berechnung schwieriger. Damit die Regressionskoeffizienten ermittelt werden können, muß ein Approximationsprozeß durchlaufen werden. Daher ist es nicht wie bei LS möglich, geschlossene Formeln für die Berechnung der Koeffizienten anzugeben.

Im folgenden werden wir den Algorithmus für die Berechnung der Koeffizienten beschreiben und anhand eines Beispiels veranschaulichen. Der Algorithmus wurde in /Rousseeuw 1984/ und /Rousseeuw, Leroy 1987/ vorgestellt.

Für den Algorithmus werden zunächst p von n Datensätzen betrachtet ($p \leq n$) und mit $J = \{i_1, ..., i_p\}$ indiziert. Mittels der p Datensätze kann ein Gleichungssystem mit p Unbekannten gelöst werden. Daher kann die Regressionsfunktion (mit p Einflußgrößen) durch die p Datenpunkte bestimmt werden. Die zu der Regressionsfunktion zugehörigen Koeffizienten bezeichnen wir mit dem Vektor β_J. Man betrachtet nun die Regressionsfunktion bezüglich der n Datensätze, indem der Wert

$$\underset{i=1,...,n}{med}(y_i - x_i \beta_J)^2$$

berechnet wird. Auf diese Weise erhält man n-Werte, deren Minimum das Ergebnis darstellt.

Es stellt sich natürlich die Frage, wie viele Teildatensätze $J_1, ..., J_m$ betrachtet werden sollen. Prinzipiell können alle möglichen Teildatensätze untersucht werden. Die Zahl der Teildatensätze steigt aber mit wachsendem n und p sehr schnell, so daß in der Praxis nur zufällig oder systematisch gewählte Teildatensätze in Frage kommen. Werden die Teildatensätze zufällig gewählt, sollte die Wahrscheinlichkeit, daß mindestens einer der m Teildatensätze „gut“ ist, nahezu 1 sein. Ein Teildatensatz ist „gut“, wenn er p unverschmutzte Datenpunkte enthält, die bis zu einem Anteil von ε verschmutzt sein können. Für große n lautet der entsprechende Wahrscheinlichkeits-Ausdruck wie folgt:

$$1 - (1 - (1 - \varepsilon)^p)^m.$$

Der Algorithmus wird nun anhand eines Beispiels illustriert.

Beispiel (LMS Algorithmus):

Damit das Beispiel überschaubar bleibt, betrachten wir den zweidimensionalen Fall und wählen n = 9 und p = 2. Für das Beispiel konzentrieren wir uns auf drei Teildatensätze, nämlich $J_1 = \{f, g\}$, $J_2 = \{f, h\}$ und $J_3 = \{g, h\}$ (Abb. 18a, /Rousseeuw, Leroy 1987/).

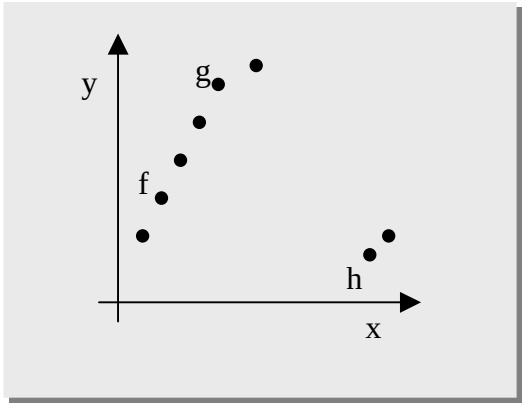


Abb. 18a: Datensatz für das Beispiel

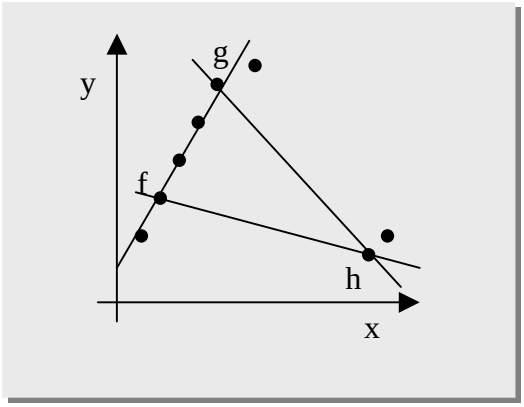


Abb. 18b: LMS Algorithmus

Wir beginnen mit den Datenpunkten f und g. Die Gleichung für die Regressionsgerade, die durch f und g verläuft (Abb. 18b, /Rousseeuw, Leroy 1987/), errechnet sich durch die Lösung des folgenden Gleichungssystems:

$y_f = \beta_0^0 + \beta_1^0 x_f$
$y_g = \beta_0^0 + \beta_1^0 x_g,$

wobei (x_f, y_f) und (x_g, y_g) die Koordinaten der Punkte f bzw. g sind.

Ist das Gleichungssystem gelöst, so erhält man den Koeffezientenvektor $\beta_0 = (\beta_0^0, \beta_1^0)^T$. Im nächsten Schritt werden die Residuen $(y_i - \beta_0^0 - \beta_1^0 x_i)$ berechnet, d.h. der Abstand aller Datenpunkte von der Regressionsgeraden (f, g) wird berechnet. Als nächstes wird der Median der quadrierten Residuale gebildet. Analog wird auch bei den Paaren (f, h) und (g, h) vorgegangen. Die Regressionsgerade mit minimalem Median-Wert ist die gesuchte Regressionsgerade.

Der Abbildung 18b können wir entnehmen, daß die Regressionsgerade (f, g) besser ist als die Regressionsgeraden (f, h) und (g, h). Auch der LMS Algorithmus wird die Regressionsgerade (f, g) als Ergebnis liefern, denn die Residuale und somit der Median für (f, g) haben die kleinsten Werte.

Der Algorithmus kann durch eine Parallelausführung verbessert werden, da er sich hierfür sehr gut eignet. Jeder Prozessor kann nämlich einen Koeffizientenvektor β_J und den entsprechenden Median berechnen.

Neben LMS wurde von P. J. Rousseeuw der Regressionsschätzer LTS (Least Trimmed Squares) vorgestellt, das genauso leistungsfähig ist wie LMS /Rousseeuw, Leroy 1987/. Diese Verfahren werden von Tools, wie z.B. „S-Plus“, „Mathematica“ und „SAS/IML“, unterstützt /Rousseeuw, Aelst 1999/.

4.3.2.3 Regressions- und Korrelationsanalyse

Bei der robusten Regression lassen sich auch Begriffe, wie z.B. Reststreuung, einführen. Die Reststreuung wird dann auch zur Identifizierung der Ausreißer verwendet. Die Definition lautet wie folgt:

Reststreuung s^*

$$(1.2) \qquad s^* = \sqrt{\frac{\sum_{i=1}^{n} w_i \varepsilon_i^2}{\sum_{i=1}^{n} w_i - p}}$$

Die Anzahl der betrachteten Einflußgrößen wird durch *p* bezeichnet, wobei n > p+1 gilt. n ist die Anzahl der betrachteten Datensätze und ε_i ist der i-te Residual (bzw. Störgröße).

Die Gewichtung w_i ist wie folgt definiert.

$$w_i = \begin{cases} 1 & falls \quad \left|\varepsilon_i / s^0\right| \le 2.5 \\ 0 & sonst. \end{cases} \quad mit \quad s^0 = 1.4826\left(1 + \frac{5}{n-p}\right)\sqrt{\underset{i=1,\ldots,n}{med}\ \varepsilon_i^2}.$$

Das Maß s^0 kann als eine einfache Variante der Reststreuung gewählt werden. Wir weisen darauf hin, daß sowohl s^0 als auch s^* wieder robust sind, d.h. gegenüber Ausreißern resistent sind.

Um Ausreißer aufzuspüren, werden die Residuale durch ε_i / s^* normiert. Wenn $| \varepsilon_i / s^* | \ge 2.5$ ist, handelt es sich bei dem i-ten Datenpunkt um einen Ausreißer. Sind die Ausreißer identifiziert, können sie näher untersucht und gegebenenfalls korrigiert werden.

Damit eine Aussage über die Stärke der linearen Beziehung zwischen der Zielgröße und den Einflußgrößen gemacht werden kann, definieren wir wieder das Bestimmtheitsmaß.

Bestimmtheitsmaß B

Sei n die Anzahl der Datensätze.

Falls eine Regressionskonstante vorliegt, dann lautet die Definition wie folgt

$$(1.3) \qquad B = 1 - \left(\frac{med\left|\varepsilon_i\right|}{mad\,(y_i)}\right)^2 \quad mit \quad mad\,(y_i) = \underset{i=1,\ldots,n}{med}\left\{\left|y_i - \underset{j=1,\ldots,n}{med}\ y_j\right|\right\}$$

Bei einem Regressionsmodell ohne eine Konstante, lautet die Definition wie folgt

$$(1.4) \qquad B = 1 - \left(\frac{med\left|\varepsilon_i\right|}{med\left|y_i\right|}\right)^2$$

Das Bestimmtheitsmaß ist ebenfalls robust und hat einen Wert zwischen 0 und 1 ($0 \leq B \leq 1$). Wünschenswert ist ein Wert dicht bei 1, einen Schwellenwert gibt es aber nicht. Beträgt der Wert von B beispielsweise 0.995, werden 95.5% der Daten durch das lineare Modell erklärt.

4.4 CEOS-Komponentenstruktur

Für die Implementierung des CEOS-Verfahrens wurde eine Vier-Schichten-Architektur entworfen, die in der nächsten Abbildung (Abb. 19) dargestellt ist.

Damit die erforderlichen Eingabedaten weitgehend automatisch erfaßt werden können, wurde als Implementierungs-Schnittstelle das UML-Werkzeug „objectiF 4.5“ gewählt (*Schicht II*). „objectiF“ legt die Modelldaten, wie z.B. Klassendiagramme, in einem eigenen objektorientierten Datenbanksystem ab (*Schicht I*). Daher werden Mechanismen, wie z.B. Transaktionsverwaltung, Mehrbenutzer-Synchronisation oder Sicherheitsaspekte zur Verfügung gestellt, die wir für die Implementierung des CEOS-Verfahrens nutzen können.

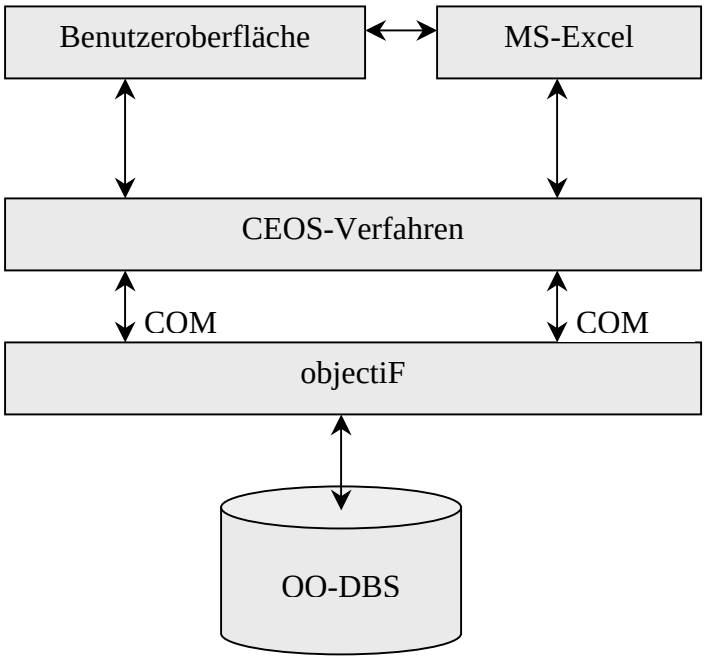


Abb. 19: Architektur des CEOS-Moduls

Der Zugriff auf die Modelldaten (z.B. Attribute oder Methoden einer Klasse) mit der *COM-Technologie* (Component Object Model) erfolgt in der dritten Schicht (*CEOS-Verfahren*). Hier findet auch die Berechnung der Kennzahlen anhand der Metriken statt. Die Bedienung des Systems und die Präsentation der Schätz-Ergebnisse werden durch die oberste Schicht (*Benutzeroberfläche* und *MS-Excel*) unterstützt. Das Tabellenkalkulations-Programm *MS-Excel* ist direkt in die Benutzerschnittstelle eingebunden. *MS-Excel* dient zur Erfassung von Projektdaten und zur Präsentation von Schätzergebnissen.

Jede der Schichten besteht aus einer Reihe von Komponenten. Die nächste Abbildung (Abb. 20) präsentiert die Schichten *CEOS-Verfahren* und *Benutzeroberfläche*. Die Schicht *Benutzeroberfläche* enthält eine Reihe von Komponenten, die Dialoge und Menüs für den Anwender zur Verfügung stellen. Die Kommunikation mit der Server-seitigen Schicht *CEOS-Verfahren* verläuft über die beiden Komponenten „Model“ und „Facade“, die jeweils die Schichten abkapseln.

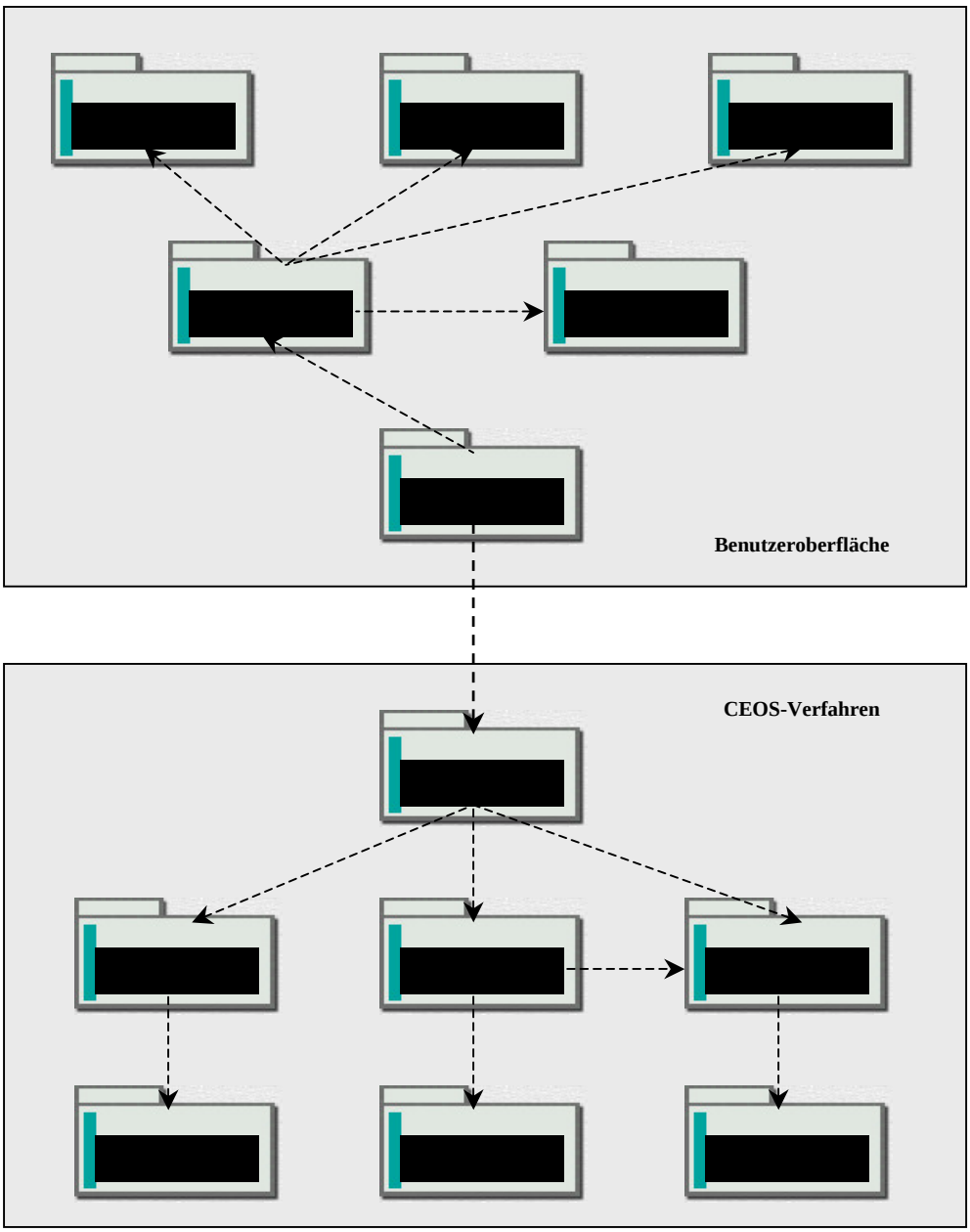


Abb. 20: Komponenten-Struktur der Schichten „CEOS-Verfahren“ und „Benutzeroberfläche“ (Pfeil-Semantik: X_1 benutzt X_2)

Neben der Komponente „Facade“ besteht die Schicht *CEOS-Verfahren* aus weiteren Komponenten, wie z.B. „Metrics“. In dieser Komponente sind die in Abschnitt 4.3.1 definierten Maße implementiert. Die Eingabedaten für die Berechnungen anhand der Maße werden mit Hilfe der Komponente „COMBridge“ aus „objectiF“ ausgelesen. Wie in Abschnitt 4.3.2 beschrieben, wird für die Berechnung der Koeffizienten die statistische Technik der robusten Regression verwendet. Die Komponente „Regression“ stellt hierzu entsprechende Klassen zur Verfügung. Da diese Komponente in *Java* implementiert ist, die objectiF-Komponenten dagegen in *Visual Basic*, wird eine entsprechende Brücke benötigt, die durch die Komponente „JavaBridge“ bereitgestellt wird. Zur Präsentation der Schätz-Ergebnisse in Excel- bzw. XML-Format wird die Komponente „Publisher“ benötigt. Diese Komponente nutzt die Dienste der Komponente „XMLSupport“, die eine Reihe von XML-Routinen zur Verfügung stellt. Solche XML-Routinen werden benötigt, da die gesamte Kommunikation zwischen Server (*Schicht I-III*) und Client (*Schicht IV*) via XML stattfindet.

Im folgenden werden wir exemplarisch die interne Struktur zweier Kernkomponenten („Metrics“ und „Regression“) vorstellen. Dazu werden in den nächsten zwei Abbildungen (Abb. 21, 22) die entsprechenden Klassendiagramme präsentiert.

Dem Klassendiagramm für die Komponente „Metrics“ (Abb. 21) kann entnommen werden, daß eine zweistufige Vererbungshierarchie vorliegt. Alle Klassen erben von der Superklasse „BAUSTEIN“, die das Interface „Serializable“ implementiert. Somit verfügen alle Unterklassen über die *Objekt-Serialisierung*. Während mit den abstrakten Klassen die Gemeinsamkeiten zusammengefaßt werden, stellen die konkreten Klassen Methoden zur Verfügung, mit denen für einen Baustein zum gegebenen Schätz-Zeitpunkt Berechnungen vorgenommen werden können. Zum Beispiel stellt die Klasse „System_A“ Methoden für die Berechnung von Maßen zur Verfügung, die auf ein System während der Analysephase angewendet werden können. Die in der Abbildung 21 dargestellte generalisierte Implementierung wurde für die Anwendung auf „objectiF“ optimiert.

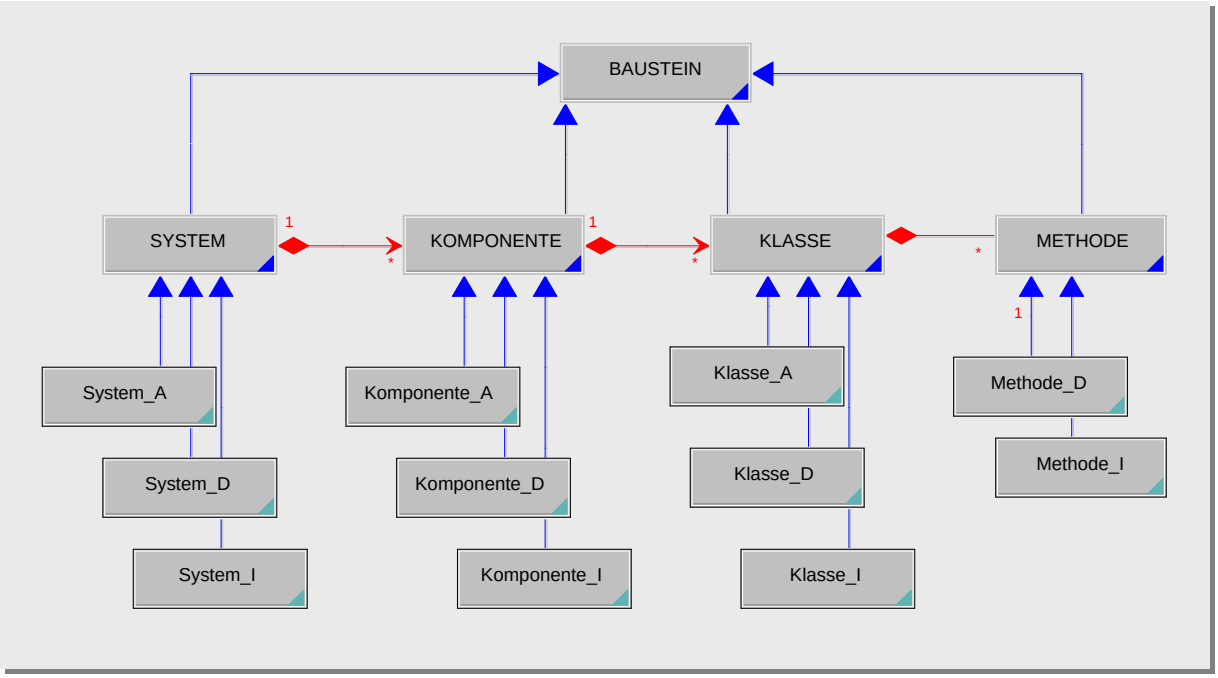


Abb. 21: Klassendiagramm für die Komponente „Metrics“

Da für das CEOS-Verfahren prinzipiell beliebige Regressions-Techniken eingesetzt werden können, wurde das Interface „Regression“ definiert (Abb. 22). Da wir die Technik LMS (Least Median of Squares) einsetzen wollen, wird dieses Interface von der Klasse „LMS“ spezialisiert, die die entsprechende Technik implementiert.

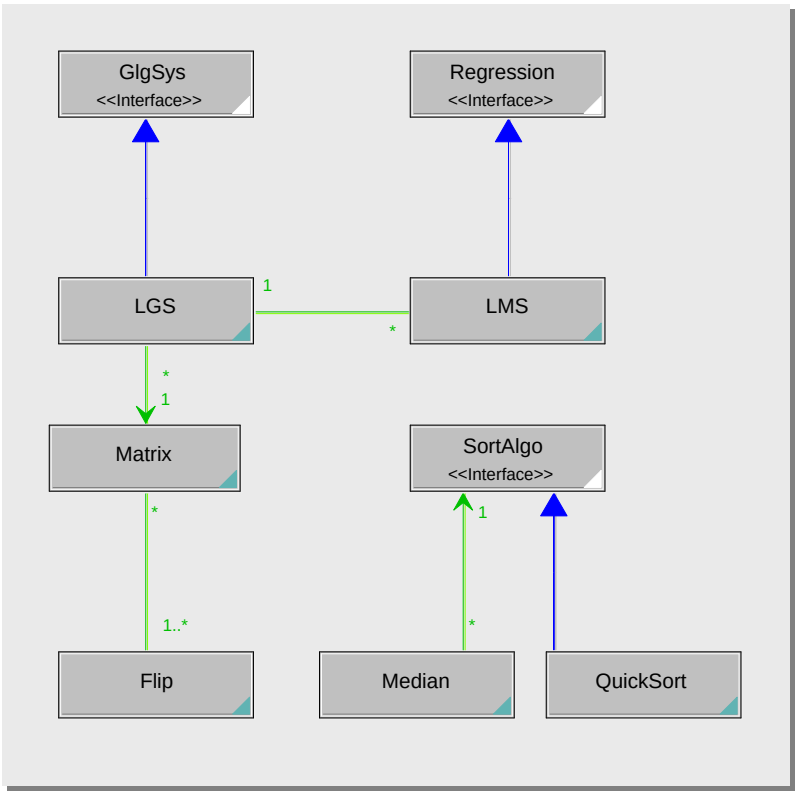


Abb. 22: Klassendiagramm für die Komponente „Regression“

Das LMS-Verfahrens setzt das Lösen von linearen bzw. nicht-linearen Gleichungssystemen voraus. Daher wurde das Interface „GlgSys“ (*Gleichungssystem*) definiert. In unserem Fall müssen lineare Gleichungssysteme gelöst werden. Daher wurde das Interface durch die Klasse „LGS“ (*Lineare Gleichungssysteme*) spezialisiert. Für das Lösen von Gleichungssystemen werden Matrizen und entsprechende Methoden, wie z.B. der Gauss-Algorithmus, benötigt. Diese Methoden werden von der Klasse „Matrix“ zur Verfügung gestellt. Die Klasse „Flip“ ist eine Hilfsklasse, die z.B. bei der Berechnung der Inversen einer Matrix benötigt wird. Da für die Berechnung des Medians (Klasse „Median“) sortiert werden muß, wurde das Interface „SortAlgo“ definiert. Wir haben den Sortieralgorithmus „Quick-Sort“ gewählt, so daß das Interface mit der Klasse „QuickSort“ implementiert wird.

4.5 CEOS-Implementierung

Wir stellen nun die Implementierung des oben beschriebenen CEOS-Verfahren anhand der Benutzerschnittstellen vor.

Nach der Installation des CEOS-Verfahrens kann es durch die Auswahl des Menüpunkts „Add-Ins/Aufwandsschätzung“ gestartet werden. Es erscheint das folgende Hauptfenster:

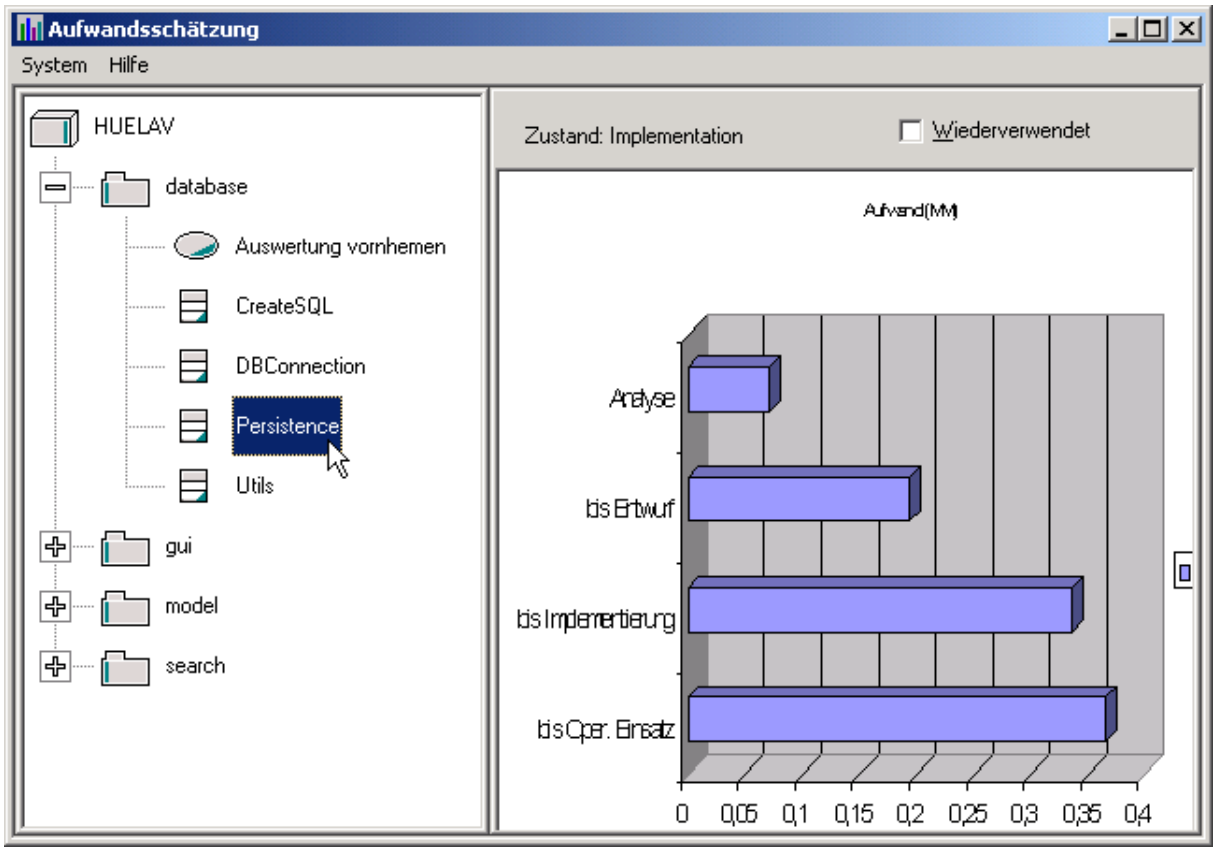


Abb. 23: Hauptfenster des CEOS-Verfahrens

Das Hauptfenster besteht aus den Menüpunkten „System“ und „Hilfe“, die allgemeine Dienste und Hilfsfunktionen zur Verfügung stellen. Im linken Teilfenster befindet sich der „Systembaum“ (Aktualisierung durch die Taste F5). In dem obigen Beispiel heißt das System „Huelav“ (*Haushaltsüberwachung*), das aus den Komponenten „database“, „gui“, „model“ und „search“ besteht. Die Komponenten enthalten wiederum Klassen bzw. Anwendungsfälle. Damit die Anwendungsfälle im Systembaum erscheinen, müssen die ihnen zugeordneten Klassen bzw. Komponenten in „objectiF“ definiert werden. Das ist in „objectiF“ möglich, da Anwendungsfälle technisch durch Klassen bzw. Komponenten realisiert werden. So läßt sich auch die Aufwandsschätzung von Anwendungsfällen auf die von Klassen bzw. Komponenten zurückführen.

Durch einen Doppelklick auf einen beliebigen Baustein bzw. Anwendungsfall wird der Aufwand berechnet und im rechten Teilfenster durch ein Balkendiagramm dargestellt. Damit detaillierte Planungen erfolgen können (z.B. für Revisionspunkte), wird der Aufwand bis zur Analyse, bis einschließlich Entwurf, bis einschließlich Implementierung und bis einschließlich operationeller Einsatz geschätzt und durch einen Balken präsentiert. Oberhalb des rechten Teilfensters wird der aktuelle Zustand eines Bausteins (z.B. Entwurf oder Implementierung) angegeben. Außerdem kann für eine Klasse oder Komponente festgelegt werden, ob es sich um einen wiederverwendeten Baustein handelt. Ist das der Fall, so wird der Aufwand neu geschätzt und die Maße für die Wiederverwendung (CU_m, XU_m) berücksichtigt. Durch das Betätigen des Kontextmenüs im rechten Teilfenster können beliebige Excel-Funktionen, wie z.B. Drucken, Speichern oder eine andere Darstellung der Ergebnisse, ausgeführt werden. Abbildung 24 zeigt hierzu ein Beispiel.

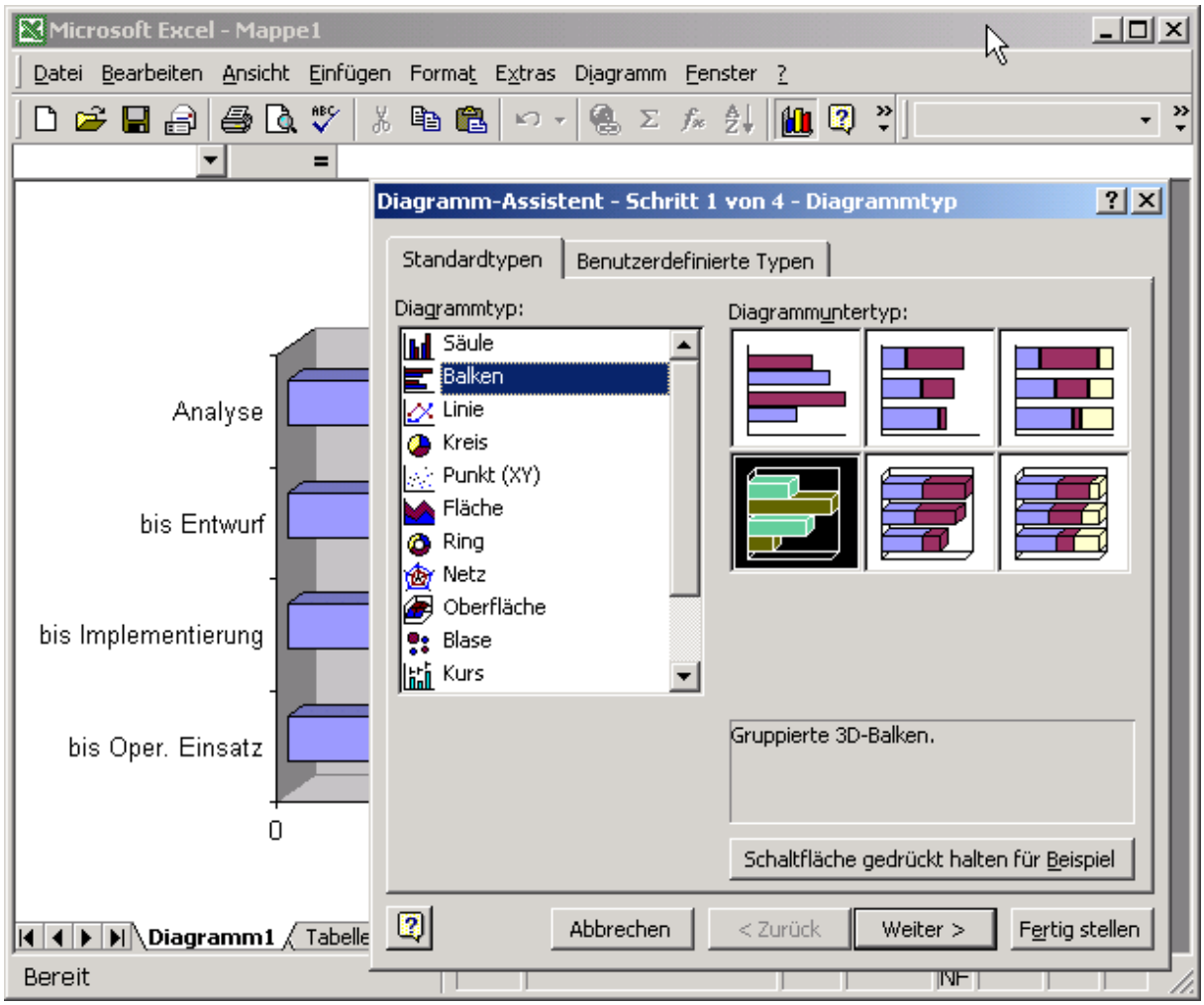


Abb. 24: Excel-Funktionen zur Darstellung des Schätzergebnises

Die Zustände, die ein Baustein durchlaufen kann, werden automatisch erkannt und angezeigt. Für den Fall, daß ein Baustein im Zustand „Analyse“ ist, müssen entsprechende Eingaben erfolgen, bevor eine Aufwandsschätzung vorgenommen werden kann. Das bedeutet, daß durch einen Doppelklick auf einen Baustein im Analyse-Zustand zunächst ein Eingabe-Dialog erscheint. Je nach Baustein (System, Komponente oder Klasse) werden folgende drei Eingabe-Dialoge unterschieden:

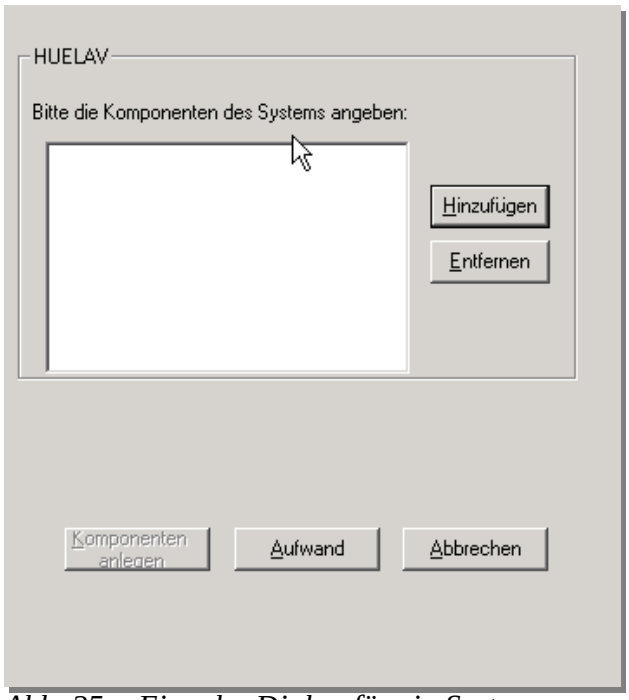


Abb. 25a: Eingabe-Dialog für ein System

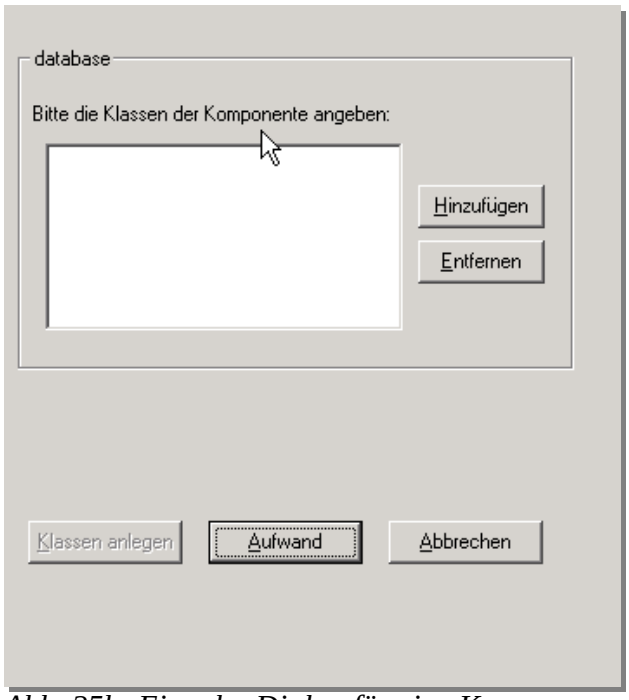


Abb. 25b: Eingabe-Dialog für eine Komp.

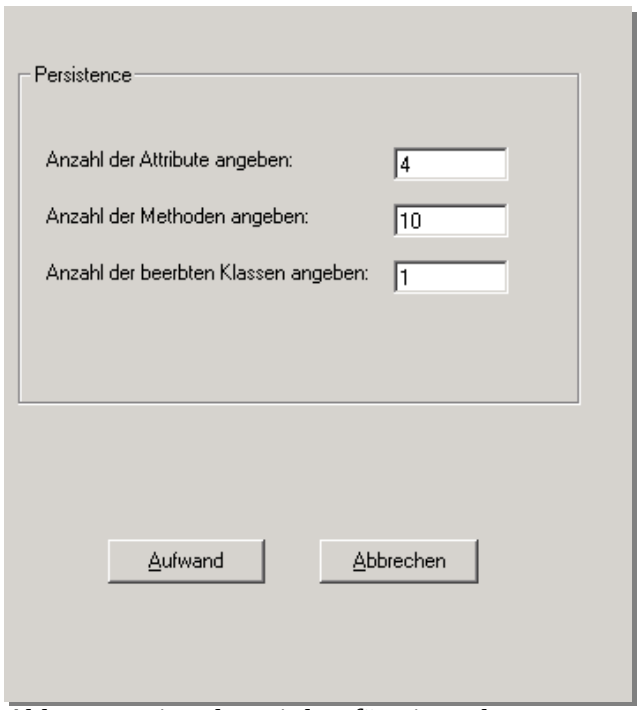
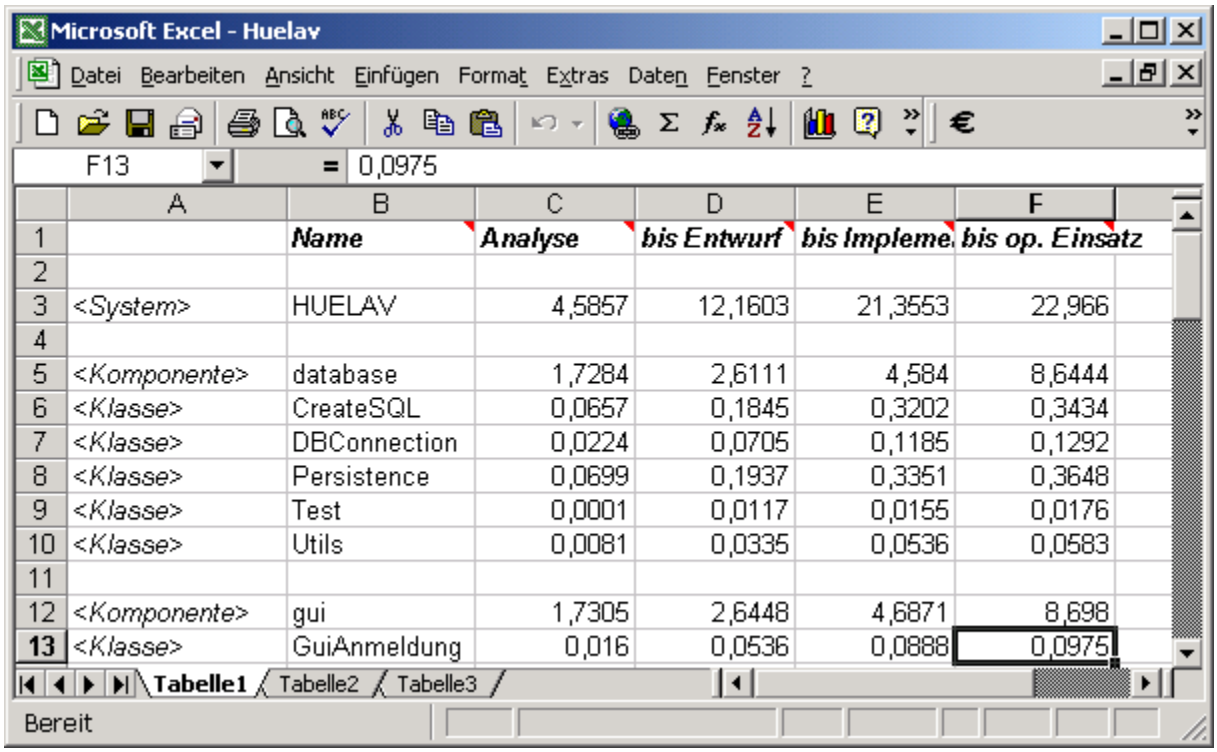


Abb. 25c: Eingabe-Dialog für eine Klasse

Bei der Aufwandsschätzung nach solchen Eingaben werden Berechnungen anhand der Maße CCA_m^{Def}, XCA_r, XCA_m, SCA_r und SCA_m durchgeführt.

In der Regel ist ein Überblick über die Aufwände aller Bausteine interessant, und zwar in einer weiter verwertbaren Form. Dazu wurde ein „Excel-Publisher“ implementiert, der mit der Wahl des Menüpunkts „System/Publisher“ gestartet werden kann.

Die nächste Abbildung (Abb. 26) zeigt hierzu ein Beispiel.



	A	B	C	D	E	F
1		**Name**	**Analyse**	**bis Entwurf**	**bis Impleme**	**bis op. Einsatz**
2						
3	<System>	HUELAV	4,5857	12,1603	21,3553	22,966
4						
5	<Komponente>	database	1,7284	2,6111	4,584	8,6444
6	<Klasse>	CreateSQL	0,0657	0,1845	0,3202	0,3434
7	<Klasse>	DBConnection	0,0224	0,0705	0,1185	0,1292
8	<Klasse>	Persistence	0,0699	0,1937	0,3351	0,3648
9	<Klasse>	Test	0,0001	0,0117	0,0155	0,0176
10	<Klasse>	Utils	0,0081	0,0335	0,0536	0,0583
11						
12	<Komponente>	gui	1,7305	2,6448	4,6871	8,698
13	<Klasse>	GuiAnmeldung	0,016	0,0536	0,0888	0,0975

Abb. 26: Ergebnis des Publishers für das Projekt „Huelav“

In der obigen Excel-Tabelle wird der geschätzte Aufwand für die einzelnen Bausteine in Bearbeiter-Monaten dargestellt. Eine Rundung der Werte kann durch MS-Excel vorgenommen werden.

Da das Excel-Format sehr kryptisch und von Fremdsystemen schwer verarbeitbar ist, wurde auch eine „Export-Funktion“ programmiert.

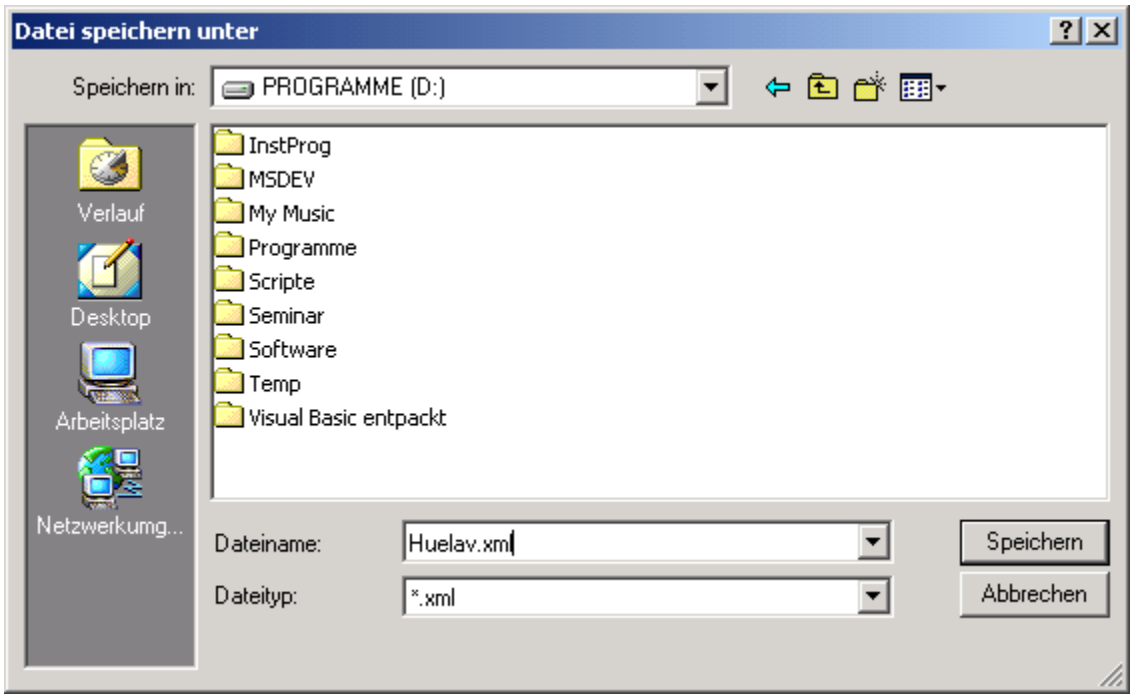


Abb. 27: XML-Export

Die Funktion exportiert die Schätzergebnisse in ein XML-Format, so daß die Ergebnisse von Fremdsystemen leicht verarbeitet werden können. Die „Export-Funktion“ kann durch das

Betätigen des Menüpunkts „System/Export“ ausgeführt werden. Die Abbildung 27 zeigt hierzu ein Beispiel.

Wenn ein Projekt abgeschlossen ist, sollte der tatsächlich angefallene Aufwand zwecks einer Regressionsanalyse erfaßt werden. Damit eine kontinuierliche Anpassung des Verfahrens an organisatorische und technische Veränderungen möglich ist, sollte eine regelmäßige Regressionsanalyse durchgeführt werden. Zur Erfassung von Projektdaten und zur Unterstützung der Regressionsanalyse wurden entsprechende Funktionen implementiert, die unter dem Menüpunkt „System/Regression“ gefunden werden können. Die nächste Abbildung zeigt hierzu ein Beispiel (Abb. 28).

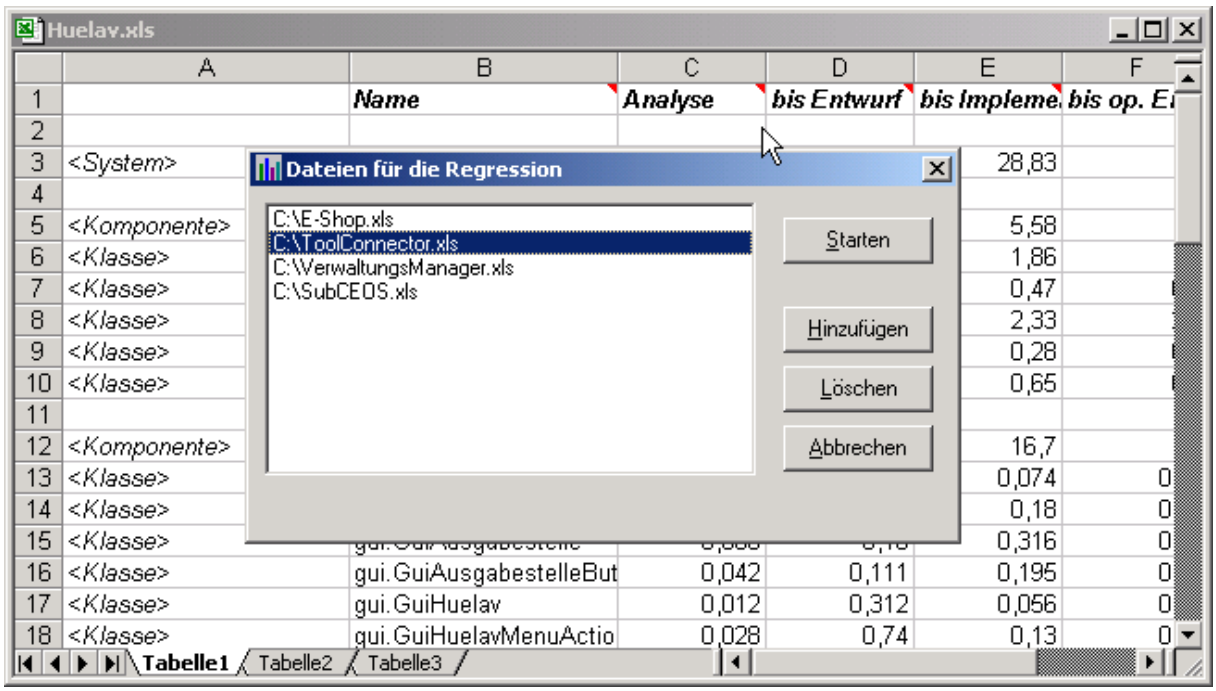
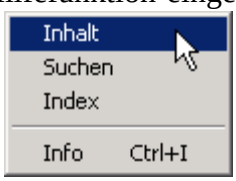


Abb. 28: Datenerfassung und Regressionsanalyse

In der obigen Abbildung wurden nach dem Abschluß des Projekts „Huelav“ die angefallenen Aufwände für die einzelnen Bausteine erfaßt. Danach wurden mehrere Projekte für die Regressionsanalyse ausgewählt. Mit dem Button „Start“ kann die Regression angestoßen werden. Hierbei wird aus den Projektdaten eine XML-Datei generiert, die einer externen Java-Klasse übergeben wird. Danach wird die „Java-Virtual-Machine“ gestartet, so daß die Regressionsanalyse parallel zur CEOS-Anwendung durchgeführt werden kann.

Damit die Anwender auch nachvollziehen können, wie das CEOS-Verfahren funktioniert und bedient werden kann, wurde eine Hilfefunktion eingerichtet. Das Hilfesystem kann über das Menü „Hilfe“ angestoßen werden.



Die nächste Abbildung zeigt einen Ausschnitt aus dem Hilfssystem (Abb. 29).

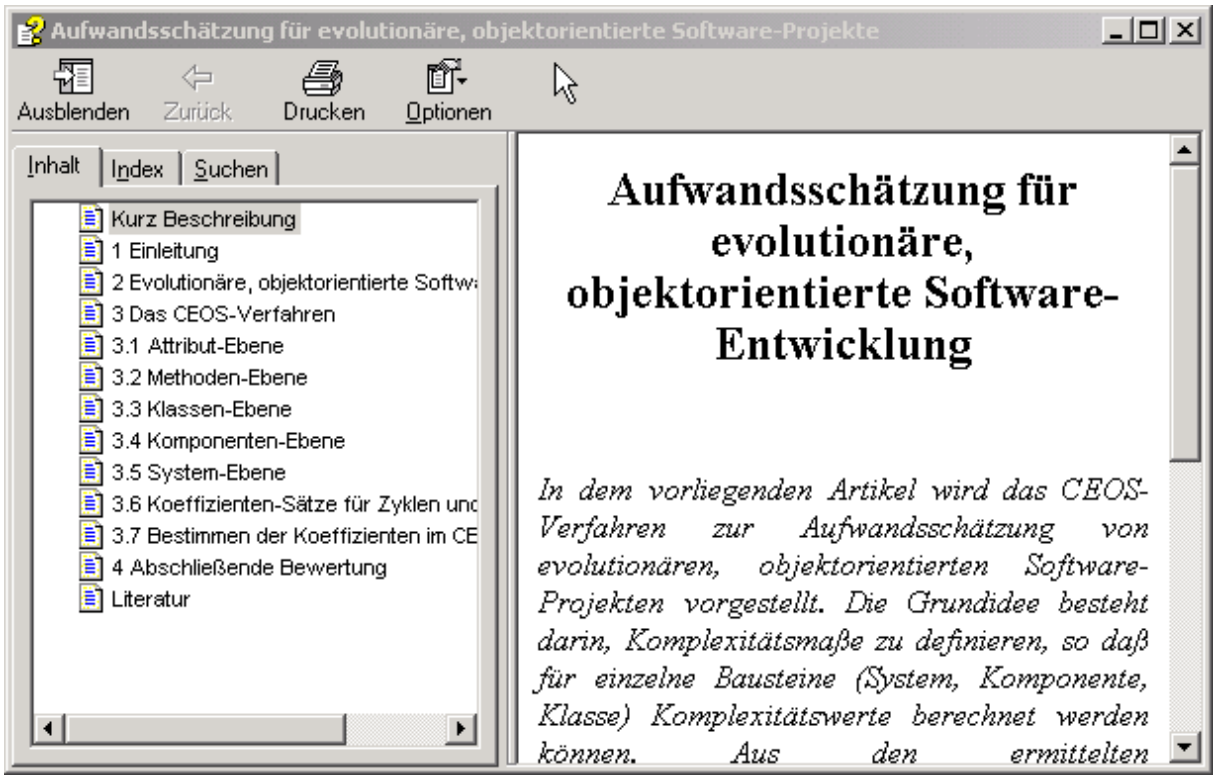


Abb. 29: Hilfesystem für das CEOS-Verfahren

4.5.1 Kurz-Studie

Zu Test- und Evaluierungs-Zwecken haben wir die Regressionsanalyse für zwei Kleinprojekte, mit jeweils ca. 100 Klassen, durchgeführt. Abschließend wurde ein bereits abgeschlossenes Kleinprojekt mit dem CEOS-Verfahren nachgeschätzt. In den nächsten Abbildungen wird das Verhältnis zwischen dem geschätzten und dem tatsächlichen Aufwand graphisch veranschaulicht.

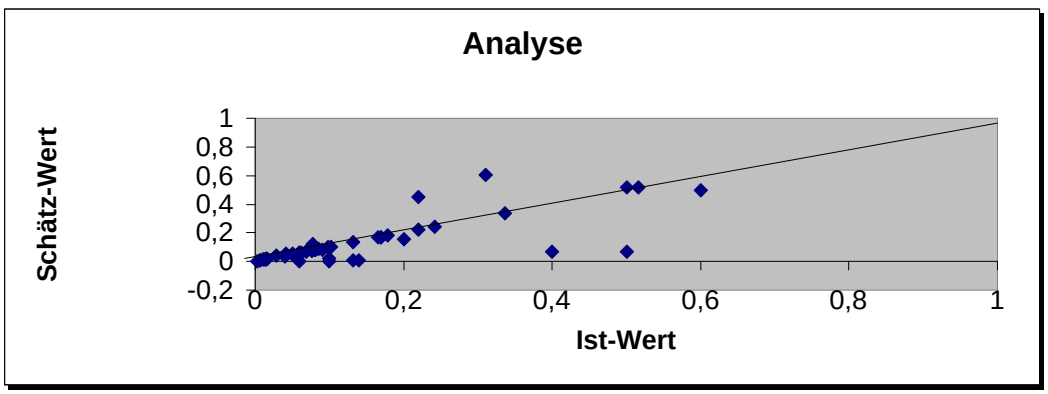


Abb. 30a: Aufwand bis einschließlich Analyse

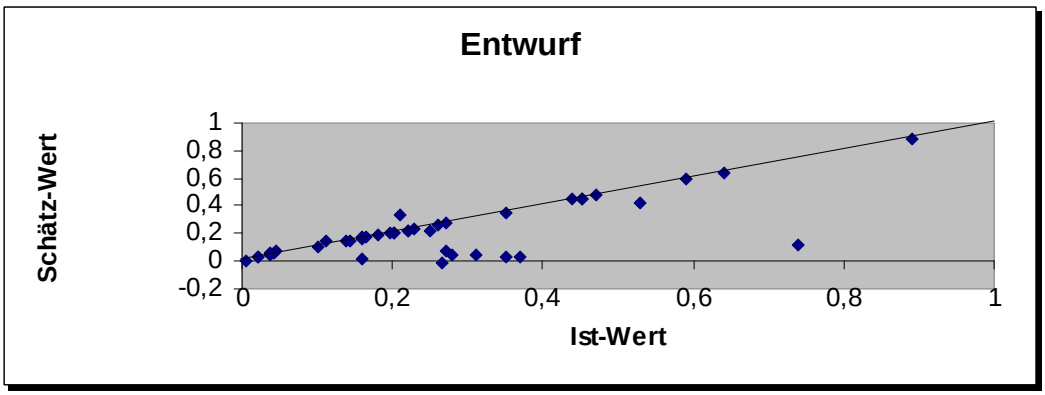


Abb. 30b: Aufwand bis einschließlich Entwurf

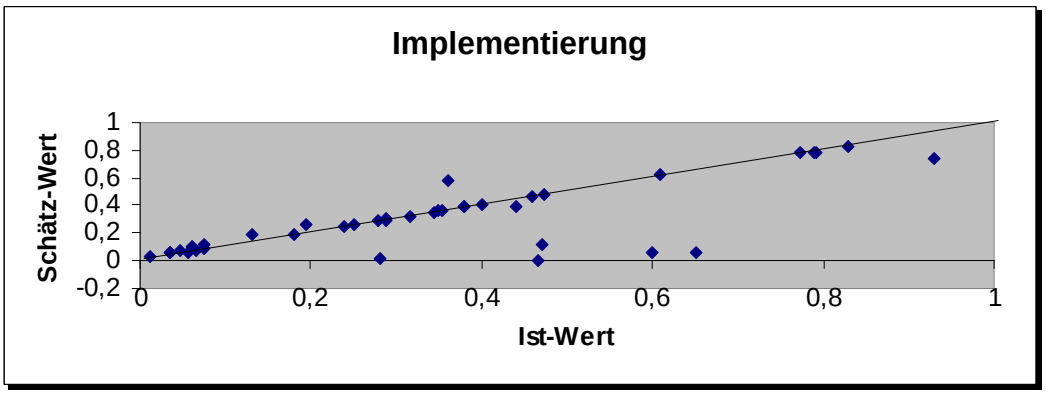


Abb. 30c: Aufwand bis einschließlich Implementierung

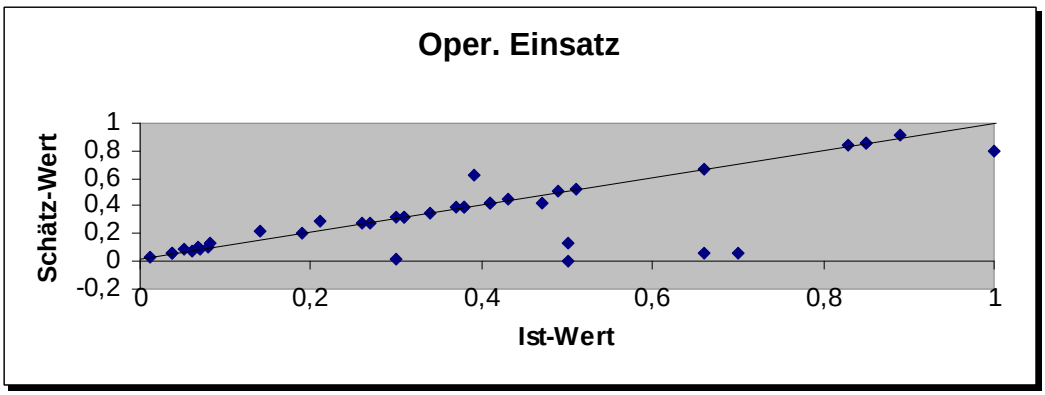


Abb. 30d: Aufwand bis einschließlich operationeller Einsatz

Die obigen Abbildungen zeigen, daß viele der Punkte die Idealgerade tangieren. Das bedeutet, daß die Differenz zwischen dem tatsächlichen (Ist-Wert) und dem geschätzten Aufwand (Schätz-Wert) für einen Baustein sehr oft minimal ist. Das zeigt, daß wir mit dem CEOS-Verfahren auf dem richtigen Weg zur Aufwandsschätzung von evolutionären, objektorientierten Projekten sind. Natürlich müssen noch mehr Projekte untersucht werden, um gegebenenfalls Verbesserungen vorzunehmen. Insbesondere muß ein Unternehmen, das das Verfahren einsetzt, eigene Projekte untersuchen, damit das Verfahren auf die unternehmensspezifischen Einflussfaktoren, wie z.B. Qualifikation der Mitarbeiter, abgestimmt werden kann.

Das Anfertigen dieser kurzen Studie hat uns gezeigt, daß man für die Regressionsanalyse EOS-Projekte einbeziehen sollte. Da zur Zeit keine EOS-Projekte vorliegen, haben wir die EOS-Charakteristiken nachgespielt. Das führt natürlich zu ungenauen Meßwerten. Was

ebenfalls zu Ungenauigkeiten führt, ist das Erfassen von Projektdaten nach ihrem Abschluß. Die Projektbeteiligten können im nachhinein den genauen Aufwand für die einzelnen Bausteine oft nicht mehr angeben, so daß sich Meßfehler einschleichen. Daher sollten die Daten projektbegleitend erfaßt und verarbeitet werden.

4.5.2 Abschließende Bewertung

Für das CEOS-Verfahren müssen zwar komplexe und aufwendige Berechnungen ausgeführt werden, dafür ergeben sich aber folgende Vorteile:

- Das CEOS-Verfahren berücksichtigt Entwicklungszyklen und Aspekte der Revision und Nutzung. Daher ist der evolutionäre Charakter der Software-Entwicklung im Verfahren verankert.
- Für jeden der Entwicklungsbausteine System, Komponente und Klassen sind Aufwandsschätzungen möglich. Damit sind detaillierte Termin-, Budget- und Personalplanungen möglich.
- Der Aufwand von Anwendungsfällen kann geschätzt werden, da die Aufwandsschätzung von Anwendungsfällen auf die von Bausteinen zurückgeführt wurde (siehe 4.5).
- Für die einzelnen Bausteine können sowohl der Gesamtaufwand, als auch der Aufwand für die einzelnen vier Phasen geschätzt werden. Dies erhöht die Flexibilität des Managements.
- Neben einzelnen Entwicklungszyklen können für jeden der Bausteine mehrere Entwicklungszyklen betrachtet werden. Damit wird die Projektdynamik und der Entwicklungsalltag widergespiegelt und unterstützt.
- Die Prinzipien der Wiederverwendung und der Bausteinbibliothek werden vom CEOS-Verfahren unterstützt.
- Objektorientierte Konzepte, wie die Vererbung, Assoziation, Aggregation usw. werden bei der Schätzung mit betrachtet.
- Durch die ständige Anpassung der Koeffizienten ist das Verfahren auf Unternehmen zugeschnitten und paßt sich den unternehmensspezifischen Faktoren, wie z.B. Personalqualifikation oder technischer Fortschritt, an.
- Das CEOS-Verfahren kombiniert die Basismethoden *Analogiemethode*, *Gewichtungsmethode* und die *Methode der parametrischen Gleichungen*. Damit können schon sehr früh erste grobe Schätzungen vorgenommen werden (Analogiemethode), die nach und nach verfeinert werden (Gewichtungsmethode und Methode der parametrischen Gleichungen).
- Die definierten Software-Maße sind allgemein (beliebiges Grundmaß m) gehalten und sind theoretisch fundiert.
- Nach einer ersten groben Schätzung ist das Verfahren automatisierbar, da Klassendiagramme und Quellcode für die Aufwandsschätzung herangezogen werden.

Im ersten Teil dieses Kapitels wurde ein Prototyp für ein Prozeßmanagement-System (PMS) vorgestellt, der das EOS-Modell durchgängig unterstützt. Der in *Java* geschriebene Prototyp besteht aus „Systemstruktur“, „Prozeßstruktur“, „Menüführung“ und „Prozeßauswahl“. Die „Systemstruktur“ bildet die Baustein-Struktur eines EOS-Projekts ab. Die „Prozeßstruktur“ beschreibt die einzelnen Teilprozesse von EOS mittels Aktivitätsdiagramme. Die „Menüführung“ erlaubt die Auswahl von allgemeinen Diensten, während die „Prozeßauswahl“ das Wechseln zwischen Subprozessen ermöglicht.

Für eine erfolgreiche Projektplanung ist die Kenntnis des Entwicklungsaufwands entscheidend. Zu diesem Zweck wurde das CEOS-Verfahren (Cost estimation for EOS projects), ein Schätzverfahren für den Entwicklungsaufwand von evolutionären, objektorientierten Projekten, vorgestellt. Hierbei werden für die einzelnen Bausteine im EOS-Modell Komplexitätsmaße definiert, in die je nach Projektstand die entsprechend vorliegenden Daten einfließen. Die definierten Software-Maße enthalten eine Reihe von Koeffizienten, die das Verfahren flexibel gestalten. Neben Gesamtschätzungen für die einzelnen Bausteine können auch Schätzungen für die vier Tätigkeiten vorgenommen werden. Außerdem kann ein oder können mehrere Entwicklungszyklen in Betracht gezogen werden. Somit ist der evolutionäre Charakter der Software-Entwicklung in dem Verfahren verankert, so daß mit dem CEOS-Verfahren ein dynamisches Projektmanagement unterstützt wird. Damit das CEOS-Verfahren praktisch angewendet werden kann, wurde eine vollständige Implementierung angegeben und in PMS integriert.

5 Architektur-Konzept

Ausgehend vom Prototypen des letzten Kapitels wird nun eine Architektur entwickelt, die eine Implementierung des Prototypen ermöglicht. Ein detaillierter Entwurf ist nicht das Ziel des Kapitels.

☑ Das Kapitel 4 wird als bekannt vorausgesetzt. Kenntnisse in Java sind hilfreich.

5 Architektur-Konzept

Im vorherigen Abschnitt haben wir einen Prototypen für das EOS-Modell vorgestellt. Mit dem Prototypen wurde gezeigt, wie ein reales Anwendungssystem aussehen könnte, das das EOS-Modell durchgängig unterstützt. In diesem Abschnitt wollen wir eine Architektur für ein solches reales Anwendungssystem vorschlagen, das heutigen Anforderungen, wie z.B. Transaktionsfähigkeit, Sicherheit, Persistenzbehandlung, Multithreadfähigkeit, Skalierbarkeit oder Verfügbarkeit, genügt. Ein detaillierter Entwurf ist aber nicht das Ziel des Kapitels. Bei der vorgeschlagenen Architektur halten wir uns an die *Java 2 Platform Enterprise Edition (J2EE)* Spezifikation von „Sun Microsystems“ und nutzen das dazugehörige Komponenten-Modell *Enterprise JavaBeans (EJB)*. Daher führen wir zunächst kurz in diese beiden Themen ein. Für eine Vertiefung verweisen wir auf die Hinweise im Literaturverzeichnis weiter unten.

5.1 J2EE und EJBs

Viele Anwendungssysteme werden nach einer zweischichtigen Client-Server-Architektur konzipiert. Folglich müssen viele Standard-Dienste, wie z.B. Sicherheitsmechanismen oder Transaktionsmanagement, hauptsächlich auf dem Client implementiert werden. Daraus resultiert das Problem, daß der Client sich als Flaschenhals des Anwendungssystems erweist. Die Clients benötigen in der Regel viele Ressourcen, ihre Wartung ist kostspielig und eine Verteilung der Anwendung auf mehrere Rechner (Erhöhnung der Performanz) ist schwer möglich. Die Lösung des Problems wurde schon früh erkannt und lautet: Mehrschichtige Architekturen. Hierbei wird das Anwendungssystem in logische Schichten aufgeteilt und auf physisch getrennte Hardware-Systeme abgebildet. Insbesondere wird eine Mittelschicht eingeführt, die die Anwendungslogik beinhaltet. Die Benutzeroberflächen werden auf den Clients implementiert. Damit ist die Realisierung von „dünnen“ Clients möglich und das oben genannte Flaschenhals-Problem gelöst.

Mit J2EE spezifiziert „Sun Microsystems“ ein Muster für mehrschichtige Architekturen, das vollständig auf Java basiert. Die nächste Abbildung zeigt die Schichten einer J2EE-Architektur.

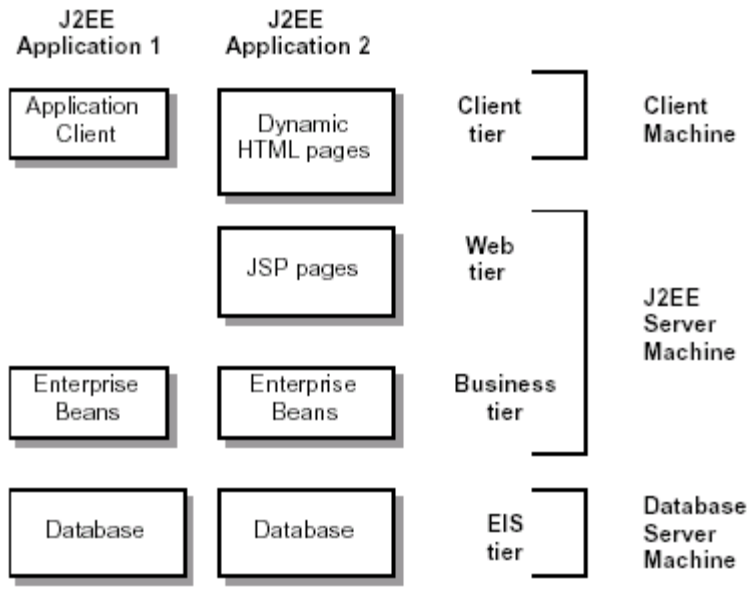


Abb. 1: J2EE-Architektur /J2EE 1.3/

In der Regel besteht eine J2EE-Architektur aus drei Schichten:

- Client-Schicht: Diese Schicht beinhaltet die Benutzerschnittstelle des Anwendungssystems als Application, Applet oder HTML-Seiten, die jeweils auf der Client-Maschine laufen.
- Business-Schicht: In der Business-Schicht ist die Anwendungslogik in Form von EJB-Komponenten implementiert. Wenn der Client auf die Anwendung via dynamische HTML-Seiten zugreift, so ist ein Web-Server und eine sogenannte JSP-Engine (JavaServer Page-Engine) für die dynamische Generierung von HTML-Seiten notwendig. Die Business-Schicht läuft auf einem J2EE-Server, der von entsprechenden Herstellern bezogen werden kann. Der J2EE-Server stellt den Anwendungsentwicklern Standard-Dienste, wie z.B. Transaktionsmanagement oder Sicherheitsmechanismen, zur Verfügung.
- EIS-Schicht (Enterprise Information System): Die EIS-Schicht besteht in der Regel aus einem Datenbanksystem zum Ablegen der Anwendungsdaten. Sie kann aber auch aus Legacy- oder anderen Fremd-Systemen bestehen.

Die J2EE-Komponenten werden in Behältern (*Container*) verwaltet und interagieren untereinander über ihre Schnittstellen. Die nächste Abbildung veranschaulicht diesen Sachverhalt.

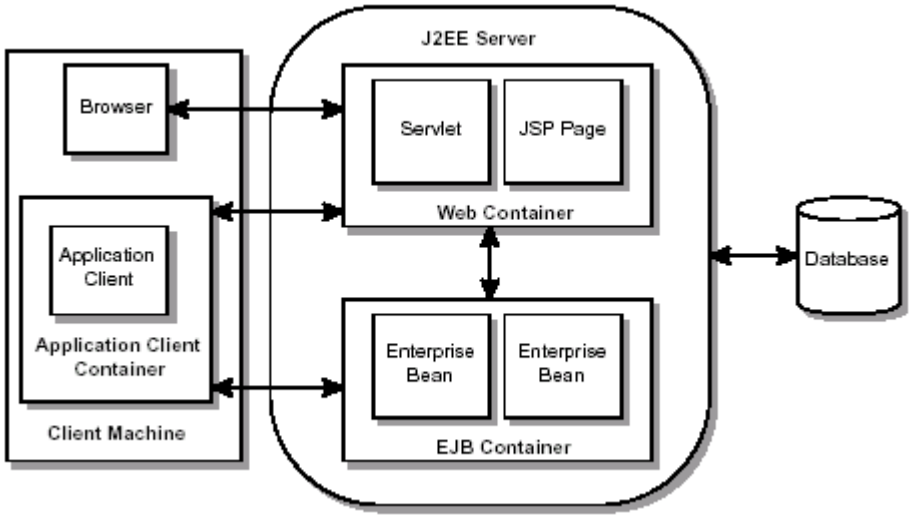


Abb. 2: J2EE-Server und Container /J2EE 1.3/

Für unsere Ausführung ist wichtig, daß der J2EE-Server einen EJB-Container beinhaltet. Ein EJB-Server ist in einem oder mehreren Prozessen organisiert. Er stellt eine Sammlung von Java-basierten Diensten bereit, z.B. einen via JNDI (*Java Naming and Directory Interface*) zugreifbaren Namensdienst, einen Transaktionsdienst, einen systemweiten Sicherheitsmechanismus oder Messaging-Dienste. Diese Dienste werden von dem EJB-Container genutzt und um weitere Dienste erweitert. Der EJB-Container verwaltet den Zugriff auf die EJBs, stellt den Transaktions-Kontext bereit, sorgt für die Bean-Persistenz, stellt EJBs Serverdienste zur Verfügung, erzeugt und zerstört EJBs. Softwareentwickler brauchen sich nur auf die Implementierung der Anwendungslogik in Form von EJBs zu konzentrieren. Im Kern ist ein EJB eine Java-Klasse mit gewissen Programmier- und Namenskonventionen. Die nächste Abbildung zeigt die Architektur eines EJBs (Abb. 3).

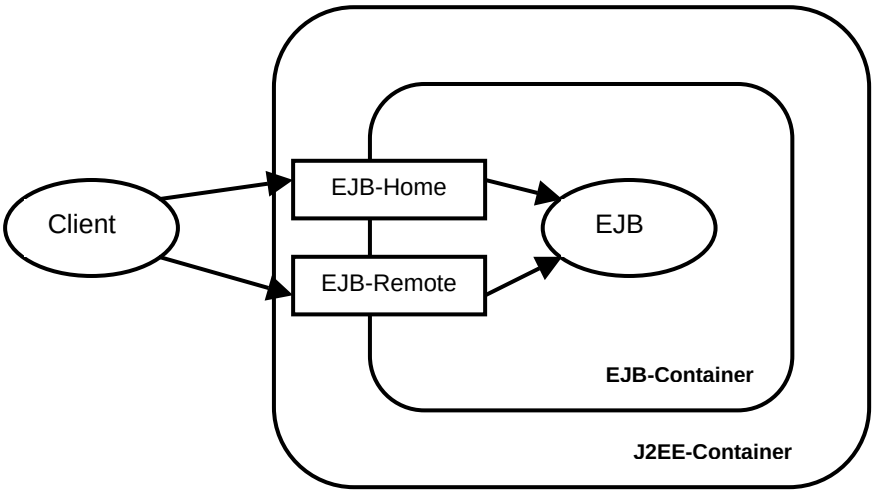


Abb. 3: EJB-Architektur /EJB 2.0/

Für jeden EJB muß ein Home- und Remote-Interface definiert werden. Das Home-Interface stellt Methoden zum Erzeugen und Löschen eines EJB bereit. Das Remote-Interface definiert Methoden, mit denen die Anwendungslogik eines EJB angesprochen werden kann. Mit Hilfe der beiden Interfaces kann ein Client ein EJB-Objekt erzeugen und seine Anwendungslogik nutzen. Grundsätzlich wird zwischen *Entity*- und *Session-Beans* unterschieden. Eine Session-Bean arbeitet als Agent eines einzigen Clients und kann als seine logische Extension betrachtet werden. Eine Entity-Bean repräsentiert die Objektsicht auf Daten einer Datenbank.

Die Programmierung in der J2EE-Umgebung wird durch die zahlreichen Bibliotheken, die „Sun Microsystems“ zur Verfügung stellt, sehr gut unterstützt. Die nächste Tabelle faßt die Java-Bibliotheken zusammen, die mit der J2EE-Spezifikation bereit gestellt werden (Abb. 4a, b).

Bezeichnung	**Bibliothek**	**Erläuterung**
Enterprise JavaBean	javax.ejb	Unterstützt die Implementierung von EJBs
JDBC	javax.sql	Unterstützt den Zugriff auf relationale Datenbanksysteme
Java Servlet	javax.servlets	Erweitert die Funktionalität eines Web-Servers
JavaServer Pages	javax.servlets.jsp	Unterstützt das Ablegen des Servlet-Codes in statische HTML-Seiten
Java Message Service	javax.jms	Unterstützt das Erzeugen, Senden und Empfangen von Nachrichten
Java Transaction	javax.transaction	Unterstützt das Transaktionsmanagement
Java Mail	javax.mail	Unterstützt das Versenden und Empfangen von Emails
JavaBeans Activation Framework	javax.activation	Wird von JavaMail benötigt

Abb. 4a: J2EE-Bibliotheken

Bezeichnung	Bibliothek	Erläuterung
Java API for XML	javax.jaxp	Unterstützt die XML-Verarbeitung
RMI-IIOP	javax.rmi, javax.rmi.CORBA	Unterstützt den Remote-Object-Call
JNDI	javax.naming	Unterstützt Naming- und Directory-Dienste
J2EE Connector API	javax.conn	Unterstützt die Integration von Fremdsystemen
Java Authentication and Authorization Services	javax.jaas	Unterstützt Authentication und Authorization

Abb. 4b: J2EE-Bibliothekten (Fortsetzung)

Zusammenfassend läßt sich sagen, daß die J2EE-Architektur die Implementierung von Anwendungssystemen aus folgenden Gründen erleichtert:

- Die J2EE-Architektur ist komponentenbasiert und plattformunabhängig.
- Die Anwendungslogik ist in wiederverwendbare, skalierbare, verteilbare und standardisierte Komponenten organisiert.
- Der J2EE-Server und -Container stellen alle notwendigen Dienste, wie z.B. Transaktions-management, zur Verfügung, so daß Produktivität und Qualität gesteigert werden können.

5.2 PMS-Architektur

Ausgehend vom vorgestellten Prototypen in Kapitel 4, stellen wir nun eine auf der J2EE-Technologie basierende Architektur für das Prozeßmanagement-System (PMS) vor. Die Architektur von PMS ist in der Abbildung 5 dargestellt. Ihre Schichten werden im folgenden von unten nach oben beschrieben.

5.2.1 DBS/Repository

Im Verlauf eines Projektes entstehen zahlreiche Ergebnisse, wie z.B. UML-Diagramme, Excel-Tabellen oder Word-Dokumente, die gespeichert werden müssen. Als Datenablage kann ein Datenbanksystem oder ein Repository-System eingesetzt werden. Wird ein Datenbanksystem eingesetzt, so wird die Datenverwaltung durch den J2EE-Server systematisch unterstützt. Zum Ablegen von Daten können nämlich Entity-Beans verwendet werden, für deren Persistenz der EJB-Container verantwortlich ist.

Als einzusetzendes Datenbanksystem empfehlen wir das marktführende Produkt „Oracale 9i“ /www.oracale.com/ (kostenlose Demoversion erhältlich).

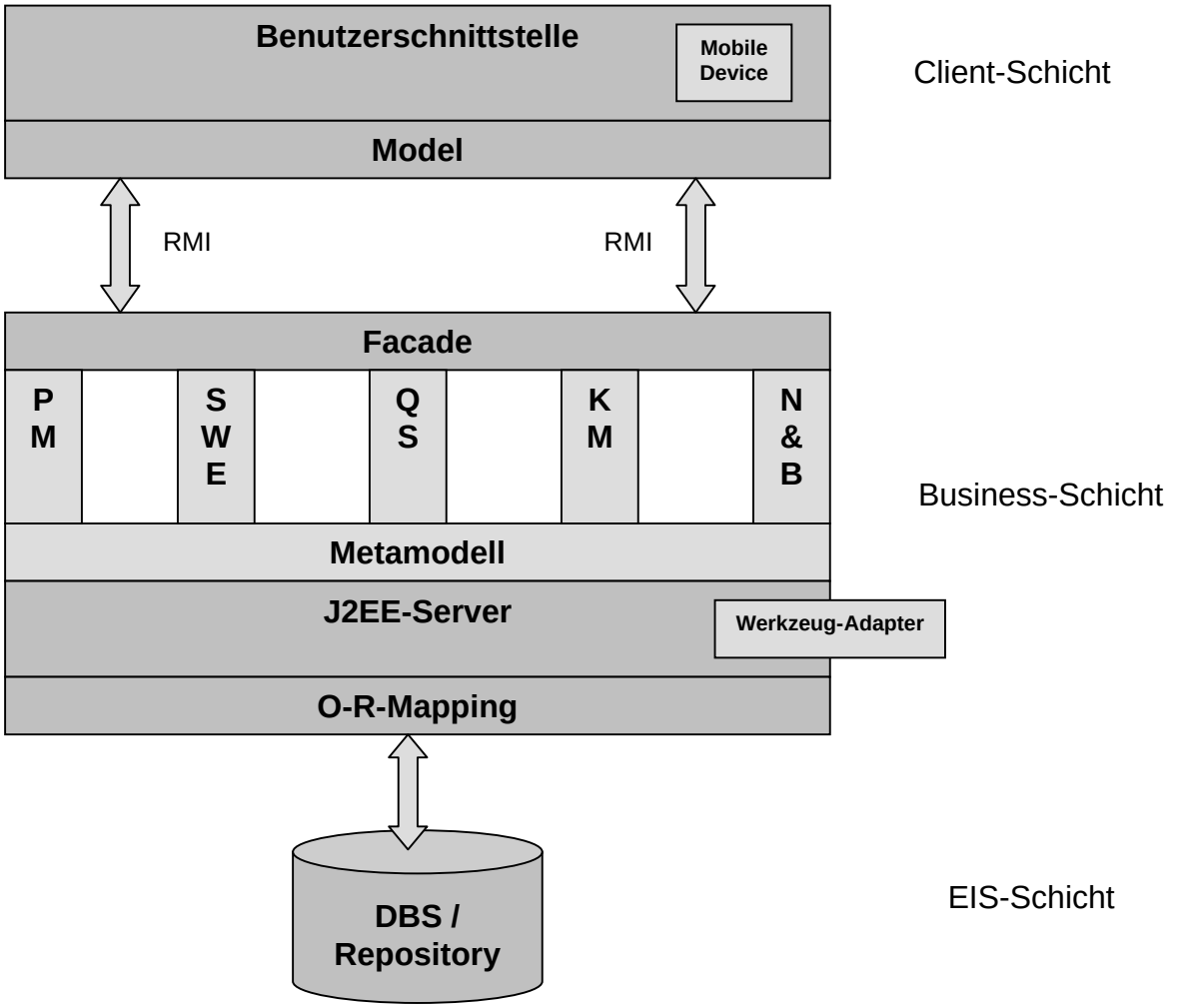


Abb. 5: PMS-Architektur

5.2.2 O-R-Mapping

Bei der Entwicklung eines objektorientierten Systems arbeiten wir mit Klassen und Objekten. Diese können inzwischen in objektorientierte Datenbanksysteme abgelegt werden. Die objektorientierten Datenbanksysteme befinden sich aber noch in einem frühen Entwicklungsstadium. Es gibt zwar einige Produkte auf dem Markt, diese weisen aber noch erhebliche Mängel auf, wie z.B. sehr niedrige Performanz im Verglich zu relationalen Datenbanksysteme. Daher werden zur Zeit hauptsächlich relationale Datenbanksysteme eingesetzt. Objekte und Relationen unterscheiden sich aber in vielerlei Hinsicht. Die folgende Tabelle faßt die Unterschiede zusammen (Abb. 6).

Nr.	Objekte	Relationen
1	Objekte werden zu Klassen abstrahiert	Relationen werden zu Tabellen abstrahiert
2	Objekte haben Merkmale	Relationen bestehen aus Feldern
3	Objekte haben eine interne Objekt-ID, d.h. eine Laufzeitadresse	Relationen werden durch einen Primär-Schlüssel identifiziert
4	Objekte haben Referenzen auf andere Objekte	Relationen beziehen sich auf andere Relationen mit Fremdschlüsseln
5	Attribute können dynamisch typisiert sein	Felder sind statisch typisiert
6	Komplexe Beziehungen können mit Hilfe	Es ist schwer, komplexe Beziehungen

	der Vererbung, Assoziation, Aggregation und Komposition leicht abgebildet werden	abzubilden (nur durch Fremdschlüssel)
7	Objekte können in heterogenen Sammlungen angeordnet werden	Jede Zeile einer Tabelle hat die gleichen Attribute

Abb. 6: Unterschiede zwischen Relationen und Objekte

Damit Klassen und Objekte in Tabellen und Relationen transformiert werden können, ist ein *objekt-relationales-Mapping* (O-R-Mapping) erforderlich. Es werden drei Strategien beim O-R-Mapping unterschieden: *Top-down*, *Bottom-up* und *Meet in the middle*.

Wir erläutern kurz das Prinzip des Top-down-Ansatzes, wo ein Objektmodell vorliegt, aber kein Datenbank-Schema existiert. In diesem Fall wird ein passendes Datenbank-Schema wie folgt entwickelt:

- definiere eine Tabelle für jede persistente Klasse,
- definiere Spalten, die die Attribute der Klasse abbilden,
- definiere Fremdschlüssel für die Beziehungsattribute.

Die einzelnen Schritte werden mit der folgenden Abbildung illustriert.

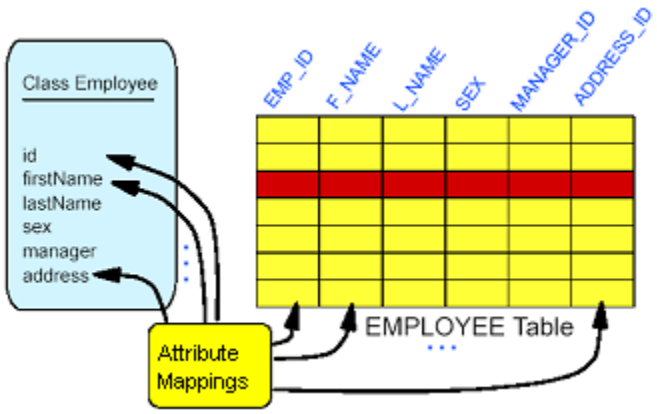


Abb. 7: O-R-Mapping: Top-down-Ansatz

Die meisten J2EE-Server implementieren O-R-Mapping-Strategien selbst oder integrieren entsprechende Werkzeuge, die das Mapping übernehmen. Ein Beispiel für solch ein Werkzeug ist „PersistencePowerTier“ /www.persistence.com/ (kostenlose Demoversion erhältlich).

5.2.3 J2EE-Server

Der J2EE-Server verwaltet EJB-Container bzw. EJBs und stellt Low level-Dienste zur Verfügung. Da der J2EE-Standard sich durchzusetzen scheint, haben sich viele Hersteller auf die Implementierung von J2EE-Servern spezialisiert und bieten entsprechende Produkte an. J2EE-Server werden auch als Application-Server (AS) bezeichnet. Die bekanntesten Application-Server sind: „BEA WebLogic“, „Borland AppServer“, „IBM WebSphere“, „JonAS“, „Iona iPortal Application Server“ und „Oracle AS“. Eine vergleichende Übersicht findet sich auf: www.flashline.com/Components/appservermatrix.jsp

Für PMS empfehlen wir den marktführenden Application-Server „BEA WebLogic“ /www.bea.com/ (kostenlose Demoversion erhältlich).

Die Entwicklung eines Anwendungssystems erfordert den Einsatz zahlreicher Werkzeuge, wie z.B. Text-Editoren, UML-, Management- oder Test-Tools. Zur Integration solcher Werkzeuge in PMS dient der „Werkzeug-Adapter“ (Abb. 5). Die Programmierung eines solchen Adapters kann auf unterschiedliche Weisen erfolgen. Wir wollen die standardisierte „J2EE Connector Technology“ nutzen. Diese Technologie ist Bestandteil des J2EE-Standards und wird von Werkzeug-Herstellern unterstützt. Das bedeutet, daß Adapter ohne Eigen-Programmierung übernommen werden können. Die nächste Abbildung verdeutlicht das Prinzip dieser Technologie (Abb. 8).

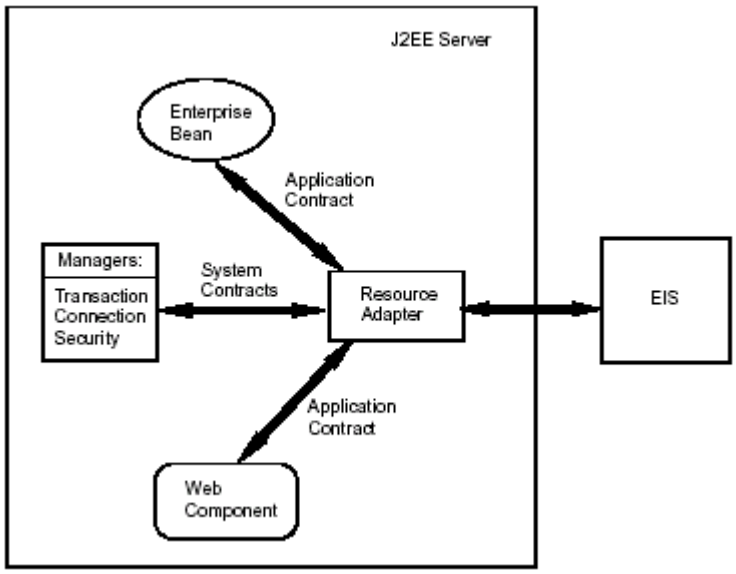


Abb. 8: J2EE Connector Technology /J2EE 1.3/

In der obigen Abbildung repräsentiert „EIS“ (Enterprise Information System) ein beliebiges Anwendungsystem. Der „Resource Adapter“ ist eine in dem J2EE-Server integrierte Komponente, die die in J2EE definierten „Connector-Interfaces“ implementiert. Zu den „Connector-Interfaces“ gehören Schnittstellen zu EJBs und Web-Komponenten (z.B. Java Servlets). Außerdem muß der „Resource Adapter“ System-Schnittstellen („System Contracts“) implementieren, so daß er über Mechanismen, wie z.B. Transaktions- oder Sicherheitsdienste, verfügen kann.

Weitere Techniken zur Integration von Werkzeugen ist der Einsatz von XML /JAXP 1.1/, die Verwendung von COM-Bridges bei Windows-Produkten /J2EE CAS COM Bridge 1.0/ oder die Nutzung des Native-Interfaces /JNI 1.1/.

5.2.4 Metamodell

Die Grundlage für die weiteren Schichten bildet das Metamodell für das EOS-Modell. Das Metamodell wurde von /Beyer, 2001/ entwickelt und wird hier für PMS erweitert (*IAktivität, IProdukt*).

Die nächste Abbildung illustriert das Metamodell (Abb. 9).

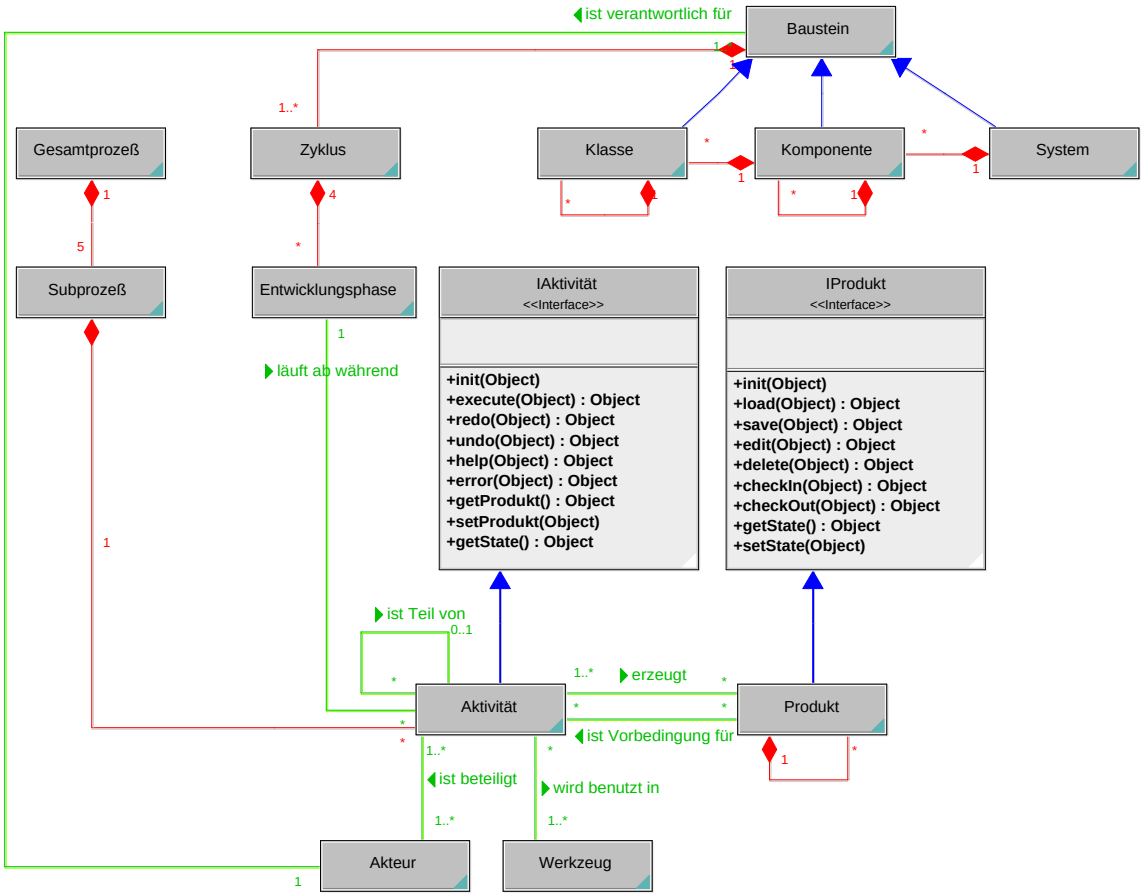


Abb. 9: Metamodell für EOS

Das Metamodell bildet die Grundelemente (z.B. Bausteine, Zyklen) des EOS-Modells ab. Für PMS sind in erster Linie die Klassen „Aktivität“ und „Produkt“ interessant, da sie Aktivitäten und die daraus resultierenden Produkte repräsentieren. Jede Aktivität muß das Interface „IAktivität“ implementieren. Dieses Interface beinhaltet Methoden zur Initialisierung, Verarbeitung und zur Statusabfrage einer Aktivität. Produkte müssen das Interface „IProdukt“ implementieren, das Methoden zur Initialisierung, Ablage und Statusabfrage von Produkten zur Verfügung stellt. Die Klasse „Produkt“ ist ein Kandidat für ein Entity-Bean, während „Aktivität“ als ein Session-Bean implementiert werden müßte.

5.2.5 Teilprozesse

Die einzelnen Teilprozesse „Projektmanagement (PM)“, „Software-Entwicklung (SWE), „Qualitätssicherung(QS)“, „Konfigurationsmanagement (KM)“ und „Nutzung und Bewertung (NB)“ basieren auf dem Metamodell. Die Implementierung eines Teilprozesses bedeutet, daß für die zugehörigen Aktivitäten und Produkte die Schnittstellen „IAktivität“ und „IProdukt“ implementiert werden müssen. Welche Aktivitäten und Produkte ein Teilprozeß aufweist, wurde bereits in Kapiteln 3 beschrieben.

Im Kapitel 4 wurden die Kontextmenüs von Aktivitäten und Produkten dargestellt. Beide Kontextmenüs verfügen z.B. über den Menüpunkt „Öffnen“. Das Betätigen dieses Menüpunktes würde einem Aufruf der Methode „execute“ („IAktivität“) bzw. der Methode

„edit“ („IProdukt“) bewirken. Die Kommunikation zu den Benutzeroberflächen verläuft über die Schicht „Facade“, die die Business-Schicht (Abb. 5) nach außen abkapselt.

5.2.6 Benutzerschnittstelle

In dieser Schicht wird die Benutzeroberfläche implementiert, mit der die Anwender PMS steuern und nutzen können. Im letzten Kapitel wurde eine entsprechende Oberfläche vorgeschlagen. Eine andere Möglichkeit der Oberflächengestaltung wäre der Einsatz von dynamischen HTML-Seiten via *JavaServer Pages* und *Java Servlets*. Bei der Kommunikation zur Business-Schicht machen wir von dem „Facade-Model“ Muster Gebrauch. Die Schicht „Model“ (Abb. 5) kapselt die Oberflächen ab und kommuniziert über „Facade“ mit der Business-Schicht mit Hilfe des RMI-Protokolls.

Die vorgeschlagene Architektur erlaubt es prinzipiell, daß PMS via „Mobile Devices“ (z.B. Palm-Organiser oder WAP-fähige Handys) gesteuert werden kann. Die Programmierung dieser Funktionen sollte sich mit „Java 2 Platfom Micro Edition“ (J2ME) relativ einfach gestalten.

Ausgehend vom Prototypen im Kapitel 4 wurde eine mehrschichtige Architektur für PMS entworfen. Bei der Entwicklung der Architektur wurde die J2EE-Standard (Java 2 Platform Enterprise Edition) herangezogen, so daß die heutigen Anforderungen (z.B. Transaktionsmanagement oder Sicherheitsmechanismen) an ein Anwendungssystem befriedigt werden können. Die Architektur besteht aus einer Client-Schicht (*Benutzerschnittstelle*, *Mobile Device*, *Model*), einer Business-Schicht (*Facade*, *Teilprozesse*, *Metamodell*, *J2EE-Server*, *Werkzeug-Adapter*, *O-R-Mapping*) und einer EIS-Schicht (*DBS* / *Repository*).

6 Abschlußbemerkung

Die Software-Entwicklung verläuft in der Regel evolutionär, d.h. als eine Folge von Erweiterungs- und Anpassungszyklen, beruhend auf Erfahrung, Nutzung und Revision. Viele der bekannten Vorgehensmodelle, wie z.B. „Rational Unified Process“ (RUP), berücksichtigen unter anderem diesen evolutionären Aspekt der Software-Entwicklung ungenügend. Mit dem *EOS-Modell* versuchen wir, dieses Defizit zu überbrücken.

Das EOS-Modell, ein Vorgehensmodell für die evolutionäre, objektorientierte Software-Entwicklung (EOS), ist durchgängig objektorientiert und berücksichtigt den evolutionären Charakter der Software-Entwicklung. Die Systementwicklung findet anhand der Gliederungseinheiten System, Komponente/Subsystem und Klasse statt. Diese Bausteine werden jeweils in einem zyklischen Entwicklungsprozeß, bestehend aus den Phasen Analyse, Entwurf, Implementierung und operationeller Einsatz, entwickelt.

In der ursprünglichen Fassung ist das EOS-Modell methodenunabhängig. Ein Ziel der Dissertation war es, konkrete Methoden für die Subprozesse „Projektmanagement“, „Software-Entwicklung“, „Qualitätssicherung“, „Konfigurationsmanagement“ und „Nutzung und Bewertung“ vorzuschlagen. Das Vorschlagen von Methoden bedeutet, zu klären „Wer tut etwas?“, „Wie tut er es?“ und „Was produziert er als Ergebnis?“. Ausgehend von der Methoden-Definition wurde als ein zweites Ziel untersucht, wie eine systematische Werkzeug-Unterstützung für das EOS-Modell aussehen könnte.

Da das EOS-Modell sich an Leitlinien wie *Hierarchischer Systemaufbau*, *Zyklische Entwicklung*, *Objektorientierung als durchgängige Entwicklungsmethodik*, *Einbezug von Erprobung und Nutzung* und *Weiter- und Wiederverwendung von Software-Bausteinen* orientiert, mußten speziell auf das EOS-Modell zugeschnittene Methoden definiert werden. Die definierten Methoden für die einzelnen Subprozesse haben wir in einem Prozeß-Handbuch (vgl. Anhang) zusammengefaßt, um es Projektmitgliedern in der Praxis zur Verfügung zu stellen.

Den Subprozeß „Projektmanagement“ haben wir in „Projekt-Initialisierung und Planung“, „Projekt-Steuerung“ und „Projekt-Abschluß“ gegliedert und dabei jeweils entsprechende Methoden vorgeschlagen. Das *Projektmanagement* sollte immer gegenstandsorientiert erfolgen. Das bedeutet, daß die Bausteine als Planungsgrundlage herangezogen wurden. Die Zyklen eines Bausteins und die an ihm ausgeführten Tätigkeiten Analyse, Entwurf, Implementierung und operationeller Einsatz werden für die Planung genutzt. Für eine erfolgreiche Projektplanung ist die Kenntnis des Entwicklungsaufwands entscheidend. Zu diesem Zweck wurde das CEOS-Verfahren (Cost estimation for EOS projects), ein Schätzverfahren für den Entwicklungsaufwand von evolutionären, objektorientierten Projekten, vorgeschlagen und vollständig implementiert.

Für die Bausteine System, Komponente/Subsystem und Klasse wurden, bezogen auf die einzelnen Entwicklungsphasen Analyse, Entwurf, Implementierung und operationeller Einsatz, entsprechende Methoden für die *Software-Entwicklung* formuliert. Insbesondere wurde die standardisierte Modellierungssprache UML durchgängig in die Software-Entwicklung einbezogen.

Die *Qualitätssicherung* im EOS-Modell wird in „Prüfung planen“, „Prüfung vorbereiten“ und „Prüfung durchführen“ gegliedert. Im ersten Teil wird die Prüfung eines Produkts oder einer

Aktivität im Detail geplant, indem der Prüfgegenstand und das Prüfverfahren festgelegt wurden. Das Ergebnis der Planung wird in einem QS-Handbuch verbindlich festgeschrieben. Im zweiten Teil werden die Prüfumgebungen eingerichtet, die Prüffälle definiert und die Prüfprozeduren entwickelt. Zuletzt werden die vorgesehenen Prüfungen durchgeführt und die Ergebnisse dokumentiert.

Den Subprozeß *Konfigurationsmanagement* haben wir in die Teile „Strukturierung“ und „Fortschreibung“ der Bausteinbibliothek gegliedert. Bei der „Strukturierung“ wird die Systemarchitektur auf die Bausteinbibliothek abgebildet. Damit wird die Grundlage zur Verwaltung von Produkten geschaffen, die während der Entwicklung eines Bausteins entstehen. Ausgehend von definierten Rollen werden Benutzergruppen abgeleitet. Es werden Benutzer mit entsprechenden Rechten auf die Bausteinbibliothek angelegt. Die Struktur der Bausteinbibliothek wird bei der „Fortschreibung“ mit Inhalten gefüllt. Das bedeutet, daß Produkte mit Versionen und Varianten in die Bausteinbibliothek aufgenommen und verwaltet werden.

Der Kern des Subprozesses *Nutzung und Bewertung* ist es, die „Stimme der Nutzer“ hinsichtlich von Verbesserungen zu berücksichtigen. Dazu wurde exemplarisch das Beschwerdemanagement eingeführt, das die Stimulierung, Annahme, Bearbeitung und Auswertung von Beschwerden systematisch unterstützt.

Zur Illustration eines Werkzeugs, das das EOS-Modell systematisch unterstützt, haben wir prototypisch ein Prozeßmanagement-System (PMS) implementiert. Der in Java geschriebene Prototyp bestand im Kern aus einer *Systemstruktur* und einer *Prozeßstruktur*. Mit Hilfe der *Systemstruktur* konnte die Baustein-Struktur eines EOS-Projekts abgebildet werden. Ausgehend von der Baustein-Struktur wurden mit der *Prozeßstruktur* die einzelnen Subprozesse von EOS mittels Aktivitätsdiagrammen beschrieben. Der Prototyp enthält eine Reihe von Komponenten, die vollständig implementiert sind und für zukünftige Projekte wiederverwendet werden können. Hierzu gehören z.B. die „Aufwandsschätzung von EOS-Projekten“, der „XML-Systembaum“ oder die „UML-Diagramme“.

Ausgehend vom Prototypen wurde eine mehrschichtige Architektur für PMS entworfen. Bei der Entwicklung der Architektur wurde der J2EE-Standard (Java 2 Platform Enterprise Edition) herangezogen, so daß heutige Anforderungen(z.B. Transaktionsmanagement oder Sicherheitsmechanismen) an ein Anwendungssystem berücksichtigt werden konnten. Die Architektur besteht aus einer Client-Schicht (*Benutzerschnittstelle*, *Mobile Device*, *Model*), einer Business-Schicht (*Facade*, *Subprozesse*, *Metamodell*, *J2EE-Server*, *Werkzeug-Adapter*, *O-R-Mapping*) und einer EIS-Schicht (*DBS* / *Repository*).

Zusammenfassend läßt sich sagen, daß mit dieser Arbeit ein vollständiger Methoden-Baukasten und ein prototypisches Werkzeug für die systematische Durchführung von evolutionären, objektorientierten Projekten zur Verfügung gestellt wird.

In dem Fachgebiet *Software Process Improvement* werden Strategien zur Einführung von Prozeßmodellen im Unternehmen vorgeschlagen. Die vorgeschlagenen Verfahren sind in der Regel für beliebige Prozeßmodelle gültig und können daher auch für das EOS-Modell verwendet werden. McFeeley definiert z.B. sechs Schritte, die bei der Einführung eines neuen Prozeßmodells in einem Unternehmen berücksichtigt werden sollten /McFeely 1996/. Hierbei wird zunächst der vorhandene Prozeß analysiert und bewertet. Die Stärken des Prozesses sollten auch weiterhin beibehalten werden. Die Schwächen werden genutzt, um die Projektmitglieder und das Management von der Notwendigkeit von Optimierungen zu

überzeugen. Ausgehend von der Ist-Analyse werden Ziele für die Einführung definiert und mögliche Risiken identifiziert. Danach wird die Prozeß-Einführung geplant, durchgeführt und abschließend bewertet. Die Einführung sollte nicht auf einen Schlag erfolgen, sondern stets schrittweise durchgeführt werden.

Bei der Prozeß-Einführung in einem Unternehmen spielt aber auch die Auswahl des ersten Projekts eine entscheidende Rolle. Bei dem ersten Projekt sollte es sich um ein „echtes“ Projekt handeln, da Projekte mit einem Fallstudien-Charakter nicht ernstgenommen werden. Es sollte ein Projekt mittlerer Größe sein, da Erfolge bei kleinen Projekten oft nicht anerkannt werden. Große Projekte besitzen eine hohe Komplexität und bergen Risiken. Daher sollten große Projekte erst dann angegangen werden, wenn ausreichende Erfahrungen über das neue Prozeßmodell vorliegen. Das Projektteam sollte unmittelbar vor der Anwendung des Prozeßmodells intensiv geschult werden.

Anhang A: Prozeß-Handbuch

Im folgenden werden Methoden für die fünf Subprozesse des EOS-Modells in Form eines Prozeß-Handbuchs beschrieben.

☑ Das objektorientierte Paradigma und die Modellierungssprache „UML“ werden als bekannt vorausgesetzt.

ⓘ

Anhang A: Prozeß-Handbuch

Im folgenden wird ein Prozeß-Handbuch für das EOS-Modell vorgestellt. Zur Definition des Handbuchs haben wir uns an dem Prozeßmodell für das auf UML basierende CASE-Werkzeug „objectiF“ /objectiF 4.5, microTOOL GmbH/ bzw. das V-Modell /www.v-modell-iabg.de/ orientiert, ziehen aber /Jacobson, Booch, Rumbaugh 1999/, /Kruchten 2000/, /Booch 1994/, /Jacobson 1993/, /Coad, Yourdon 1991/, /Rumbaugh 1991/, /Shlaer, Mellor 1991/, /Dröschel 1998/, /Larman 2001/, /Müller-Ettrich 1998/ und /Oestereich 1999/ hinzu.

Aktivitäten werden den Projektmitgliedern über Rollen zugeordnet. Eine Rolle steht für Kenntnisse, Fähigkeiten und Erfahrungen, die benötigt werden, um eine spezielle Aktivität durchführen zu können. Eine Rolle steht also für eine Qualifikation und nicht für eine Person oder Institution. In Anlehnung an /IBM 1997/ werden in den nächsten Kapiteln folgende Rollen unterschieden:

Der ***Anwender*** wird als Vertreter der Endbenutzer bei der Anforderungsanalyse und zur Abnahme des Anwendungssystems beteiligt. Er verfügt über detaillierte Kenntnisse des Anwendungsbereichs.

Der ***Anwendungsbereichsexperte*** gehört dem IT-Bereich an und unterstützt den Anwender bei der Anforderungsanalyse. Er kennt die Geschäftsprozesse des Anwendungsbereichs und sorgt bei der Analyse und beim Entwurf dafür, daß die Probleme des Anwendungsbereichs gelöst werden. Er spricht die Sprache des Anwenders und verfügt über Abstraktionsfähigkeit.

Der ***Projektmanager*** ist für Planung und Organisation des Projekts zuständig. Er kommuniziert mit dem Auftraggeber, motiviert und koordiniert das Team. Er kennt den Anwendungsbereich, die aktuellen Basistechniken und verfügt über betriebswirtschaftliches und juristisches Wissen.

Der ***Projektleiter*** schätzt den Entwicklungsaufwand ab, plant Termine und Ressourcen, steuert und kontrolliert das Projekt. Er kennt den Anwendungsbereich, Entwicklungsmethoden und Werkzeuge, besitzt technisches und betriebswirtschaftliches Grundwissen.

Der ***Sytemarchitekt*** entwirft die Gesamtarchitektur des Anwendungssystems und trifft Entscheidungen über technische Anforderungen, wie z.B. die Definition von Schnittstellen oder die Auswahl von Komponenten zur Wiederverwendung. Er verfügt über fundiertes technisches Wissen und kann Architekturkonzepte erstellen und bewerten. Er kann abstrahieren, strukturieren und kommunizieren.

Der ***Systemanalytiker*** modelliert das zu entwickelnde System aus fachlicher Sicht, indem er die definierten Anforderungen in einem Anwendungsfallmodell umsetzt. Er kennt den Anwendungsbereich und beherrscht Methoden und Werkzeuge. Er kann abstrahieren, strukturieren und kommunizieren.

Der ***Softwaredesigner*** bereitet die Implementierung vor, indem er Komponenten und Klassen entwirft. Er beherrscht die einzusetzenden Methoden und Werkzeuge und kann kommunizieren, abstrahieren und strukturieren.

Der ***Softwareentwickler*** setzt den Entwurf in Quellcode um. Er kennt die Systemarchitektur, die Schnittstellen und die geforderten Programmiersprachen. Strukturierung, Erkennung und Bewertung von Abhängigkeiten gehören zu seinen Stärken.

Der ***Technische Autor*** entwickelt zielgruppenorientierte Anwenderdokumentationen und Schulungsunterlagen. Er verfügt über technisches Verständnis und kann die wesentlichen Aspekte des Anwendungssystems didaktisch und verständlich beschreiben.

Der ***Qualitätsprüfer*** ist für die Durchführung von Qualitätssicherungs-Maßnahmen verantwortlich. Er kennt den Zweck des Anwendungssystems und beherrscht Prüf- und Testmethoden. Er kann analytisch denken und zwischen Ursache und Wirkung unterscheiden.

Der ***Anwenderbetreuer*** unterstützt die Anwender, wenn das Anwendungssystem in Betrieb ist. Er klärt Fragen der Anwender und erfaßt ihre Anmerkungen, Probleme und Beschwerden. Er muß kommunikations- und kritikfähig sein und über Konfliktlösungskompetenz verfügen.

Der ***Projektbibliothekar*** verwaltet die entstehenden Bausteine und die damit verknüpften Produkte. Er richtet die Bausteinbibliothek ein und verwaltet sie. Er ist kommunikationsfähig und beherrscht das eingesetzte Konfigurationsmanagement-System.

1 Projektmanagement

Der erfolgreiche Abschluß eines Projekts hängt wesentlich von der Güte des Projektmanagements ab. Denn Fehlentscheidungen durch das Management sind in der Regel schwer zu korrigieren. Daher ist das Projektmanagement eine große Herausforderung und bringt zugleich eine große Verantwortung mit sich. Die Planung und Steuerung eines evolutionären, objektorientierten Projekts stellt noch weitere Anforderungen an das Projektmanagement, wie z.B. das Planen von Zyklen und Revisionspunkten. Kurz gesagt, Projektmanagement wird nicht leichter, berücksichtigt aber stärker die Dynamik eines Projekts.

Das Projektmanagement im EOS-Modell haben wir in die Teilabschnitte „Projekt-Initialisierung und Planung“, „Projekt-Steuerung“ und „Projekt-Abschluss“ gegliedert, die wir in den nächsten Abschnitten erläutern werden.

1.1 Projekt-Initialisierung und Planung

Gegenstand der Projekt-Initialisierung und -Planung ist es, die Projektziele zu definieren, das Projekt abzugrenzen, Risiken zu analysieren und organisatorische Maßnahmen, wie z.B. die Planung der Wiederverwendung oder des Konfigurationsmanagements, vorzubereiten. Die Ergebnisse der Projekt-Initialisierung werden in einem zentralen Dokument, dem Projekthandbuch, zusammengefaßt. Den Prozeß der Projekt-Initialisierung und -Planung haben wir wie folgt definiert (Abb. 2):

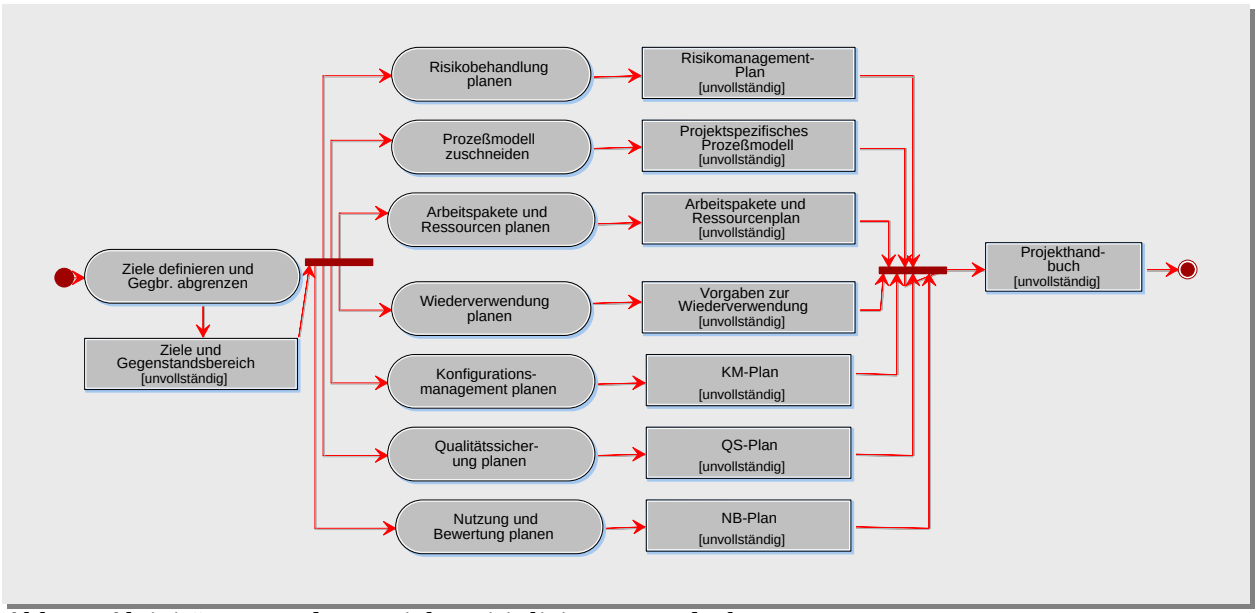


Abb. 1: Aktivitätstypen der Projekt-Initialisierung und Planung

1.1.1 Ziele definieren und Gegenstandsbereich abgrenzen

Zweck

Das Ziel dieses Aktivitätstyps ist es, unter allen Projektbeteiligten ein einheitliches Verständnis dafür zu schaffen, was mit der geplanten Entwicklung beabsichtigt wird. Hierzu

sind folgende Fragen zu klären: Aus welchem Grund wird das Anwendungssystem entwickelt? Welche Ziele werden mit dem zu entwickelnden Anwendungssystem verfolgt?

Neben der Zieldefinition muß der Gegenstandsbereich des Projekts abgegrenzt werden. Eine detaillierte Lösung für die erkannten Probleme braucht man nicht anzugeben. Jedoch sollte ein grundsätzlicher Lösungsansatz und Vorstellungen über eine mögliche Grobarchitektur vorhanden sein, um daran die Planungsschritte des Projektmanagements zu orientieren.

Beteiligte

Ausführend: Auftraggeber, Projektmanager
Mitwirkend: Anwender

Vorgehen

Wenn das Projekt Bestandteil des Geschäftsprozeßmanagements ist, so liegt ein Projektauftrag mit einer Abgrenzung der zu unterstützenden Geschäftsprozesse und eine grobe Anforderungsdefinition mit möglichen Lösungsvorgaben vor. In diesem Fall müssen die für die Geschäftsprozeßmodellierung definierten Ziele in solche übertragen werden, die mit dem zu realisierenden Anwendungssystem erreicht werden sollen. Die Ziele müssen möglichst quantifiziert werden, damit nach Projektabschluß nachgewiesen werden kann, daß die angestrebten Ziele tatsächlich erreicht wurden. Liegt keine Geschäftsprozeßmodellierung vor, so sind folgende Schritte zu durchlaufen:

- Ziele identifizieren,
- Ziele quantifizieren und bewerten,
- Lösungswege ermitteln,
- Lösungswege bewerten,
- Gegenstandsbereich abgrenzen.

Ziele identifizieren

Mit dem Auftraggeber sind die Ziele zu definieren, die mit dem zu entwickelnden Anwendungssystem verfolgt werden. Es werden folgende Ziele unterschieden:

- *wirtschaftliche Ziele:* Ziele, die Kosten und die finanziell meßbaren Erträge beinhalten. Sie werden z.B. in Kennziffern, wie laufende Kosten oder Kosteneinsparungen, ausgedrückt.
- *funktionelle Ziele:* Diese beschreiben Leistungen eines Systems, wie z.B. Sicherheitsmechanismen oder Transaktionsfähigkeit.
- *personelle Ziele:* Alle Ziele, die gewünschte oder unerwünschte personelle Auswirkungen zum Inhalt haben, wie z.B. Bedienungsfreundlichkeit.
- *soziale und gesellschaftliche Ziele:* Ziele, welche sich auf die Beachtung sozialer und gesellschaftlicher Auswirkungen, wie z.B. Umweltbelastung oder Entsorgungs-freundlichkeit, richten.
- *Projektablaufziele:* Das sind solche Ziele, die das Projekt direkt betreffen, wie z.B. Termin- oder Budget-Ziele.

Ziele quantifizieren und bewerten

Die im vorherigen Schritt identifizierten Ziele sind nun soweit wie möglich zu quantifizieren, damit nachgeprüft werden kann, ob sie erreicht worden sind. Ein Beispiel für ein quantifiziertes Ziel wäre: „Das Buchungssystem darf innerhalb von 5 Werktagen nicht länger als 30 Minuten ausfallen." Damit entschieden werden kann, in welchem Zyklus ein Ziel realisiert werden soll, sollte es bewertet werden. Zunächst ist eine Klassifizierung in „Muß-Ziele", „Soll-Ziele" und „Kann-Ziele" ausreichend.

Lösungsansätze ermitteln

Sofern der Projektauftrag keine Vorgaben enthält, sollte nach prinzipiellen Lösungsansätzen gesucht werden, mit denen die gesetzten Ziele erreicht werden könnten. Hierbei sollte eine erste vage Strukturierung des Anwendungssystems in Komponenten vorgenommen werden und eine grobe Systemarchitektur vorgeschlagen werden.

Lösungsansätze bewerten

Lösungen bestehen immer aus einer Reihe von Maßnahmen. Für die einzelnen Maßnahmen ist zu prüfen, ob sie zum Erreichen der quantifizierten Ziele beitragen. Alternative Lösungsansätze sollten in Hinsicht auf das Erreichen der definierten Ziele bewertet werden. Gegebenenfalls sollte eine Entscheidung über den ausgewählten Lösungsansatz getroffen und begründet werden.

Gegenstandsbereich abgrenzen

Auf der Basis des ausgewählten Lösungsansatzes (gegebenenfalls auch mehrere) ist der Teil des Geschäftsbereichs, der durch das zu entwickelnde Anwendungssystem tatsächlich maschinell unterstützt werden soll, grob zu definieren und zu beschreiben.

Ergebnisse

Die formulierten Ziele und die Abgrenzung des Gegenstandsbereichs können mit dem folgenden Beschreibungsschema dokumentiert werden. Funktionale Ziele können mit Hilfe von Anwendungsfall-Diagrammen beschrieben werden. Die erste grobe Systemstruktur kann mit Hilfe von Package-Diagrammen festgehalten werden.

Hinweise und Tipps

Für die Formulierung der Projektziele eignen sich folgende Grundsätze:

- Zielformulierung sollte lösungsneutral sein.
- Nicht nur positive sondern auch die Vermeidung negativer Wirkungen sollten beschrieben werden.
- Ziele sollten möglichst operational formuliert werden.
- Zwischen Muß- und Wünsch-Zielen sollte unterschieden werden.

Projektinhalt	⇨ Hier sollte der Anlaß und Hintergrund des Projekts dargestellt werden. Außerdem sollte der Auftraggeber/Anwender und der Untersuchungsbereich beschrieben werden. [**Hier** klicken]		
Gegenstands-bereich	⇨ Beschreiben Sie die Abläufe und Zuständigkeiten beim bestehenden Verfahren. Welche Probleme, Schnittstellen und Restriktionen gibt es? [**Hier** klicken]		
Ziele	**#**	**Beschreibung**	**Priorität**
	1	⇨ Beschreiben Sie hier die identifizierten Ziele und klassifizieren Sie diese (z.B. wirtschaftliche Ziele, funktionelle Ziele, personelle Ziele, soziale und gesellschaftliche Ziele, Projektablaufziele). [**Hier** klicken]	⇨ Nicht alle Ziele sind gleich wichtig. Vergeben Sie hier pro Ziel eine Priorität, z. B. zwischen 1 wie sehr hoch und 5 wie sehr gering. [**Hier** klicken]
	2	[**Hier** klicken]	[**Hier** klicken]
Lösungsansätze	**#**	**Beschreibung**	**Typ**
	1	⇨ Beschreiben Sie hier Ihren Lösungsansatz samt der entsprechenden Systemstruktur (Komponenten, Systemarchitektur). [**Hier** klicken]	⇨ Beschreiben Sie, ob es sich um eine kleine, mittlere oder große Lösung handelt. [**Hier** klicken]
	2	[**Hier** klicken] [**Hier** klicken]	[**Hier** klicken]
Hinweise	⇨ Welche Chancen bietet das Projekt, und welche Risiken birgt es? Was sind die Kosten und was der Nutzen? [**Hier** klicken]		

Abb. 2: Beschreibungsschema für Projekt-Ziele und Abgrenzung

1.1.2 Risikobehandlung planen

Zweck

Alles, was den Erfolg des Projekts gefährdet, ist ein Risiko, dem präventiv zu begegnen ist. Da alle Projekte Risiken bergen, ist es wichtig, sich dieser Risiken bewußt zu sein, um sie soweit wie möglich zu vermeiden. Das Ziel dieses Aktivitätstyps ist es, Risiken zu identifizieren und zu bewerten, damit entsprechende Abwehrmaßnahmen ermittelt und eingeleitet werden können.

Beteiligte

Ausführend: Projektmanager
Mitwirkend: Projektleiter

Vorgehen

Folgende Schritte sind zu durchlaufen:

- Risiken identifizieren,
- Risiken bewerten,
- Abwehrmaßnahmen ermitteln,
- Abwehrstrategie festlegen.

Risiken identifizieren

Das Ergebnis dieses Schritts ist es, eine projektspezifische Risikoliste zu ermitteln. Zur Identifikation kann die nachfolgende Klassifikation herangezogen werden:

- *Personelle Risiken:* Diese Art von Risiken besteht vor allem aus dem unerwartetem Ausfall von Projektbeteiligten, z.B. durch Krankheit, aber auch durch soziale oder technische Defizite der Mitarbeiter.
- *Technologische Risiken:* Neue Technologien werden verwendet, für die noch nicht genügend Erfahrungen gesammelt werden konnten.
- *Anwendungsbereichbezogene Risiken:* Sie entstehen, wenn die fachlichen Aufgaben zu komplex und undurchschaubar sind.
- *Anforderungsbezogene Risiken:* In der Regel können die Anwenderforderungen nicht vollständig erfaßt werden, so daß möglicherweise am Bedarf vorbei entwickelt wird.
- *Qualitätsrisiken:* Die Qualität des Ergebnisses ist unzureichend, weil die entsprechenden Qualitätsanforderungen nicht berücksichtigt wurden.
- *Planungsbedingte Risiken:* Sie ergeben sich unter anderem, weil der Zeitplan zu optimistisch geschätzt wird, so daß Planungen nicht eingehalten werden können.
- *Politische Risiken:* Neue Zielsetzungen und Prioritäten werden aufgrund von Veränderungen der Unternehmenskultur und Aufbauorganisation gesetzt.
- *Vertragsbedingte Risiken:* Sie entstehen, wenn ein Vertrag aufgrund externer oder interner Probleme nicht eingehalten werden kann.
- *Wirtschaftliche Risiken:* Äußere Faktoren, wie z.B. Verschlechterung der finanziellen Lage der Firma oder Veränderungen des Marktes, verursachen Risiken dieser Art.

Auf der Basis der obigen Checkliste können im Rahmen eines Workshops projektspezifische Risiken gefunden werden.

Risiken bewerten

Die im vorherigen Schritt identifizierten Risiken sind nun zu bewerten. Hierzu sind für jedes Risiko die Eintrittswahrscheinlichkeit und die möglichen Schäden abzuschätzen.

Diese Informationen sind in der folgenden Risikomatrix festzuhalten /Versteegen 2000/:

Risikoklasse \ Eintrittswahrsch.	1	2	3	4	5
A					
B					
C					
D					
E					

Legende:

Risikoklasse A: Abbruch des Projekts
Risikoklasse B: Budgetüberschreitung um 50%
Risikoklasse C: Budgetüberschreitung um 30%
Risikoklasse D: Mehraufwand für einen Zyklus
Risikoklasse E: Geringfügiger Mehraufwand für einen Zyklus

Risikoeintritt 1: sehr wahrscheinlich
Risikoeintritt 2: wahrscheinlich
Risikoeintritt 3: bedingt wahrscheinlich
Risikoeintritt 4: unwahrscheinlich
Risikoeintritt 5: nahezu ausgeschlossen

Abb. 3: Risikomatrix zur Bewertung von identifizierten Risiken

Je nach Projektumfang und Projektart wird die Beschreibung der Risikoklassen variiert werden. Als Faustregel gilt: Wenn zum jetzigen Zeitpunkt ein Risiko mit der Eintrittswahrscheinlichkeit 1 oder 2 in der Risikoklasse A existiert, so sollte ernsthaft darüber nachgedacht werden, ob mit dem Projekt begonnen werden sollte.

Abwehrmaßnahmen ermitteln

Nachdem die identifizierten Risiken mit Hilfe der Risikomatrix nach Prioritäten geordnet sind, müssen entsprechende Abwehrmaßnahmen ermittelt werden. Prinzipiell werden folgende Arten von Abwehrmaßnahmen unterschieden:

- *Risiko ausräumen:* Hierbei wird die Ursache des Risikos eliminiert. Ein unerfahrener Mitarbeiter kann z.B. durch Schulungen weitergebildet werden.
- *Risiko verringern:* Kann ein Risiko nicht ausgeräumt werden, so sollte es aktiv bekämpft werden, um die Auswirkungen und die Eintrittswahrscheinlichkeit zu minimieren. Durch die Einstellung von zusätzlichen Entwicklern kann man Ressourcen-Engpässen im Vorfeld begegnen.
- *Schadensbegrenzung:* Wenn die Eintrittswahrscheinlichkeit des Risikos nicht verringert werden konnte, so muß das Eintreten des Risikos akzeptiert werden. Es müssen aber Maßnahmen unternommen werden, mit denen die Schäden begrenzt werden können.
- *Notfall planen:* Wenn ein Risiko eintritt, so muß möglichst schnell reagiert werden. Daher ist ein Notfallplan zu erstellen, an dem sich die Projektmitglieder orientieren können.

Für die weiter oben definierten Risikoarten listen wir nun exemplarisch einige Gegenmaßnahmen auf:

- *Personelle Risiken:* Können vermieden werden, indem im Vorfeld genügend Mitarbeiter eingestellt werden. Defizite können durch gezielte Weiterbildung beseitigt werden.
- *Technologische Risiken:* Schulung der Projektmitarbeiter in neuen Technologien können solche Risiken verringern.

- *Anwendungsbereichbezogene und Anforderungsbezogene Risiken:* Zyklische Entwicklungsmethoden können solche Risiken verringern helfen.
- *Qualitätsrisiken:* Die QS-Maßnahmen sollten vom Projektmanagement besser koordiniert werden. Das EOS-Modell ermöglicht es, durch seine Revisionspunkte die Qualität bis auf die Klassen-Ebene zu kontrollieren und zu koordinieren.
- *Planungsbedingte Risiken:* Solche Risiken entstehen durch eine zu optimistische Planung, bedingt durch unrealistische Aufwandsschätzungen. Um dieses Risiko zu verringern, sollte der Entwicklungsaufwand kontinuierlich und systematisch ermittelt werden. Für das EOS-Modell wurde hierzu das CEOS-Verfahren entwickelt.
- *Politische Risiken:* Neue Zielsetzungen oder Veränderungen der Unternehmenskultur sollten nicht schlagartig, sondern nach und nach, vollzogen werden.
- *Vertragsbedingte Risiken:* Diese Art von Risiken läßt sich durch eine juristisch geschickte Formulierung des Vertrags verringern.
- *Wirtschaftliche Risiken:* Solche Risiken lassen sich zum Teil durch Kooperationen mit anderen Firmen vermindern.

Abwehrstrategie festlegen

Für die identifizierten Risiken sollten entsprechende Gegenmaßnahmen vorgesehen werden. Dabei ist zu beachten, daß durch die Kombination von Risiken neue Risiken entstehen können. Die Entwicklungszyklen sollten gegebenenfalls so geplant werden, daß die größten Risiken direkt angegangen werden können. Ziel sollte es sein, die Risiken so früh wie möglich zu minimieren, indem kritische Bausteine zuerst realisiert werden.

Ergebnisse

Der vorliegende Aktivitätstyp bildet mit den Aktivitätstypen „Ziele definieren und Gegenstandsbereich abgrenzen“ und „Entwicklungsstrategie festlegen“ den Inhalt einer Vorstudie. Diese dient als Entscheidungsgrundlage für die Fortsetzung des Projekts.

Die identifizierten Risiken können mit dem folgenden Beschreibungsschema dokumentiert werden.

Bezeichnung	⇨ Nennen Sie die Bezeichnung des Risikos. [**Hier** klicken]
Beschreibung	⇨ Beschreiben Sie hier das identifizierte Risiko. [**Hier** klicken]
Risikoklasse	⇨ Zu welcher Risikoklasse kann das identifizierte Risiko zugeordnet werden? ☐ Risikoklasse A: Abbruch des Projekts ☐ Risikoklasse B: Budgetüberschreitung um 50% ☐ Risikoklasse C: Budgetüberschreitung um 30% ☐ Risikoklasse D: Mehraufwand für ein Zyklus ☐ Risikoklasse E: Geringfügige Mehraufwand für einen Zyklus
Eintrittswahr-scheinlichkeit	⇨ Wie hoch ist die Eintrittswahrscheinlichkeit?

	☐ Risikoeintritt 1: sehr wahrscheinlich ☐ Risikoeintritt 2: wahrscheinlich ☐ Risikoeintritt 3: bedingt wahrscheinlich ☐ Risikoeintritt 4: unwahrscheinlich ☐ Risikoeintritt 5: nahezu ausgeschlossen
Abwehrmaß-nahmen	⇨ Nennen Sie die entsprechenden Abwehrmaßnahmen. Hierzu gehören auch Maßnahmen zur Schadens-Begrenzung. [**Hier** klicken]
Kosten der Abwehrmaß-nahmen	⇨ Wieviel Kosten verursachen die Abwehrmaßnahmen? [**Hier** klicken]
Kosten beim Auftreten	⇨ Welche Kosten und Schäden entstehen, wenn das Risiko trotz bzw. ohne Abwehrmaßnahmen auftreten würde? [**Hier** klicken]
Status	⇨ Beschreiben Sie den aktuellen Status des Risikos. ☐ offen ☐ in Bearbeitung ☐ reduziert ☐ eliminiert
Hinweise	⇨ Erkannte Risiken, Probleme oder Zusammenhänge sind hier aufzuführen. [**Hier** klicken]

Abb. 4: Beschreibungsschema für Risiken

Hinweise und Tipps

Maßnahmen zur Risikovermeidung sind nicht umsonst und müssen daher in der Projektplanung berücksichtigt werden.

1.1.3 Prozeßmodell zuschneiden

Zweck

Das EOS-Modell besitzt eine hohe Allgemeingültigkeit und muß auf die Gegebenheiten eines Projekts angepaßt werden. Das bedeutet, daß die Aktivitätstypen, Produkttypen und das Rollenmodell, bezogen auf das jeweilige Projekt, modifiziert und ergänzt werden müssen. Dieser Prozeß wird auch als *Tailoring* bzw. *Zuschneiden* bezeichnet, der im Rahmen des Aktivitätstyps „Prozessmodell zuschneiden“ durchgeführt wird. Das Ergebnis ist ein spezialisiertes Prozeßmodell, das nur solche Rollen, Aktivitätstypen und Produkttypen vorsieht, die für das konkrete Projekt relevant sind.

Beteiligte

Ausführend: Projektleiter
Mitwirkend: Projektmanager

Vorgehen

Folgende zwei Schritte müssen durchlaufen werden:

- Projektprofil definieren,
- Tailoring durchführen.

Projektprofil definieren

Um entscheiden zu können, auf welche Aktivitäts- und Produkttypen verzichtet werden kann, sollten zunächst die Charakteristika und die Besonderheiten des Projekts ausgearbeitet werden. Dies kann durch die Klassifikation des Projekts geschehen. Folgende für das Tailoring relevante Projekttypen können unterschieden werden /Goldberg, Rubin 1995/:

- *„Das-Erste-seiner-Art"-Projekt:* Hierbei handelt es sich um ein Projekt, das in dem Unternehmen zum ersten Mal durchgeführt wird. Da bei so einem Projekt wenig Erfahrung vorliegt, sollte EOS vollständig angewendet werden. In Abhängigkeit vom Projektumfang können aber zumindest die Beschreibungsschemata modifiziert werden.
- *„Variation-eines-Themas"-Projekt:* Bei dieser Art von Projekten wird aus einer existierenden Lösung durch Spezialisierung und Modifikation ein neues System erstellt. Da hier vorliegende Komponenten aus- und neugebaut werden, sollte viel Zeit für die Einarbeitung vorhanden sein. Daher sollten die Analyse-Tätigkeiten auf jeden Fall vollständig durchlaufen werden. Ansonsten können, wieder in Abhängigkeit vom Projektumfang, die Aktivitätstypen und Produkttypen angepaßt werden.
- *„Aus-alt-mach-neu"-Projekt:* Hierbei wird ein Altsystem neu geschrieben. Das bedeutet, daß die fachlichen Anforderungen nahezu unverändert bleiben, während die Systemarchitektur aufgrund von neuen Technologien womöglich komplett verändert wird. Daher sollten die Entwurfs-Tätigkeiten auf jeden Fall vollständig ausgeführt werden. Die anderen Tätigkeiten können je nach Projektgröße angepaßt werden.
- *„Entwicklung-von-wiederverwendbaren-Komponenten"-Projekt:* Prinzipiell unterscheiden sich Projekte, bei denen wiederverwendbare Komponenten entwickelt werden, kaum von denen, bei denen ein Anwendungssystem entwickelt wird. Im Detail müssen aber z.B. die Anforderungen aus Sicht vieler Projekte verstanden und spezifiziert werden, so daß die Analyse-Tätigkeiten eine besondere Rolle spielen. Außerdem wirkt die Qualitätssicherung in einem hohen Maß, da alle produzierten Bausteine gut dokumentiert und intensiv getestet sein müssen. Daher sollten die Analyse- und Qualitätssicherungs-Tätigkeiten vollständig durchlaufen werden.

Neben der oben genannten Klassifikation spielt die Projektgröße eine wesentliche Rolle. Das EOS-Modell und die definierten Methoden können für Projekte mit beliebigem Umfang eingesetzt werden. Bei kleinen Projekten werden z.B. die Projektmanagement Aktivitäten sehr reduziert ausfallen, während bei großen Projekten diese unverzichtbar sein werden.

Tailoring durchführen

Die nicht benötigten Aktivitäts- und Produkttypen sollten durchgestrichen bzw. entsprechend modifiziert werden. Das Rollenmodell sollte ebenfalls auf das aktuelle Projekt angepaßt werden.

Ergebnisse

Das Ergebnis dieses Aktivitätstyps ist eine Beschreibung des auf das Projekt zugeschnittenen Prozeßmodells.

1.1.4 Arbeitspakete und Ressourcen planen

Zweck

Dieser Aktivitätstyp dient der Definition von Arbeitspaketen. Für die Arbeitspakete müssen der Aufwand geschätzt und die benötigten Ressourcen, wie z.B. Personal oder Arbeitsmittel, festgelegt werden. Zunächst wird eine Grobplanung für das System vorgenommen, in der die Arbeitspakete im Projektteam verteilt werden und die Termine und die benötigten Arbeitsmittel festgelegt werden. Diese Grobplanung wird schrittweise durch Feinplanungen für Komponenten und Klassen detailliert. Die Planung wird durch das von uns vorgeschlagene Aufwandschätzverfahren CEOS unterstützt.

Beteiligte

Ausführend: Projektmanager, Projektleiter
Mitwirkend: Systemanalytiker, Softwaredesigner, Softwareentwickler

Vorgehen

Im Detail sind folgende Schritte zu durchlaufen:

- Arbeitspakete definieren,
- Termine planen,
- Personal planen,
- Arbeitsmittel planen.

Arbeitspakete definieren

Im EOS-Modell sind die Projektmitarbeiter primär für Gegenstände und erst dann für Tätigkeiten zuständig. Das bedeutet, daß die Mitarbeiter Bausteine betreuen. Eine Komponente wird einem Team und eine Klasse einem Mitarbeiter zugeordnet. Zur Definition von Arbeitspaketen wird die grobe Systemarchitektur herangezogen, die in dem Schritt „Ziele definieren und Gegenstandsbereich abgrenzen“ vorgeschlagen wurde. Je nach Detaillierungsgrad der Systemarchitektur werden Arbeitspakete, wie z.B. „System analysieren“, „Komponente X_1 analysieren und entwerfen“ oder „Klasse K_1 analysieren, entwerfen und implementieren“, definiert. Ein Arbeitspaket bezieht sich also auf eine oder mehrere der bausteinbezogenen Tätigkeiten „Analyse“, „Entwurf“, „Implementierung“ und

„operationeller Einsatz“. Nachdem die Arbeitspakete definiert wurden, werden die Aufwände dafür geschätzt und eine personelle Zuordnung vorgenommen. Diese Aspekte werden in den nächsten Schritten näher dargestellt. Zur Synchronisation des Projekts werden Revisionspunkte verwendet (siehe Abschnitt 1.2.1). Für die Dokumentation eines Arbeitspakets kann das folgende Beschreibungsschema genutzt werden.

Arbeitspaket	⇨ Beschreiben Sie hier das Arbeitspaket nach dem Schema: „Baustein [Name] Tätigkeit [analysieren, entwerfen, implementieren, operationelle Einsatz durchführen]“ (z.B. „Klasse K_1 analysieren und entwerfen“). [**Hier** klicken]
Person	⇨ Nennen Sie die verantwortlichen Personen und ihre Rollen. [**Hier** klicken]
Ergebnis	⇨ Welche Ergebnisse werden erwartet? [**Hier** klicken]
Aufwand (Soll)	⇨ Tragen Sie hier den geschätzten Aufwand ein. [**Hier** klicken]
Aufwand (Ist)	⇨ Tragen Sie hier den tatsächlichen Aufwand ein. [**Hier** klicken]
Start- und Endtermin (Soll)	⇨ Legen Sie das geplante Start- und Ende-Datum des Arbeitspakets fest. [**Hier** klicken]
Start- und Endtermin (Ist)	⇨ Legen Sie das tatsächliche Start- und Ende-Datum des Arbeitspakets fest. [**Hier** klicken]
Status	⇨ Beschreiben Sie den aktuellen Status des Arbeitspakets. ☐ in Bearbeitung ☐ geplant ☐ zur QS vorgelegt ☐ akzeptiert
Arbeitsmittel	⇨ Welche Arbeitsmittel, wie z.B. Hard- oder Software, werden wann benutzt? [**Hier** klicken]
Hinweise	⇨ Erkannte Risiken, Probleme oder Zusammenhänge sind hier aufzuführen. [**Hier** klicken]

Abb. 5: Beschreibungsschema für Arbeitspakete

Termine planen

Zur Aufwandsschätzung von EOS-Projekten haben wir das CEOS-Verfahren (Cost estimation for EOS projects) vorgeschlagen, das weiter unten detailliert beschrieben wird. Mit dem CEOS-Verfahren kann für das System und seine einzelnen Komponenten und Klassen der Aufwand geschätzt werden. Es kann der Gesamtaufwand oder der Aufwand für die Tätigkeiten Analyse, Entwurf, Implementierung und operationeller Einsatz, jeweils bezogen auf einen Zyklus i, ermittelt werden (siehe Abb. 15). Ausgehend von der groben Systemarchitektur aus dem Schritt 1.1.1 können erste Schätzungen vorgenommen werden, die im Laufe des Projekts schrittweise verfeinert werden.

Daher ist die Terminplanung von Arbeitspaketen einfach. Denn mit dem CEOS-Verfahren kann für ein Arbeitspaket, wie z.B. „Komponente X_1 analysieren und entwerfen“, der Aufwand geschätzt werden und eine personelle Zuordnung getroffen werden. Hierbei werden innerhalb des Unternehmens diejenigen verfügbaren Mitarbeiter ausgewählt, deren Qualifikationen sich mit der Bearbeitung des vorgesehenen Arbeitspakets decken. Die personelle Besetzung muß für jeden Entwicklungszyklus neu geplant werden.

In der Regel wird eine *zweistufige* Terminplanung vorgenommen. Die erste Stufe umfaßt Termine für externe Revisionspunkte und die zweite für interne. Auf der zweiten Stufe kann schnell und unbürokratisch auf unerwartete Probleme reagiert werden, während die Planung auf der ersten Stufe nur bei erheblichen Abweichungen davon betroffen ist. So wird eine Planungs-Flexibilität nach innen und eine Planungs-Stabilität nach außen erreicht.

Eine einfache und übersichtliche Art, Termine zu erstellen, bieten die Balkendiagramme. Abhängigkeiten können mit den sog. Netzplänen detailliert dargestellt werden. Diese beiden Diagrammtypen werden durch die gängigen Werkzeuge des Projektmanagements unterstützt.

Personal planen

Ziel bei der Personalplanung ist es, den optimalen Personaleinsatz für den gesamten Projektverlauf zu ermitteln und eine entsprechende Rollenzuweisung vorzunehmen. Zu hohe Überlastungen und zu geringe Auslastungen sollten möglichst vermieden werden. Ein Projekt kann scheitern, wenn das notwendige Personal nicht zeitgerecht verfügbar ist. Daher ist neben einem zeitkritischen Pfad auch ein kapazitätskritscher Pfad zu berücksichtigen. Dieser Satz trifft auch für die Planung der Arbeitsmittel, wie z.B. Hard- oder Software, zu. Bei der Personalplanung sollten die Qualifikation und die zeitliche bzw. örtliche Verfügbarkeit der Mitarbeiter berücksichtigt werden. Die Teamzugehörigkeit und die Identifikation mit der zu erledigenden Aufgabe spielen ebenfalls eine wichtige Rolle.

Ausgangspunkt für die Optimierung des Personaleinsatzes sind die Terminanforderungen. Es wird zwischen einer *termintreuen* und *kapazitätstreuen Einsatzplanung* unterschieden. Wenn die Termine vom Auftraggeber vorgegeben werden, muß ermittelt werden, wie viele Personen in welcher zeitlichen Belegung erforderlich sind (*termintreue Einsatzplanung*). Wenn aber das zur Verfügung stehende Personal auf der Auftragnehmerseite feststeht, so muß ermittelt werden, wann das Anwendungssystem bei optimalem Personaleinsatz ausgeliefert werden kann. Die Personalplanung verläuft in folgenden Schritten /Balzert 1998/:

1. Ermitteln des Personalvorrats:
   Der „Personalvorrat“ wird in Mann-Monaten (MM) angegeben. Dieser Wert wird auf der Basis der zur Verfügung stehenden Mitarbeiter ermittelt. Der *Bruttowert* ergibt sich durch die Berücksichtigung von Neueinstellungen, Kündigungen, Verrentungen, Versetzungen, Teilzeitarbeit und Arbeitszeitverkürzungen. Der *Nettowert* wird aus dem Bruttowert errechnet, indem Fehl- und Ausfallzeiten abgezogen werden.

2. Errechnen des Personalbedarfs:
   Zwischen dem Aufwand eines Arbeitspakets und dem zu seiner Ausführung benötigten, Personal wird ein lineares Verhalten angenommen, wobei Grenzfälle beachtet werden müssen. Da der Aufwand eines Arbeitspakets durch das von uns vorgeschlagene Verfahren geschätzt wird, kann auch der Personalbedarf errechnet werden.

3. Vergleich von Bedarf und Vorrat:
   Der ermittelte Bedarf und der vorliegende Personalvorrat werden nach projektorientierten, qualifikationsorientierten und organisationsorientierten Aspekten analysiert, um das Optimierungspotential erfassen zu können.

4. Optimierung der Auslastung:
   Die Auslastung wird optimiert, indem nichtkritische Vorgänge aus Überlastbereichen in Bereiche mit geringer Auslastung verlagert werden. Es wird zwischen der *termin-* und *kapazitätstreuen Bedarfsoptimierung* unterschieden. Beim ersten Typ wird versucht, die einzelnen Arbeitspakete innerhalb ihrer jeweiligen Zeitpuffer so zu organisieren, daß eine möglichst gleichmäßige Auslastung erreicht wird (Abb. 6, links). Bei der kapazitätstreuen Bedarfsoptimierung wird vom Personalvorrat ausgegangen und versucht, durch eine Auslagerung eine möglichst gleichmäßige Auslastung zu erreichen, selbst wenn die vereinbarten Termine verschoben werden müssen (Abb. 6, rechts).

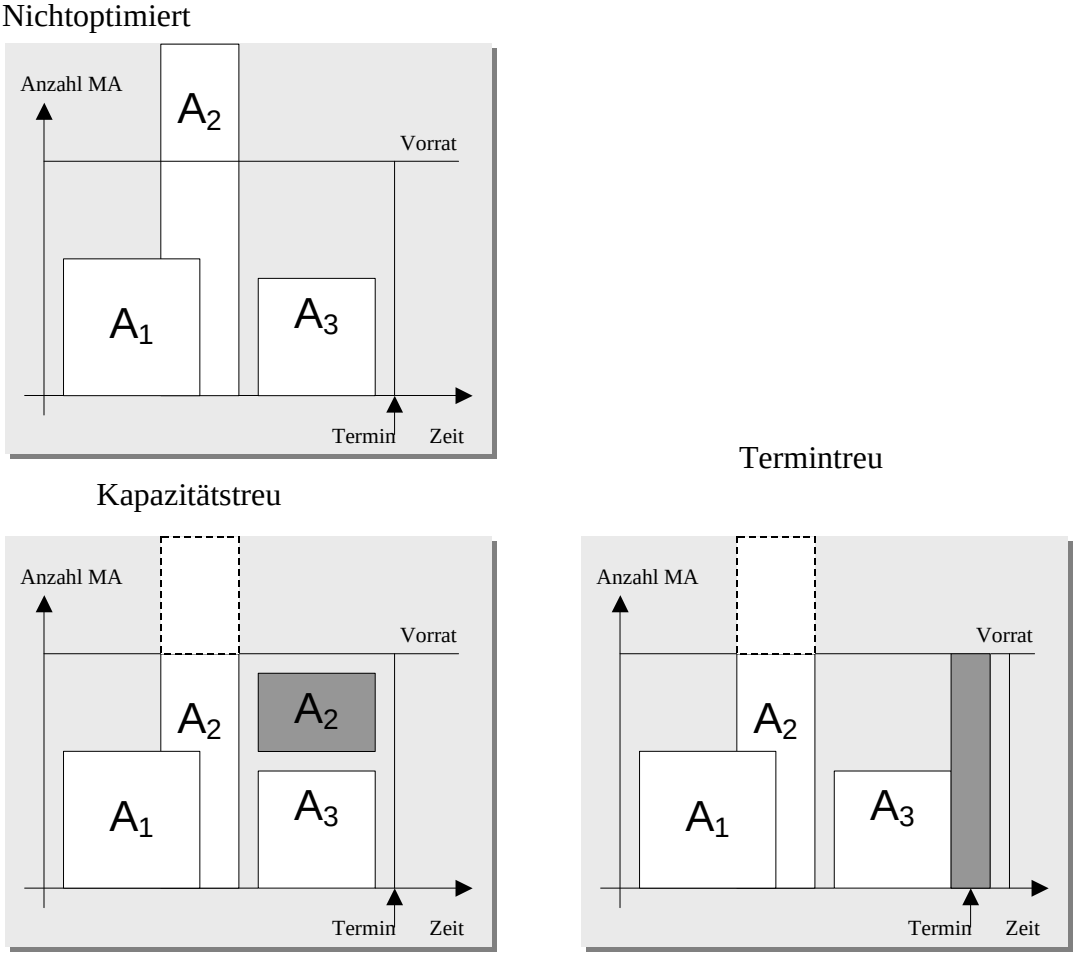


Abb. 6: Termin- und kapazitätstreue Bedarfsoptimierung

Da die heutigen Aufgaben sehr komplex sind und in der Regel nicht durch einen einzelnen Mitarbeiter durchgeführt werden können, müssen Teams gebildet werden. In einem Team arbeiten Mitarbeiter mit unterschiedlichen Qualifikationen, um ein diskretes Ziel zu erreichen. Teams haben eine Reihe von Stärken, wie z.B. einen hohen Problemlösungsgrad oder gegenseitige Anregung und Verstärkung, aber auch Schwächen, wie z.B. den hohen Zeit- und Kommunikationsaufwand. Um die Schwächen zu vermeiden, muß ein Team optimal zusammengestellt und geführt werden. Zur optimalen Zusammenstellung eines Teams sollte ein Projektmanager gute Menschenkenntnisse, Einfühlungsvermögen in Vorlieben und

Abneigungen der Mitarbeiter und ein Gespür für soziale Geflechte haben. Damit er das Team auch optimal führen kann, sollte er imstande sein, Verantwortungsgefühl und Teambewußtsein wecken zu können und das Team zu motivieren wissen.

Arbeitsmittel planen

Der Zweck dieser Tätigkeit ist es, die benötigten Arbeitsmittel, wie z.B. Hard- und Software, zu identifizieren und ihren Einsatz zu planen. Es wird zwischen „verzehrbaren“ und „nicht verzehrbaren“ Arbeitsmitteln unterschieden. „Verzehrbare“ Arbeitsmittel sind solche, die nach dem Gebrauch nicht mehr zur Verfügung stehen, wie z.B. Büromaterial. „Nicht verzehrbare“ Arbeitsmittel, wie z.B. Software, Transportmittel oder Lagerflächen, können dagegen mehrmals verwendet werden. Ein Arbeitsmittelplan wird aufgestellt, wenn Engpässe bei relevanten Arbeitsmitteln eintreten. Wenn aber mehrere Projekte ein Arbeitsmittel, wie z.B. Werkzeuge, teilen müssen, so ist es sehr empfehlenswert, eine Planung vorzunehmen. Die Methoden zur Arbeitsmittelplanung entsprechen weitgehend denen für die Personalplanung. Es werden jedoch drei Vorgehensweisen unterschieden:

- *vorratseingeschränkte Arbeitsmittelplanung:* Hierbei wird von einem beschränkten Vorrat eines Arbeitsmittels ausgegangen. Das Ziel ist es, diesen Vorrat auf mehrere Nutzer möglichst fair aufzuteilen. Dazu wird ein Schichtenplan ausgearbeitet, der festlegt, wer wann und wie lange das Arbeitsmittel nutzen kann. Zum Beispiel wird die Nutzung von Spezialrechnern auf eine Früh- und eine Spätschicht aufgeteilt.
- *bedarfsbezogene Arbeitsmittelplanung:* Hier wird von einem unbegrenzten Vorrat der Arbeitsmittel ausgegangen. Es wird festgestellt, welches Arbeitsmittel zu welchen Zeiten benötigt wird. Damit dafür gesorgt werden kann, daß das benötigte Arbeitsmittel auch zum vorgesehenen Zeitpunkt zur Verfügung steht, muß entsprechend optimiert geplant werden.
- *freie Arbeitsmittelplanung:* Es wird davon ausgegangen, daß keine Gefahr von Engpässen besteht. Daher darf jeder Nutzer die gewünschte Belegung und seinen Bedarf in einem Belegungsplan eintragen.

Ergebnisse

Zur Dokumentation der Ressourcenplanung kann das Beschreibungsschema für Arbeitspakete (Abb. 7) und das folgende Schema genutzt werden.

Gesamtplan für das System im Zyklus i	
Arbeitspakte	⇨ Beschreiben Sie hier die identifizierten Arbeitspakete und verweisen Sie auf deren Dokumentation. [**Hier** klicken]
Terminplan	⇨ Geben Sie in Form eines Balkendiagramms eine Übersicht über die einzuhaltenden Termine im Projekt. Stellen Sie die terminlichen Abhängigkeiten der Arbeitspakete dar. [**Hier** klicken]
Personalplan	⇨ Stellen Sie hier das Projektteam vor. Wie ist das Personal auf die System-Phasen verteilt. Welche Restriktionen gibt es? [**Hier** klicken]
Arbeits-mittelplan	⇨ Beschreiben Sie hier die benötigten Arbeitsmittel und deren Einsatz im Projekt. Für die Planung sind Arbeitsmittel interessant, bei denen Engpässe auftreten können. [**Hier** klicken]

Feinplan für Komponenten und Klassen bezogen auf einem Zyklus		
#		
1	**Baustein**	⇨ Für welchen Baustein ist die Feinplanung? [**Hier** klicken]
	Arbeits-pakte	⇨ Beschreiben Sie hier die identifizierten Arbeitspakete und verweisen Sie auf die Dokumentation der Arbeitspakete. [**Hier** klicken]
	Terminplan	⇨ Stellen Sie die terminlichen Abhängigkeiten der Arbeitspakete dar. [**Hier** klicken]
	Personal-plan	⇨ Wie ist die Personaleinteilung für Analyse, Entwurf, Implementierung und den operationellen Einsatz? Welche Restriktionen gibt es? [**Hier** klicken]
	Arbeits-mittelplan	⇨ Beschreiben Sie hier die benötigten Arbeitsmittel und deren Einsatz für den Baustein. [**Hier** klicken]
2	[**Hier** klicken]	[**Hier** klicken]
Hinweise		⇨ Erkannte Risiken, Probleme oder Zusammenhänge sind hier aufzuführen. [**Hier** klicken]

Abb.7: Beschreibungsschema für die Ressourcenplanung

Hinweise und Tipps

Motivierte Projektmitarbeiter tragen entscheidend zum Erfolg eines Projekts bei. Daher sollten die Projektmitarbeiter stets in die Planung einbezogen werden. Insbesondere sollten die Verantwortlichen bei einem Arbeitspaket stets mit der entsprechenden Arbeitspakets-Planung einverstanden sein. Denn erst wenn ein Projektmitarbeiter eine Verpflichtung freiwillig eingeht, wird sie für ihn bindend. Er wird alles unternehmen, um das „selbstgesteckte“ Ziel zu erreichen. Dieses Aspekt wird als *Management by Commitment* bezeichnet.

1.1.5 Wiederverwendung planen

Zweck

Das Ziel dieses Aktivitätstyps ist es, Vorgaben zu machen, welche vorhandenen Ergebnisse wieder verwendet und welche Produkte für eine Wiederverwendung in nachfolgenden Zyklen oder Projekten entwickelt werden sollen.

Sowohl die Wiederverwendung als auch die Entwicklung zum Zweck der Wiederverwendung kosten Zeit und Geld, so daß sie bei der Projektplanung mit berücksichtigt werden müssen. Wiederverwendung kann zwar Zeit und Geld sparen, ist aber nicht umsonst zu bekommen. Denn wiederverwendete Bausteine müssen gefunden und analysiert werden. Das Entwickeln von wieder zu verwendenden Bausteinen stellt hohe Ansprüche an Entwurf, Dokumentation und Qualitätssicherung. Kurzfristig werden so höhere Kosten verursacht als bei einer Speziallösung für ein Projekt.

Beteiligte

Ausführend: Projektmanager
Mitwirkend: Projektleiter, Systemarchitekt
Beratend: Auftraggeber

Vorgehen

Bei diesem Aktivitätstyp wird vorausgesetzt, daß in der IT-Organisation Wiederverwendung projektübergreifend praktiziert wird und wiederverwendbare Bausteine zentral in einer Bausteinbibliothek verwaltet werden. Es sind folgende Schritte zu durchlaufen:

- Möglichkeiten zur Wiederverwendung analysieren,
- Wiederverwendungsfähige Bausteine ermitteln,
- Vorgaben festschreiben.

Möglichkeiten zur Wiederverwendung analysieren

Sowohl aus projektinterner Sicht als auch projektübergreifend sollte geklärt werden, was wiederverwendet werden kann und was zum Zweck der Wiederverwendung entwickelt werden kann. Diese Frage sollte im Projektverlauf immer wieder gestellt werden.

Wiederverwendungsfähige Bausteine ermitteln

Es ist zu klären, welche der für das geplante Anwendungssystem noch zu entwickelnden Bausteine später wiederverwendet werden können. Liegen mögliche Kandidaten für die Wiederverwendung vor, so ist eine Kosten/Nutzen-Analyse durchzuführen. Überwiegt der Nutzen auf lange Sicht, so sollte man sich für die Wiederverwendung des Bausteins entscheiden. Solche Bausteine sind bei der Planung gesondert zu berücksichtigen, da sie mehr Aufwand und Kosten als die anderen Bausteine verursachen.

Vorgaben festschreiben

Wenn entschieden wurde, daß Bausteine des zu realisierenden Anwendungssystems wiederverwendet werden können, so ist diese Entscheidung zu dokumentieren. Die Entscheidung über gekaufte oder aus der Bausteinbibliothek wiederverwendete Bausteine ist ebenfalls zu dokumentieren.

Ergebnisse

Die Vorgaben zur Wiederverwendung werden in einem Textdokument festgehalten.

Hinweise und Tipps

Für die Einarbeitung von wiederverwendeten Bausteinen sollte genügend Zeit eingeräumt werden. Sonst werden die Projektmitarbeiter versuchen, „es doch lieber selbst und schneller zu machen“.

1.1.6 Konfigurationsmanagement planen

Zweck

Ziel des Konfigurationsmanagements ist es:

- entstehende Bausteine über den gesamten Lebenszyklus zu verfolgen,
- den Projektmitgliedern stets die Rekonstruktion von Entwicklungs- oder Änderungsstand zu ermöglichen,
- nur passende Teile zusammenarbeiten zu lassen.

Zu einem Baustein gehören neben dem produzierten Quellcode auch alle sonstigen Ergebnisse, die im Laufe der Baustein-Entwicklung entstehen. Zum Baustein „System“ gehören z.B. Produkte wie Ressourcenplan, Entwicklungsstrategie oder Komponenten-Diagramme. Diese Produkte, die verschiedene Versionen haben können, werden zu einer Konfiguration zusammengefaßt.
In großen Projekten mit mehreren tausend Bausteinen stellt das Konfigurationsmanagement eine logistische Aufgabe dar, die ohne Werkzeugunterstützung nicht zu bewältigen ist. Daher werden im Rahmen des Aktivitätstyps „Konfigurationsmanagement planen“ organisatorische Aspekte, wie z.B. die Regelung von Namenskonventionen, behandelt. Detaillierte Ausführungen werden bei der Beschreibung des Subprozesses „Konfigurationsmanagement“ gegeben.

Beteiligte

Ausführend: Projektmanager
Mitwirkend: Projektleiter, Projektbibliothekar

Vorgehen

Es sind folgende Schritte zu durchlaufen:

- Organisatorische Regelungen treffen,
- Projektspezifische Regelungen treffen,
- Bausteinbibliothek planen.

Organisatorische Regelungen treffen

Die Planung, Steuerung und Kontrolle des Konfigurationsmanagements liegt beim Projektmanager. Für administrative Aufgaben, wie z.B. das Einrichten der Bausteinbibliothek, ist der Projektbibliothekar zuständig. Sollte keine organisationsweite Bausteinbibliothek vorliegen, so ist dafür zu sorgen, daß eine solche Bibliothek angeschafft und eingerichtet wird.

Projektspezifische Regelungen treffen

Für die projektspezifischen Regelungen sollten die folgenden Aspekte betrachtet und die entsprechenden Entscheidungen dokumentiert werden:

- Welche der Produkte, die im Rahmen der Entwicklung eines Bausteines entstehen, sollten Gegenstand des Konfigurationsmanagements sein?
- Wie wird mit gekaufter Software verfahren?
- Sollten die Werkzeuge, wie Compiler oder Editoren, in das Konfigurationsmanagement einbezogen werden?
- Welche Zustände der Produkte sollten unterschieden werden? Die nächste Abbildung zeigt stark vereinfachend mögliche Zustände bzw. Zustandsübergänge.

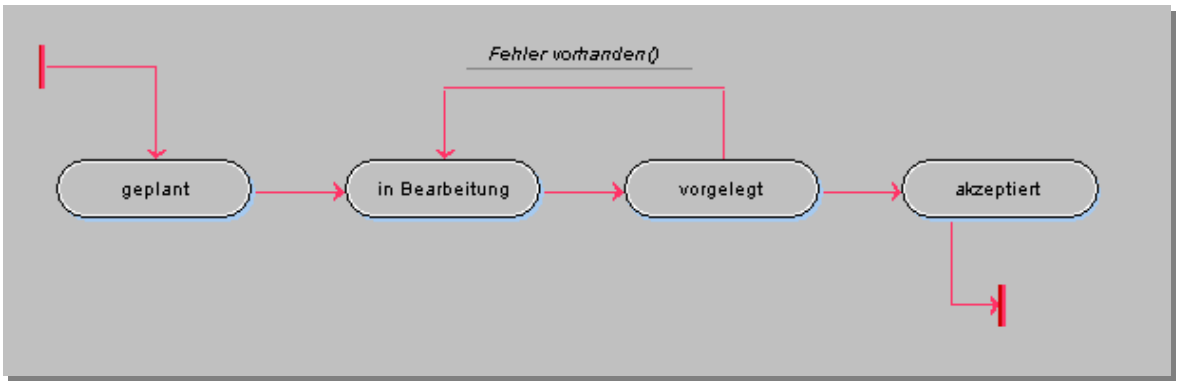


Abb. 8: Mögliche Zustände und Zustandsübergänge für Produkttypen

Bausteinbibliothek planen

In diesem Schritt wird der Einsatz und die technische Realisierung der Bausteinbibliothek geplant. Hierzu muß über die einzusetzende Technik und die Grobstruktur der Bausteinbibliothek entschieden werden. Außerdem muß ein Rechtekonzept ausgearbeitet werden. Folgende Fragen sollten geklärt werden:

- Wer hat welche Rechte auf die Bausteinbibliothek?
- Wer kontrolliert und vergibt die Zugriffsrechte?
- Welche Namenskonventionen sollten eingehalten werden?
- Wie sind die abgelegten Elemente zu identifizieren?
- Wie werden Änderungsaufträge behandelt? Die nächste Abbildung zeigt vereinfachend eine Möglichkeit, wie mit Änderungsaufträgen verfahren werden könnte:

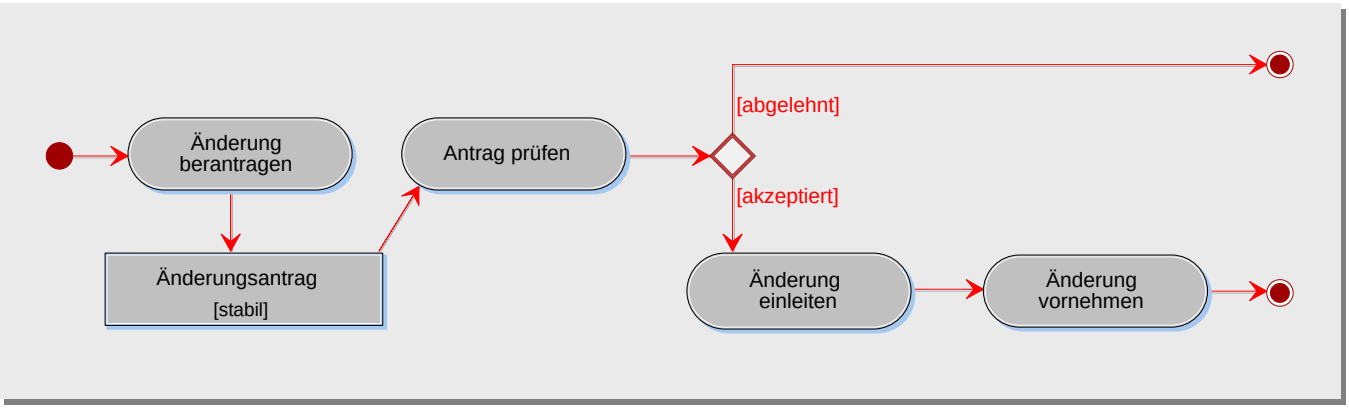


Abb. 9: Änderungsverfahren

Ergebnisse

Das Ergebnis dieses Aktivitätstyps ist ein Konfigurationsmanagementplan, der wie folgt gegliedert werden kann:

1. **Organisatorische Regelungen**

⇨ Notieren Sie hier organisatorische Regelungen, wie z.B. das Vorliegen bzw. Beschaffung eines KM-Systems.

[**Hier** klicken]

2. **Projektspezifische Regelungen**

⇨ Notieren Sie hier projektspezifische Aspekte, wie z.B. das Einbeziehen von Entwicklungswerkzeugen durch das Konfigurationsmanagement.

[**Hier** klicken]

3. **Regelungen zur Bausteinbibliothek**

⇨ Beschreiben Sie hier die, auf die Bausteinbibliothek bezogenen, Regelungen, wie z.B. Namenskonventionen oder Rechtevergabe.

[**Hier** klicken]

4. **Regelungen zum Änderungsverfahren**

⇨ Beschreiben Sie, wie mit Änderungen verfahren werden soll.

[**Hier** klicken]

Abb.10: Beschreibungsschema für den Konfigurationsmanagement-Plan

1.1.7 Qualitätssicherung planen

Zweck

Das Ziel dieses Aktivitätstyps ist es, projektbezogene Regelungen für die Qualitätssicherung zu treffen. Es wird festgelegt, welche Produkttypen einer Qualitätssicherung unterzogen werden müssen (*Was wird Qualitäts-gesichert?*). Die Qualitätsziele, -Merkmale und -Vorgaben werden präzisiert und gewichtet. Außerdem werden die anzuwendenden Verfahren zur Qualitätssicherung bestimmt (*Wie wird gesichert?*). Hierbei wird zwischen *konstruktiver* und *analytischer* Qualitätssicherung unterschieden:

- Konstruktive Qualitätssicherung umfaßt alle Maßnahmen, die schon während der Entwicklung zur Qualität beitragen.
- Analytische Qualitätssicherung umfaßt Maßnahmen, die nach der Entwicklung eines Produkts zu Bewertung seiner Qualität herangezogen werden.

Die geplanten Maßnahmen müssen in den Entwicklungsprozeß eingegliedert werden (*Wann wird gesichert?*). Analytische Maßnahmen können z.B. bei Erreichen eines Revisionspunktes (siehe Projekt-Steuerung) oder nach Abschluß einzelner Schritte eines Baustein-Zyklus ausgeführt werden. Zuletzt muß noch definiert werden, wer für die Durchführung der Maßnahmen verantwortlich ist (*Wer sichert?*).

Beteiligte

Ausführend: Projektmanager
Mitwirkend: Projektleiter, Qualitätsprüfer

Vorgehen

Es wird hier vorausgesetzt, daß auf ein unternehmensweites Qualitätsmodell zurückgegriffen wird. Folgende Schritte sind zu durchlaufen:

- Konstruktive Qualitätssicherungs-Maßnahmen planen,
- Produkt- und Aktivitätstypen für die Qualitätssicherung festlegen,
- Qualitätsziele dokumentieren,
- Analytische Qualitätssicherungs-Maßnahmen planen,
- Spezifische Prüfungen planen.

Konstruktive Qualitätssicherungs-Maßnahmen planen

Da Qualität nicht allein durch nachträgliche Tests und Verbesserungen in ein Produkt „hineingeprüft“ werden kann, müssen von Anfang an konstruktive Maßnahmen definiert werden. Die systematische Anwendung von Methoden, Werkzeugen, Richtlinien und Standards kann z.B. dafür sorgen, daß das entstehende Produkt a priori eine hohe Qualität aufweist.

Produkt- und Aktivitätstypen für die Qualitätssicherung festlegen

Es ist festzulegen, welche Produkttypen qualitätsgesichert werden sollen. Qualität entsteht aber auch durch die Art, wie Produkte erzeugt werden. Daher können auch Aktivitätstypen Gegenstand der Qualitätssicherung sein.

Qualitätsziele dokumentieren

Der Qualitätsmodell legt fest, welche Qualitätsmerkmale gemessen werden sollen und wie sie quantifiziert werden können. Es soll nun festgelegt werden, „wieviel“ Qualität pro Produkt erreicht werden soll. Dazu ist eine Skala, von z.B. unannehmbar bis sehr gut, zu definieren. Für jedes Produkt muß ein Wert innerhalb der Skala festgelegt werden. Dieser Wert ist für das Produkt durch entsprechende Maßnahmen zu erreichen.

Analytische Qualitätssicherungs-Maßnahmen planen

Die vorgesehenen analytischen Maßnahmen, wie z.B. Reviews, Inspektionen, Verifikation oder Tests, sind zu dokumentieren. Für die einzelnen Maßnahmen sind die Ausführungszeitpunkte festzulegen.

Spezifische Prüfungen planen

Wenn Software-Komponenten gekauft werden, so müssen diese gegebenenfalls qualitätsgesichert werden. Das gleiche gilt auch für wiederverwendete Bausteine.

Ergebnisse

Das Ergebnis dieses Aktivitätstyps ist ein Qualitätssicherungs-Plan. Für die Gliederung dieses Dokuments sind eine Reihe von Normen, wie z.B.IEEE-Standard 730/84 oder ISO 9000, definiert worden, die im Kern der folgenden Gliederung entsprechen.

1 **Qualitätsziele**

⇨ Definieren Sie hier eine Skala, wie z.B. unannehmbar bis sehr gut, für die zu erreichende Qualität.

[**Hier** klicken]

1.1 **Qualitätsziele der Produkttypen**

⇨ Definieren Sie für die Produkttypen, die qualitätsgesichert werden sollen, die Qualitätsziele mit der oben definierten Skala.

[**Hier** klicken]

1.2 **Qualitätsziele der Aktivitätstypen**

⇨ Definieren Sie für die Aktivitätstypen, die qualitätsgesichert werden sollen, die Qualitätsziele mit der oben definierten Skala.

[**Hier** klicken]

2 **Qualitätssicherungs-Maßnahmen**

⇨ Hier können allgemeine Hinweise zu den Qualitätssicherungs-Maßnahmen angegeben werden.

[**Hier** klicken]

2.1 **Konstruktive Qualitätssicherungs-Maßnahmen**

⇨ Nennen Sie die geplanten konstruktiven Maßnahmen zur Qualitätssicherung. Wer wird wann welches Prüfverfahren durchführen?

[**Hier** klicken]

2.2 **Analytische Qualitätssicherungs-Maßnahmen**

⇨ Nennen Sie die geplanten analytischen Maßnahmen zur Qualitätssicherung. Wer wird wann welches Prüfverfahren durchführen?

[**Hier** klicken]

3 **Spezifische Prüfungen**

⇨ Werden gegebenenfalls wiederverwendete Bausteine oder gekaufte Software qualitätsgesichert?

[**Hier** klicken]

Abb. 11: Beschreibungsschema für den Qualitätssicherungs-Plan

Hinweise und Tipps

Für die Planung von Qualitätssicherungs-Maßnahmen können auch Qualitätsmuster verwendet werden /Gilb 1988/. Die nächste Abbildung zeigt ein Beispiel hierzu.

Bezeichnung und Beschreibung:	Durchschnittliche Antwortzeit des Systems bei normalen Abfragen
Voraussetzungen:	Es sind ein Testrahmen und Testdaten vorhanden.
Zeitpunkt der Prüfung:	Während des Alpha-Tests
Prüfung:	Die fünf häufigsten Abfragen sind jeweils 20 mal durchzuführen. Aus den gemessenen Antwortzeiten ist die durchschnittliche Antwortzeit pro Abfrage zu berechnen.
Maßeinheit/Skala:	Antwortzeit in Sekunden
Schlechtester, gerade noch zu akzeptierender Wert:	30 Sekunden
Geplanter Wert:	20 Sekunden
Bester Wert:	10 Sekunden
Gegenwärtiger Vergleichswert:	40 Sekunden
Referenzen:	Zur Beschreibung der fünf Abfragen vgl. Dokumentation des User-Interface-Modells.
Ursprung	Bereich Marketing

Abb. 12: Qualitätsmuster für die Planung von Qualitätssicherungs-Maßnahmen mit Qualitätsmerkmal „Effizienz“ und Qualitätsindikator Antwortzeit /Goldberg, Rubin 1995/.

1.1.8 Nutzung und Bewertung planen

Zweck

Im Rahmen des Subprozesses „Nutzung und Bewertung“ wird der Nutzer von Anfang bis zum Ende des Projekts begleitet und betreut. Beispielsweise werden bei der Software-Entwicklung die Anwender von der Anforderungsanalyse bis hin zur Abwicklung von Schulungen stets einbezogen. Für den Erfolg des Projekts sind u.a. regelmäßige Workshops und Sitzungen mit dem Auftraggeber bzw. Anwender entscheidend. Solche Projektsitzungen müssen geplant werden, was das Ziel dieses Aktivitätstyps ist. Detaillierte Ausführungen werden bei der Beschreibung des Subprozesses „Nutzung und Bewertung“ gegeben.

Beteiligte

Ausführend: Projektmanager
Mitwirkend: Projektleiter

Vorgehen

Folgende Schritte sind zu durchlaufen:

- Projektsitzung vorbereiten,
- Projektsitzung durchführen.

Projektsitzung vorbereiten

Damit eine Projektsitzung ihren Zweck erfüllt, muß Klarheit über den Anlaß geschaffen werden. Ist vielleicht ein vereinbarter Revisionspunkt erreicht? Erfordert ein kritisches Ereignis eine Entscheidung? Müssen wichtige Informationen weitergegeben werden? Ist man sich über den Anlaß im Klaren, so müssen die Ziele formuliert und die Tagesordnungspunkte geplant werden. Zur Planung der Tagesordnungspunkte gehört ein Zeitplan für die einzelnen zu besprechenden Themen. Als nächstes sollte eine Einladung an die Teilnehmer verschickt werden. Die Einladung sollte Informationen, wie Ziele, Themen, Zeiten oder Vorabinformationen zur Vorbereitung der Teilnehmer enthalten. Zuletzt sollte eventuelles Präsentationsmaterial vorbereitet werden und Arbeitsmedien, wie z.B. Flipchart oder Projektor, reserviert werden.

Projektsitzung durchführen

Zu Beginn der Projektsitzung sollte die Tagesordnung besprochen werden, ein Protokollführer ernannt werden und die Rollen der Teilnehmer geklärt werden. Es ist darauf zu achten, daß am Thema diskutiert wird und Mißverständnisse soweit wie möglich beseitigt werden.

Ergebnisse

Die Ergebnisse der Projektsitzung sollten in Form eines Protokolls festgehalten werden.

Hinweise und Tipps

Folgende Punkte sollten berücksichtigt werden:

- dafür sorgen, daß Leute, die einen Beitrag leisten können/wollen, auch Gelegenheit bekommen,
- bei Bedarf Regeln formulieren und zur Diskussion stellen,
- dominierende Leute bremsen,
- Konflikte offenlegen, wenn sie den Fortgang der Sitzung stören.

1.2 Projekt-Steuerung

Das primäre Ziel des Projektmanagements ist es, das Projekt erfolgreich durchzuführen. Das bedeutet, die Erwartungen des Auftraggebers zu erfüllen und das gewünschte Produkt kosten- und zeitgerecht zu liefern. Zu diesem Zweck muß das Projekt kontinuierlich begleitet und der Entwicklungsstand mit Hilfe der Revisionspunkte synchronisiert werden. Risiken müssen rechtzeitig identifiziert und Änderungswünsche systematisch koordiniert werden. Abbildung 13 faßt die von uns definierten Aktivitätstypen für die Projekt-Steuerung zusammen.

Die Grundlage für die Projekt-Steuerung bildet die Projekt-Initialisierung und Planung. Bausteine sind die Hauptgegenstände der Projekt-Steuerung. Ihre fortschreitende Entwicklung gilt es ständig zu begleiten und zu koordinieren. Bei jeder der vier Phasen wird die Planung anhand der beobachteten Bausteinzustände revidiert. Die Planung in den verschiedenen Detaillierungsstufen System, Komponente und Klasse muß an veränderte Rahmenbedingungen angepaßt werden. Zur Synchronisation und Steuerung von Zyklen und Bausteinen werden weitere Revisionspunkte geschätzt bzw. bestehende verfeinert.

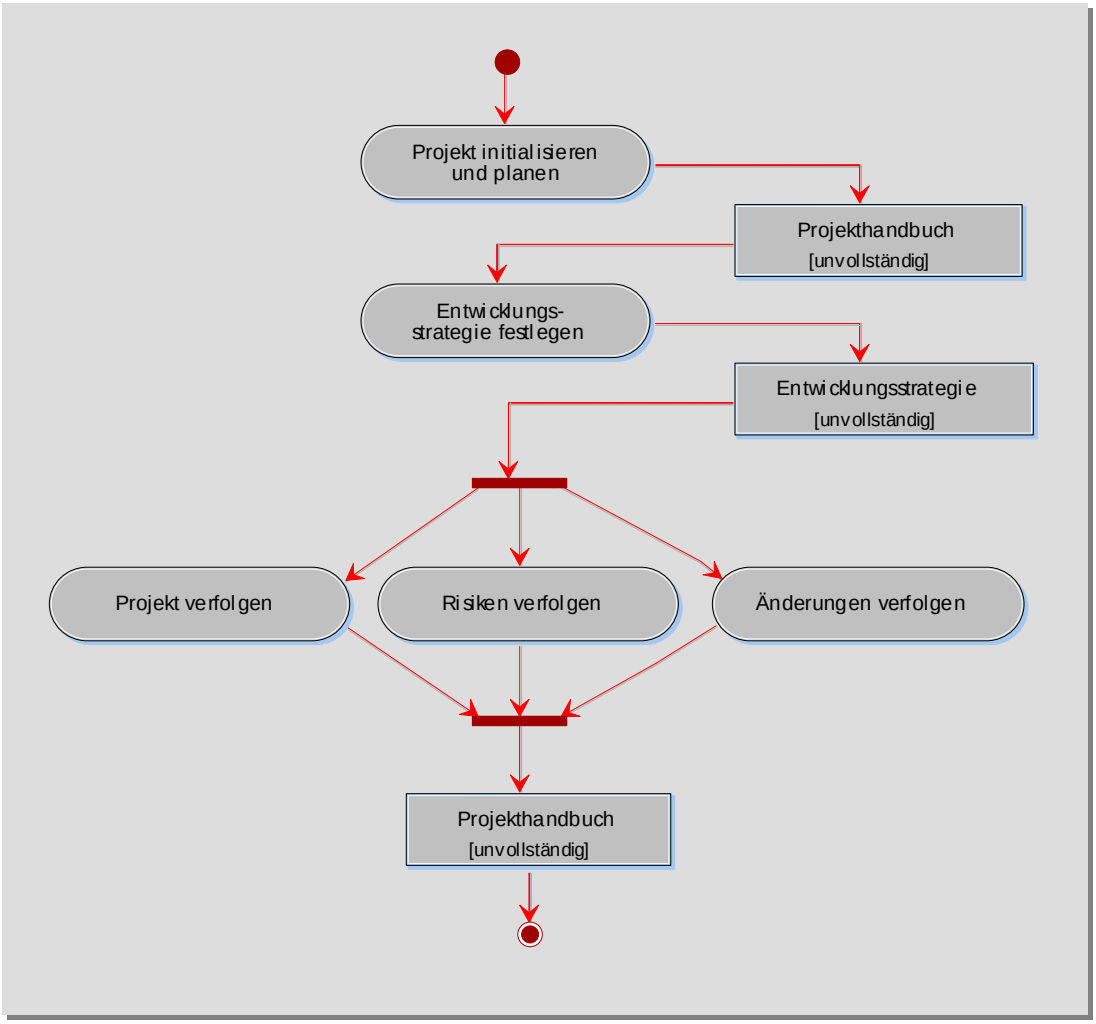


Abb. 13: Aktivitätstypen der Projekt-Steuerung

1.2.1 Entwicklungsstrategie festlegen

Zweck

Dieser Aktivitätstyp dient zur Identifikation und Definition von EOS-Elementen, wie z.B. ersten Bausteinen, Zyklen oder Revisionspunkten. Die im Rahmen der vorherigen Schritten (1.1 Projekt-Initialisierung und Planung) vorgeschlagene Groblösung und ihre entsprechende Systemarchitektur werden vorausgesetzt.

Beteiligte

Ausführend: Projektmanager
Mitwirkend: Projektleiter

Vorgehen

Im Detail sind folgende Schritte durchzuführen:

- Bausteine identifizieren,
- Zyklen planen,
- Revisionspunkte definieren.

Bausteine identifizieren

Der Entwicklungszyklus auf der Systemebene ist bereits angestoßen, so daß zumindest die Kernanforderungen beschrieben worden sind (z.B. durch Anwendungsfälle oder Tabellen). Sicherlich sind auch schon Bausteine aus den funktionalen bzw. nicht-funktionalen Anforderungen abgeleitet worden. Die Anforderungen dienen hierbei als ein möglicher Ausgangspunkt zur Identifikation von Bausteinen (keine 1:1 Beziehung zwischen Anforderungen und Bausteine). Mit der Unterstützung der Projektleitung wird versucht, zusätzliche Bausteine zu den bereits gefundenen zu identifizieren, z.B. durch Auf- oder Abspaltung von Bausteinen oder durch das Erkennen wiederverwendbarer Bausteine. Die gefundenen Bausteine werden genutzt, um die Grobarchitektur des Anwendungssystems weiter zu erarbeiten und zu überdenken.

Die grundsätzlichen Überlegungen oder Vorgaben aus *1.1 Projekt-Initialisierung und Planung* („Ziele definieren und Gegenstandsbereich abgrenzen“) zur angestrebten Lösung und Systemarchitektur sind mit zu berücksichtigen.

Zyklen planen

Nach der Identifikation von Bausteinen sind Entwicklungszyklen zu planen. Wir betrachten hierzu die funktionalen bzw. nicht-funktionalen Anforderungen, die durch Bausteine realisiert werden. Je höher die Priorität der Anforderung, desto höher ist auch die Priorität der Bausteine, die die Anforderung realisieren. Aus den Prioritäten der Bausteine und ihren gegenseitigen Abhängigkeiten ist eine Realisierungsreihenfolge für die Bausteine zu entwickeln. Aus der Realisierungsreihenfolge sind Entwicklungszyklen abzuleiten, die jeweils eine Ausbaustufe des Bausteins darstellen. Bei der Planung von Entwicklungszyklen sind

stets die Projektgröße, die Komplexität der Projektaufgaben, die Stabilität der vorliegenden Anforderungen und die verfügbaren Ressourcen mit zu berücksichtigen

Die zu diesem Zeitpunkt identifizierten Komponenten müssen nicht in einem Komponenten-Zyklus realisiert werden. Entscheidend ist wieder die Priorität der ihnen zugeordneten Anforderungen. Anforderungen mit hoher Priorität sind in den ersten Komponenten-Zyklen zu realisieren und solche mit niedriger Priorität später. Damit ist auch die Realisierungsreihenfolge der Klassen einer Komponente - für den Fall, daß die Klassen schon bekannt sind - vorgegeben. Die Klassen, die eine Anforderung mit hoher Priorität abbilden, sind zuerst zu realisieren. So kann entschieden werden, welche Klassen in welchem Komponenten-Zyklus realisiert werden müssen.

Sollten Klassen identifiziert worden sein, so sind Klassen-Zyklen in Abhängigkeit von den Diensten zu planen, die die Klasse zur Verfügung stellen muß. Zum Beispiel werden in einem ersten Klassen-Zyklus nur die notwendigsten Methoden implementiert. In einem zweiten Klassen-Zyklus werden Verfeinerungen, wie z.B. Sortierung, realisiert.

Es sei noch mal betont, daß die Planung von Zyklen stets den Bausteinen angepaßt werden muß. Am Anfang wird man sicherlich nur die Zyklen für das System und möglicherweise für einige wenige Komponenten planen können. Im späteren Projektverlauf wird die Planung von Klassen-Zyklen hinzukommen.

Revisionspunkte definieren

Zur Projektverfolgung werden zu Projektbeginn Revisionspunkte vorgesehen, die inhaltlich erst mit fortschreitender Detailplanung festgelegt werden. Revisionspunkte werden auf der Basis von Bausteinen, Zyklen und Tätigkeiten definiert. Die nächste Abbildung zeigt einen möglichen Revisionspunkt für ein Projekt im fortgeschrittenen Entwicklungsstand.

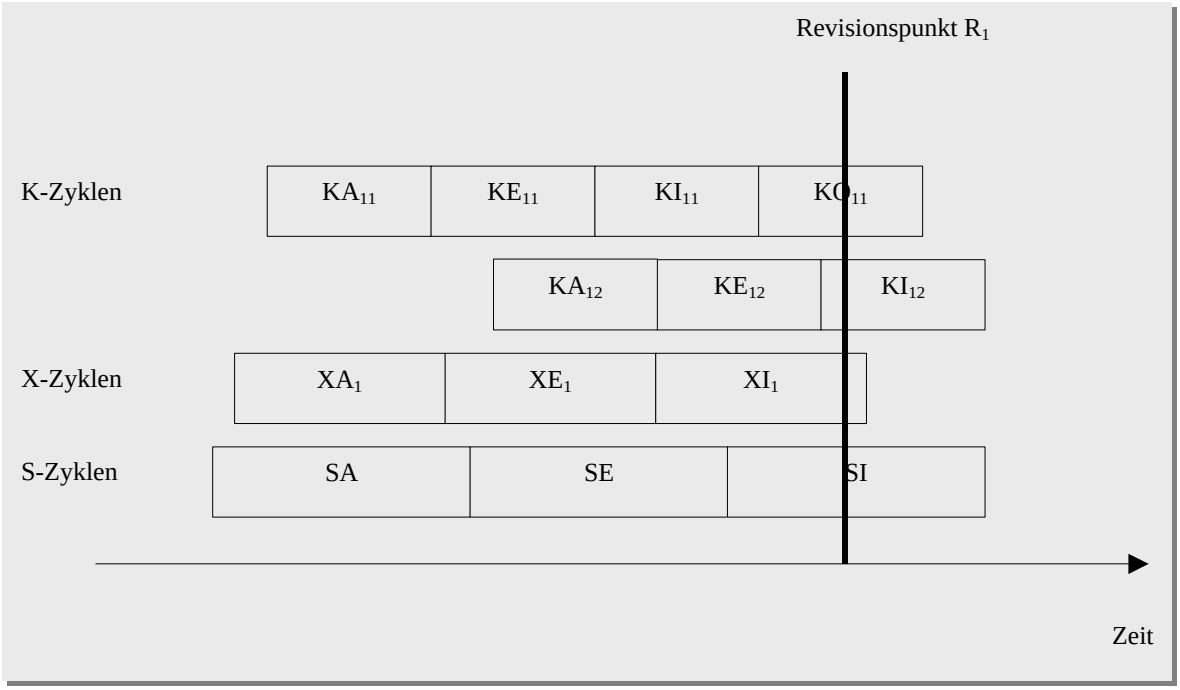


Abb. 14: Beispiel für einen Revisionspunkt

In der obigen Abbildung ist der Revisionspunkt R_1 erreicht, wenn die Klasse A_{11} implementiert ist, Klasse A_{12} entworfen ist, die Komponente X_1 und das System entworfen sind.

Ein Revisionspunkt ist mit einem Termin und einem inhaltlichen Ergebnis verknüpft. Die Ergebnisse sind durch die Phasen „Analyse“, „Entwurf“, „Implementierung“ und „operationeller Einsatz“, die an jedem Baustein durchgeführt werden, inhaltlich definiert. Wie ist es aber mit dem terminlichen Aspekt? Wann ist ein Revisionspunkt erreicht? Um diese Fragen im voraus beantworten zu können, muß man den Entwicklungsaufwand für Bausteine und ihre Zyklen abschätzen können. Hierzu kann wieder das CEOS-Verfahren zur Schätzung des Entwicklungsaufwands genutzt werden. Die Grundidee des CEOS-Verfahrens ist es, für Bausteine Komplexitätswerte zu berechnen, aus denen mit Hilfe statistischer Techniken der Aufwand geschätzt wird. Für einen beliebigen Baustein kann der Aufwand für die einzelnen Phasen und Zyklen geschätzt werden. Wie das Verfahren im Detail funktioniert, wurde im Rahmen des Prototypen erklärt. Für die Planung der Revisionspunkte ist das Ergebnis des Schätzverfahrens interessant, das in der nächsten Abbildung dargestellt wird.

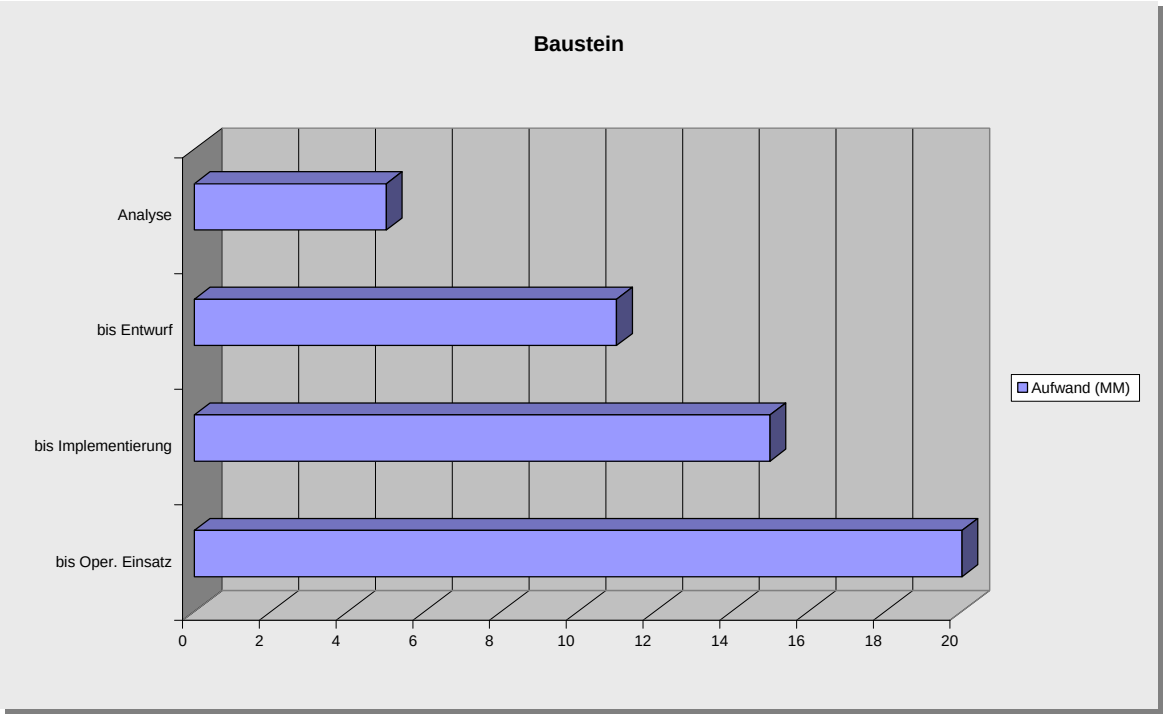


Abb. 15: Aufwand eines Bausteins für einen Entwicklungszyklus

Da für einen beliebigen Baustein der Aufwand bis zu einem definierten Zeitpunkt, wie z.B. „bis einschließlich Entwurf“, geschätzt werden kann, können Revisionspunkte, wie z.B. in der Abbildung 14, leicht geplant werden. Wie Revisionspunkte softwaregestützt definiert werden können, wurde ebenfalls im Rahmen der Darstellung von PMS gezeigt.

In Abhängigkeit mit den Entscheidungsträgern werden zwei Arten von Revisionspunkten unterschieden:

- *externe Revisionspunkte:* Zeitpunkt zur Prüfung von Projektergebnissen durch den Auftraggeber.

- *interne Revisionspunkte:* Zeitpunkt zur Prüfung von Projektergebnissen durch interne Projektmitglieder, ohne Beteiligung des Auftraggebers.

Beide Typen von Revisionspunkten können auf der Grundlage der Aufwandsschätzung geplant werden. In der Regel wird man den Zeitpunkt für einen internen Revisionspunkt vor einem externen definieren.

Ergebnisse

Das Ergebnis dieses Aktivitätstyps ist ein Textdokument, das wie folgt gegliedert werden kann:

Entwicklungs-strategie	⇨ Hier können allgemeine Bemerkungen, wie z.B. Lösungs- oder Architektur-Vorgaben, hinterlegt werden. [**Hier** klicken]
Bausteine	⇨ Beschreiben Sie welche Bausteine identifiziert wurden. Welche Anforderungen werden mit den jeweiligen Bausteinen realisiert? Welche Priorität haben die Anforderungen und die Bausteine? Wie sieht die Realisierungsreihenfolge aus? [**Hier** klicken][**Hier** klicken]
Zyklen	⇨ Beschreiben Sie hier die vorgesehenen Zyklen für die einzelnen Bausteine. Wie viele Zyklen sind geplant? Was wird pro Zyklus realisiert? [**Hier** klicken]
Revisionspunkte	⇨ Beschreiben Sie hier die vorgesehenen Revisionspunkten. Wann ist der Revisionspunkt erreicht? Was muß bis dahin vorliegen? Handelt es sich um einen internen oder externen Revisionspunkt. [**Hier** klicken]
Hinweise	⇨ Erkannte Risiken, Probleme oder Zusammenhänge sind hier aufzuführen. [**Hier** klicken]

Abb. 16: Dokumentation der Entwicklungsstrategie

Hinweise und Tipps

Beim Planen von Zyklen müssen die zu erreichenden Ziele vereinbart werden. Hierbei können folgende Fragen nützlich sein:

- Was soll am Ende des Zyklus erreicht werden?
- Wie können die Ergebnisse am Ende geprüft werden?
- Welche Prioritäten sind zu beachten?
- Sind die Ziele auch realistisch?

Für Unternehmen, die über wenig Erfahrung mit objektorientierten Techniken verfügen, ist das EOS-Modell sehr zu empfehlen. Denn nach den ersten Bausteinen und Entwicklungszyklen hat das Projektteam schon viel an Wissen und Erfahrung gesammelt, so daß für die nächsten Bausteine schnellere und bessere Ergebnisse erwartet werden können.

1.2.2 Projekt verfolgen

Zweck

Ziel dieses Aktivitätstyps ist es, das Projekt kontinuierlich zu beobachten, zu geeigneten Zeitpunkten - nämlich beim Erreichen eines Revisionspunktes - zu bewerten und gegebenenfalls notwendige Korrektur-Maßnahmen zu ergreifen. Zum Beispiel muß die Qualitätssicherung, das Konfigurationsmanagement oder das Wiederverwendungsmanagement verfolgt werden.

Beteiligte

Ausführend: Projektmanager
Mitwirkend: Projektleiter

Vorgehen

Folgende Schritte sind zu durchlaufen /Hesse 1999/:

- Projekt verfolgen,
- Projekt bewerten,
- Plan/Ist vergleichen,
- Maßnahmen ergreifen.

Projekt verfolgen

Voraussetzung für die Projekt-Verfolgung ist eine gut funktionierende Kommunikation zwischen den Projektmitgliedern. Die Unternehmenskultur kann viel dazu beitragen, daß Mitarbeiter offen miteinander kommunizieren. Um das Projekt verfolgen zu können, müssen regelmäßige Projekttreffen stattfinden, elektronische Kommunikationsmedien genutzt werden, gegebenenfalls die Freigabemechanismen der Bausteinbibliothek herangezogen werden aber vor allem die definierten Revisionspunkte genutzt werden.

Anhand der Bausteine und der entsprechenden Zyklen und Tätigkeiten kann der Projektmanager das gesamte Projekt systematisch verfolgen. Hierzu sollten folgende Fragen geklärt werden:

- Welche Tätigkeit (Analyse, Entwurf, Implementierung oder operationeller Einsatz) wird zur Zeit an einem Baustein ausgeführt?
- In welchem Entwicklungszyklus befindet sich der Baustein?
- Gibt es Bausteine, bei denen es Schwierigkeiten oder unvorhergesehene Risiken gibt?

Die Kommunikation mit dem Auftraggeber obliegt in der Regel dem Projektmanager. Er muß für eine ständige Kommunikation sorgen, um Probleme und Mißstände rechtzeitig erkennen zu können.

Projekt bewerten

Für die Bewertung des Projektsstands werden zuverlässige Angaben über geleistete Tätigkeiten, abgeschlossene Entwicklungsphasen, fertiggestellte Bausteine oder Ergebnisse von Reviews, Inspektionen und Revisionspunkt-Prüfungen benötigt. Dabei reicht die formale Vollzugsmeldung nicht aus. Die geforderte Qualität muß erbracht werden.

Für die Entwicklungszyklen der Bausteine sollten folgende Fragen beantwortet werden:

- Wurden die gesetzten Ziele durch den Zyklus erreicht?
- Wie gut (Qualität z. B. im Bezug auf Vollständigkeit und Korrektheit) wurden die gesetzten Ziele erreicht?
- Welche Mängel liegen vor?
- Welche Ursachen und Gründe gibt es für die Mängel?

Für den folgenden Vergleich ist es notwendig, die entstandenen Zeit- und Budget-Kosten zu ermitteln. Dazu sollte für jeden Baustein die bis zum jetzigen Zeitpunkt entstandenen Zeit- und Budget-Kosten berechnet und summiert werden.

Plan/Ist vergleichen

Um entscheiden zu können, ob Maßnahmen zur Projektsteuerung notwendig sind, sollte der gegenwärtige Projektstand mit der Planung verglichen werden. In der Regel werden Termine und Kosten der Bausteine verglichen. Übersteigen die Kosten oder Termine die geplanten Werte, so sind entsprechende korrigierende Maßnahmen zu veranlassen. Der Ist-/Soll-Stand von Terminen bzw. Kosten kann mit Hilfe von Balkendiagrammen bzw. Kosten-Kurven dargestellt werden. Die nächste Abbildung zeigt hierzu Beispiele.

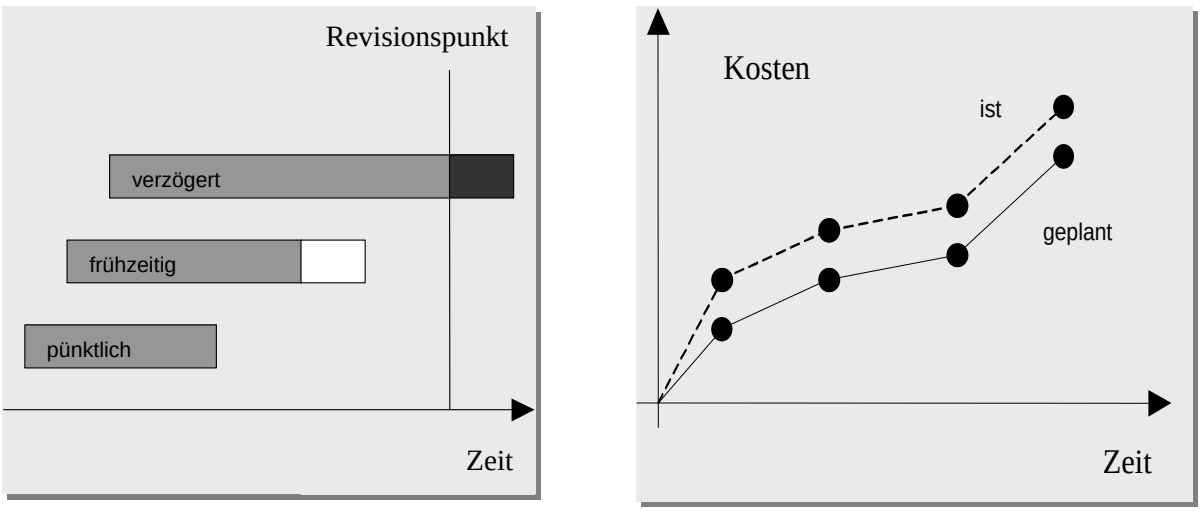


Abb. 17: Graphische Darstellung von Kosten und Termin Plan-/Ist-Vergleich

Maßnahmen ergreifen

Termin- und Kosten-Überschreitungen sind die Hauptursachen für korrigierende Maßnahmen. Es lassen sich zwei Arten von korrigierenden Maßnahmen unterscheiden:

- *Unmittelbare Maßnahmen:* Das sind solche Maßnahmen, die kurzfristig angelegt sind und zur Schadensbehebung bzw. –begrenzung dienen. Sie setzen an den Symptomen, wie z.B.

Termin- oder Kosten-Überschreitung an. Bei Termin-Überscheitung kann z.B. der Lieferumfang reduziert, der Liefertermin hinausgezögert oder die Mitarbeiterzahl kontrolliert erhöht werden.

- *Vorbeugende Maßnahmen:* Diese sind mittel- bis langfristig angelegt und dienen der Vermeidung von Schäden. Sie setzen an den Ursachen an, wie z.B. unrealistische Planungen, ungenügende Qualifikation des Personals oder nachträglich geänderte Anforderungen. Das Bekämpfen von Ursachen erfordert in der Regel schwierige und möglicherweise unangenehme Entscheidungen, wie z.B. Personalwechsel oder Revision der Planungen.

Ergebnisse

Der Projektstatus und mögliche korrigierende Maßnahmen sollten protokolliert werden. Termin- und Personalplanungen sollten gegebenenfalls aktualisiert werden.

Hinweise und Tipps

Korrigierende Maßnahmen sollten nicht hinausgezögert werden, da die erwartenden Probleme sich in der Regel nicht von selbst lösen.

1.2.3 Risiken verfolgen

Zweck

Ziel ist es, mögliche Risiken im laufenden Projekt zu identifizieren und soweit wie möglich abzuwehren. Die Grundlage für diesen Aktivitätstyp bildet der in 1.1.2 erstellte Risikoplan.

Beteiligte

Ausführend: Projektleiter
Mitwirkend: Projektmanager

Vorgehen

Das Risikomanagement läßt sich in drei Schritte unterteilen:

- Plan/Ist der Risiken vergleichen,
- Abwehrmaßnahmen einleiten,
- Schäden begrenzen und Ursachen analysieren.

Plan/Ist der Risiken vergleichen

Der aktuelle Projektstand wird hinsichtlich der eingetretenen Risiken und des Erfolgs der Abwehrmaßnahmen analysiert. Folgende Fragen sollten dabei berücksichtigt werden:

- Waren die eingeleiteten Abwehrmaßnahmen für die erkannten Risiken erfolgreich? Wenn nicht, so sind neue Abwehrstrategien zu entwickeln.

- Welche Risiken konnten erfolgreich abgewehrt werden? Für solche Risiken sollten gegebenenfalls die Abwehrmaßnahmen eingestellt werden.
- Sind neue Risiken aufgetreten? Wenn ja, so sind diese zu dokumentieren und entsprechende Abwehrmaßnahmen zu definieren.

Abwehrmaßnahmen einleiten

Für die neu identifizierten Risiken sollten die entwickelten Abwehrmaßnahmen eingeleitet und koordiniert werden.

Schäden begrenzen und Ursachen analysieren

Falls ein Risiko trotz Abwehrmaßnahmen eingetreten ist, sollte alles unternommen werden, um den entstehenden Schaden zu minimieren. Hierzu sind die Maßnahmen zur Schadensbegrenzung einzuleiten, die im Risikoplan definiert worden sind. Wenn die definierten Maßnahmen nicht ausreichen, so müssen zusätzliche Abwehrmaßnahmen ermittelt und eingeleitet werden. Die Ursachen des eingetretenen Risikos werden analysiert, um daraus verbesserte Abwehrmaßnahmen abzuleiten und den Projektplan zu revidieren.

Ergebnisse

Der Risikoplan sollte aktualisiert werden. Insbesondere sollten die neu erkannten Risiken dokumentiert werden.

Hinweise und Tipps

In jedem Projekt tauchen Probleme auf. Manchmal erreichen aber Probleme ein Ausmaß, daß das Projekt sich in einer Krise befindet, weil das Projekt zu Scheitern droht. Hier geht es nicht mehr um die Frage, wann das Projekt fertig wird, sondern ob überhaupt. In solchen Fällen muß das Krisenmanagement eine Bestandsaufnahme durchzuführen, um die Situation zu bewerten und Abwehrmaßnahmen ausarbeiten. Dazu sollte im Rahmen eines Workshops eine Liste der aktuellen Probleme erstellt und mit Prioritäten versehen werden. Jedem Problem sollte ein Verantwortlicher zugeordnet werden, der die Lösung vorantreiben muß. Größere Probleme sollten an spezielle Teams delegiert werden, die sich um eine Lösung bemühen müssen. Mit dieser Vorgehensweise können Probleme und ihre Lösungen koordiniert gesteuert werden.

1.2.4 Änderungen verfolgen

Zweck

Mit der evolutionären Software-Entwicklung steht dem Anwender schon sehr früh eine erste Ausbaustufe des Anwendungssystems zur Verfügung. Sobald z.B. das Anwendungssystem genutzt wird, treten Fehler und Änderungswünsche auf. Gewisse Fehlermeldungen und Änderungswünsche können im Rahmen des operationellen Einsatzes berücksichtigt und abgearbeitet werden, andere werden erst im nächsten Entwicklungszyklus behandelt. Das Ziel dieses Aktivitätstyps ist es, Änderungsanforderungen systematisch aufzunehmen, zu behandeln und gegebenenfalls den Projektplan zu revidieren.

Beteiligte

Ausführend: Projektmanager
Mitwirkend: Projektleiter, Systemarchitekt, Softwaredesigner, Softwareentwickler, Projektbibliothekar

Vorgehen

Es wird vorausgesetzt, daß ein allgemeines Vorgehen für den Änderungsablauf, wie z.B. in Abbildung 9, bereits definiert und etabliert ist. Nach diesem standardisierten Ablauf sollten Änderungsanforderungen behandelt werden. Prinzipiell sollten Änderungsanforderungen von Entwicklern beurteilt und ihre Auswirkungen ermittelt werden. Es ist zu entscheiden, in welchem Zyklus die Änderungsanforderung behandelt wird. Der Anwender sollte über eine Ablehnung oder Akzeptanz seiner Änderungsvorschläge informiert werden. Änderungs- und Konfigurationsmanagement verlaufen verzahnt ab. Denn jede Änderung verursacht die Bildung einer neuen Version bzw. Variante.

Ergebnisse

Änderungsanforderungen sollten dokumentiert werden. Ein mögliches Beschreibungsschema liefert die nächste Abbildung.

Baustein	⇨ Geben Sie hier den betroffenen Baustein an. [**Hier** klicken]
ID der Änderungs-anforderung	⇨ Geben Sie hier ein eindeutiges Merkmal zur Identifikation der Änderungsanforderung ein. Das erleichtert die Kommunikation [**Hier** klicken]
Datum	⇨ Wann wurde der Änderungsanforderung eingereicht? [**Hier** klicken]
Autor	⇨ Von wem wurde die Änderungsanforderung eingereicht (Name, Abteilung, Telefonnr.)? [**Hier** klicken]
Beschreibung	⇨ Beschreiben Sie hier kurz die Problemstellung. [**Hier** klicken]
Kategorie	⇨ Wie ist der Änderungsanforderung einzuordnen? ☐ Stabilisierung, d.h. Fehlerbehandlung ☐ Optimierung, d.h. Verbesserung der Leistung ☐ Anpassung, d.h. mit neuen Gegebenheiten abstimmen ☐ Erweiterung, d.h. funktionale Ergänzung
Dringlichkeit	⇨ Wie schnell soll die Änderungsanforderung abgearbeitet werden? ☐ sehr schnell ☐ schnell

	☐ im Rahmen des üblichen Verfahren ☐ kann zurückgestellt werden ☐ Gewünschter Fertigungstermin:
Lösungsweg	⇨ Beschreiben Sie hier kurz, wie die Lösung für die gestellte Änderungsanforderung aussieht. [**Hier** klicken]
Ver-antwortlicher	⇨ Wer ist für die Durchführung der Änderungsanforderung zuständig? [**Hier** klicken]
Kosten	⇨ Wieviel Kosten verursachen die Abwehrmaßnahmen? Diese Angabe fließt in die Projektplanung ein. [**Hier** klicken]
Terminvorgaben	⇨ Wie lange wird man für die Realisierung benötigen? Diese Angabe fließt in die Projektplanung ein. [**Hier** klicken]
Status	⇨ Beschreiben den aktuellen Status der Änderungsanforderung. ☐ offen ☐ in Bearbeitung ☐ reduziert ☐ eliminiert
Hinweise	⇨ Erkannte Risiken, Probleme oder Zusammenhänge sind hier aufzuführen. [**Hier** klicken]

Abb. 18: Beschreibungsschema für Änderungsanforderungen

1.3 Projekt-Abschluß

Zweck

Ein Projekt hat nicht nur einen formalen Beginn, sondern auch einen formalen Abschluß. Dieser Vorgang dient u.a. dazu, die im Projekt gesammelten Erfahrungen zur systematischen Verbesserung des Software-Entwicklungsprozesses zu nutzen. Außerdem wird das Projekt gedanklich abgeschlossen, um an neue Aufgaben heranzugehen.

Beteiligte

Ausführend: Projektmanager
Mitwirkend: Projektleiter

Vorgehen

Im Detail sind folgende Schritte abzuarbeiten:

- Projektverlauf und Ergebnisstand dokumentieren,

- Ist- und Planwerte abschließend vergleichen,
- Prozeß bewerten,
- Projekt abschließen.

Projektverlauf und Ergebnisstand dokumentieren

Der Ablauf und die erzielten Ergebnisse sollten beschrieben werden. Dabei sollte insbesondere auf die Veränderungen eingegangen werden, die sich im Projektverlauf hinsichtlich der Aufgabenstellung und der technischen bzw. wirtschaftlichen Aspekte ergeben haben. Welche Schwierigkeiten gab es? Welche Ursachen hatten diese und wie wurden sie bewältigt? Welche positiven bzw. negativen Erfahrungen wurden gemacht?

Damit eine kontinuierliche Anpassung des CEOS-Verfahrens an die projekt- und unternehmensspezifischen Gegebenheiten möglich ist, sollten die tatsächlich erbrachten Aufwände für die einzelnen Bausteine erfaßt werden. Die Implementierung des CEOS-Verfahrens stellt entsprechende Benutzerfunktionen zur Daten-Erfassung und -Analyse an.

Ist- und Planwerte abschließend vergleichen

Die Ist-Werte sollten der Planung gegenübergestellt werden, um Abweichungen zu identifizieren. Änderungen, wie z.B. Termin-Verzug oder Budget-Überschreitung, sollten nach Ursachen analysiert werden. Der Umgang mit solchen Änderungen oder Problemen sollte für künftige Projekte dokumentiert werden.

Prozeß bewerten

Der geplante Prozeß sollte mit dem tatsächlich durchgeführten verglichen werden. Abweichungen sollten nach Ursachen, wie z.B. zu wenige Zyklen, analysiert werden.

Projekt abschließen

Alle relevanten Projektergebnisse und –unterlagen sollten archiviert werden. Die Projektorganisation sollte aufgelöst werden, und die Mitarbeiter von ihren Projektpflichten entbunden werden.

Ergebnisse

Das Ergebnis dieses Aktivitätstyps wird in Form eines Projekt-Abschlußberichts festgehalten. Die nächste Abbildung zeigt eine mögliche Gliederung dazu.

1. Einführung und Übersicht

⇨ Beschreiben Sie hier die Projektziele, Randbedingungen und Querverbindungen.

[**Hier** klicken]

2. Projektverlauf und Ergebnis

⇨ Beschreiben Sie hier den Projektverlauf und die erzielten Ergebnisse. Welche Schwierigkeiten waren zu bewältigen?

[**Hier** klicken]

3. Plan-/Ist-Vergleich

⇨ Stellen Sie hier die Planwerte mit den tatsächlichen Werten gegenüber. Gab es einen Termin- oder Kosten-Verzug?

[**Hier** klicken]

4. Prozeßbewertung

⇨ Bewerten Sie den Software-Entwicklungsprozeß. Welche Stärken und Schwächen liegen vor?

[**Hier** klicken]

5. Hinweise

⇨ Tragen Sie hier sonstige Zusammenhänge und Erfahrungen, wie z.B. der Verlauf der Kooperation mit dem Auftraggeber, ein.

[**Hier** klicken]

Abb. 19: Beschreibungsschema für den Projekt-Abschlußbericht

In diesem Kapitel haben wir den Subprozeß „Projektmanagement“ des EOS-Modells vorgestellt. Der Subprozeß wurde in die Teile „Projekt-Initialisierung und Planung“, „Projekt-Steuerung“ und „Projekt-Abschluß“ gegliedert. Im ersten Teil wurden Aktivitäts- und Produkttypen zur Definition von Projektzielen, Anpassung des Prozeßmodells und zur Planung von Ressourcen, Risiken, Wiederwendung, Konfigurationsmanagement und Qualitätssicherung vorgestellt. Im zweiten Teil ging es um die Durchführung und Einhaltung der Planungen. Dazu waren eine kontinuierliche Projektbegleitung und ein Plan-/Ist-Vergleich anhand der definierten Revisionspunkten notwendig. Im letzten Teil wurde das Projekt formal abgeschlossen und die gemachten Erfahrungen für künftige Projekte dokumentiert. Das Projektmanagement war stets gegenstandsorientiert. Das bedeutet, daß die Bausteine zur Planungsgrundlage herangezogen wurden. Die Zyklen eines Bausteins und die an ihm ausgeführten Tätigkeiten Analyse, Entwurf, Implementierung und operationeller Einsatz wurden für die Planung genutzt.

2 Software-Entwicklung

Im EOS-Modell wird zwischen den drei Entwicklungsebenen System, Komponente/Subsystem und Klasse unterschieden. Jeder Baustein durchläuft – unabhängig von der ihm zugeordneten Ebene - die Phasen Analyse, Entwurf, Implementierung und operationeller Einsatz. Im folgenden definieren wir für die einzelnen Bausteine und ihre entsprechenden Phasen den Prozeß der Software-Entwicklung.

2.1 System-Ebene

In diesem Abschnitt wird der Entwicklungszyklus des Systems beschrieben.

2.1.1 Analyse

Das Ziel der Analyse-Phase ist es, die Wünsche und den Bedarf der Anwender an das zu entwickelnde System zu identifizieren und in einer möglichst einfachen Form zu dokumentieren. Die Aktivitätstypen und die entsprechenden Produkttypen dieser Phase sind in der Abbildung 1 in Form eines Aktivitätsdiagramms in der UML-Notation dargestellt. Es wird zwischen funktionalen und nicht-funktionalen Anforderungen unterschieden. Die funktionalen Anforderungen stellen die Systemdienste aus der Sicht des Anwenders dar. Zur Beschreibung der funktionalen Anforderungen können verschiedene Techniken verwendet werden. Wir haben uns für die weit verbreiteten Anwendungsmodelle entschieden. Nicht-funktionale Anforderungen, wie z.B. Qualitätsmerkmale der Anwendung, werden in Textform beschrieben.

Sowohl die funktionalen als auch die nicht-funktionalen Anforderungen müssen geprüft und bewertet werden. Denn oft können nicht alle Anforderungen im Rahmen des zunächst verfügbaren Budgets realisiert werden, so daß gegebenenfalls weitere Entwicklungszyklen geplant werden müssen. Parallel zu den genannten Aktivitätstypen wird eine Systemdokumentation, orientiert an dem IEEE STD 830-1993 Standard, soweit wie möglich ausgefüllt. Zum Beispiel werden die Stereotypen, die im Rahmen des Aktivitätstyps „Vorgaben definieren“ ermittelt wurden, in der Systemdokumentation festgehalten.

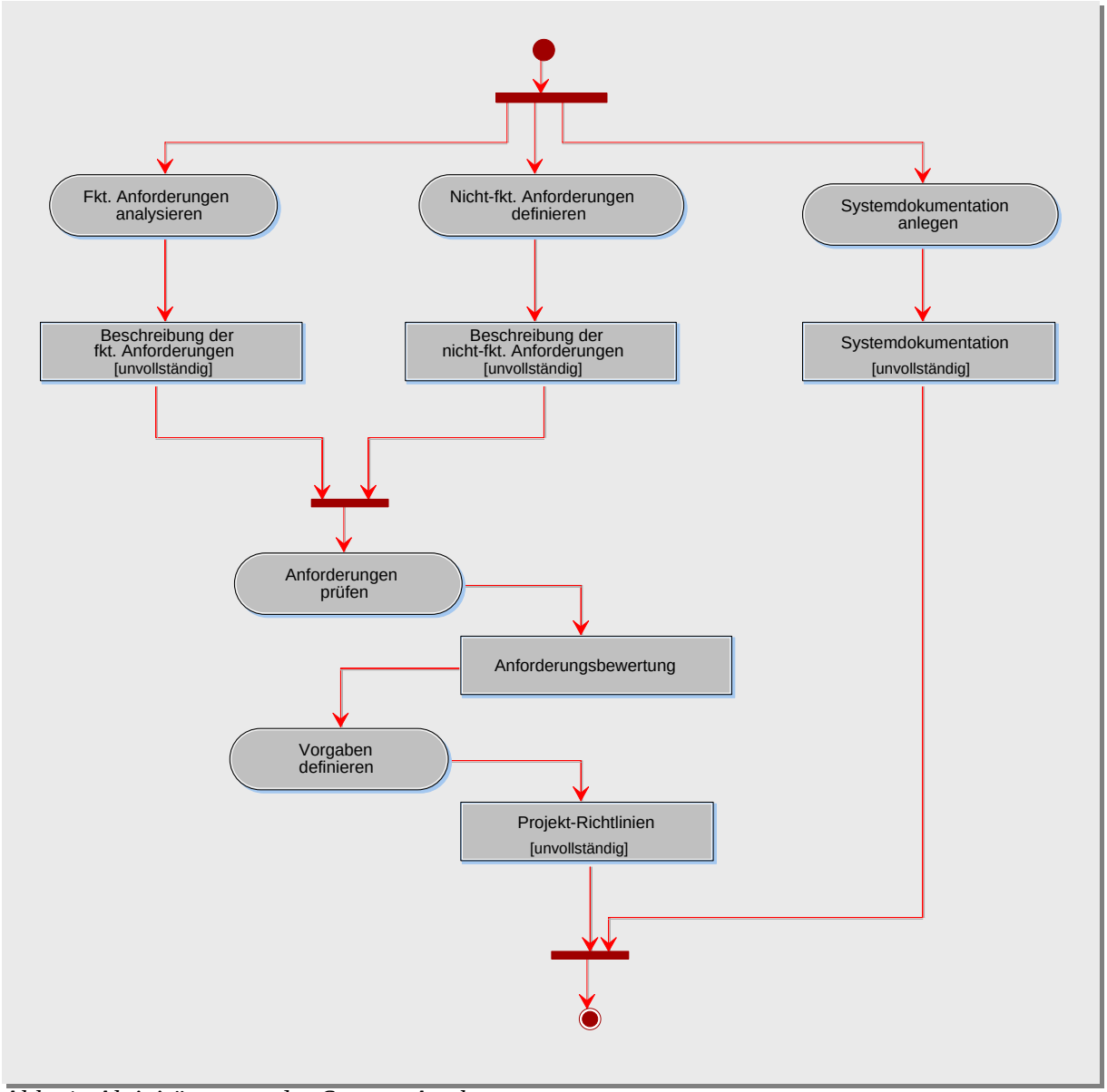


Abb. 1: Aktivitätstypen der System-Analyse

2.1.1.1 Funktionale Anforderungen analysieren

Zweck

Der Aktivitätstyp „Funktionale Anforderungen analysieren“ dient dazu, eine klare, möglichst vollständige, konsistente und überprüfbare Definition der Anforderungen zu liefern, die an das Anwendungssystem gestellt werden. Die definierten Anforderungen müssen in einer verständlichen Form dokumentiert werden, damit eine gute Kommunikation mit dem künftigen Anwender möglich ist. Denn wenn Anwender bzw. Auftraggeber die Dokumentation der Anforderungen nicht verstehen, werden sie sich daran nicht gebunden fühlen. Daraus resultiert ein potenzieller Risikofaktor hinsichtlich des Projekterfolges.

Eine mögliche Form der Dokumentation, die die obige Bedingung erfüllt, ist der Einsatz von Anwendungsfalldiagrammen. Die in einem Anwendungsfalldiagramm enthaltenen Anwendungsfälle repräsentieren die funktionalen Anforderungen. Neben der Beschreibung der Systemfunktionalität auf hoher Ebene können die Anwendungsfälle zur Definition von Entwicklungs- und Testeinheiten genutzt werden. Es gibt aber noch weitere Techniken zur

Anforderungsanalyse, die in der Softwaretechnik im Rahmen des *Requirements-Engineering* behandelt werden. Wir haben uns für die weit verbreitete und bewährte Technik der Anwendungsfall-Modellierung entschieden.

Beteiligte

Ausführend: Systemanalytiker
Mitwirkend: Anwender

Vorgehen

Das Ziel dieses Tätigkeitstyps ist das Entwickeln eines Anwendungsfallmodells. Hierzu sind folgende Schritte zu durchlaufen:

- Akteure finden,
- Akteure beschreiben,
- Anwendungsfälle finden,
- Anwendungsfälle beschreiben,
- Anwendungsfalldiagramme erstellen,
- Aktivitätsdiagramme zu Anwendungsfällen erstellen,
- Prototypen zu Anwendungsfällen erstellen,
- Erkannte Risiken dokumentieren,
- Anwendungsfallmodell prüfen.

Akteure finden

Je nachdem, ob ein Geschäftsprozeßmodell vorliegt oder nicht, variiert das Vorgehen. Sollte kein Geschäftsprozeßmodell vorliegen, dann werden zunächst die Rollen der Personen und Systeme, die mit der künftigen Anwendung zusammenarbeiten, identifiziert. Die Akteure sind – wie wir im nächsten Schritt sehen werden – in erster Linie Mittel zum Zweck der Auffindung von Anwendungsfällen. Alles, was mit dem Anwendungssystem interagiert, ist ein Kandidat für einen Akteur. Die primären Akteure können mit der folgenden Fragestellung ermittelt werden:

- Wer braucht das Anwendungssystem, um seine täglichen Aufgaben durchführen zu können?

Sekundäre Akteure können durch die nachfolgenden Fragen gefunden werden:

- Wer wird das künftige Anwendungssystem administrieren und pflegen?
- Welche sind die externen Systeme, mit denen das zu entwickelnde Anwendungssystem zusammenarbeiten wird?

Liegt ein Geschäftsprozeßmodell vor, dann ist jede Rolle im Geschäftsprozeßmodell ein Kandidat für einen Akteur. Die beteiligten Rollen im Geschäftsprozeßmodell entsprechen den Swimlanes (siehe UML-Spezifikation) der jeweiligen Aktivitätsdiagramme. Repräsentiert eine Rolle eine Organisation oder Interessengruppe, so empfiehlt es sich, daraus mehrere handelnde Akteure abzuleiten. Denn sonst besteht die Gefahr, daß die aus den Akteuren gebildeten Anwendungsfälle, begrifflich auf einem zu abstrakten Niveau angesiedelt sind. Ein

häufiger Fehler beim Auffinden von Akteuren ist, daß externe Systeme, mit denen das Anwendungssystem zusammenarbeitet, nicht als Akteure aufgenommen werden.

Akteure beschreiben

Zuerst werden die Akteure in die Kategorien „primär“ bzw. „sekundär“ klassifiziert. Danach werden die Akteure in Textform beschrieben. Dabei müssen die Ziele, die der entsprechende Akteur mit dem künftigen Anwendungssystem erreichen möchte, dargestellt werden. Zuletzt sollten die Ziele mit Prioritäten versehen werden, um die Projektplanung zu unterstützen. Die nächste Abbildung stellt ein Muster dar, das zur Beschreibung von Akteuren verwendet werden kann.

<table>
<tr><td>Typ</td><td colspan="3">⇨ Dieser Eintrag kann entfallen, wenn Sie den Typ des Akteurs mit Hilfe von entsprechenden Stereotypen abbilden.
☑ primär
☐ sekundär</td></tr>
<tr><td>Ziele</td><td>#</td><td>Beschreibung</td><td>Priorität</td></tr>
<tr><td></td><td>1</td><td>⇨ Beschreiben Sie hier für jeden primären Akteur die fachlichen Ziele, die er in Interaktion mit dem zu entwickelnden System erreichen will. Aus jedem hier beschriebenen Ziel sollten Sie später einen Anwendungsfall ableiten.
[Hier klicken]</td><td>⇨ Nicht alle Ziele sind gleich wichtig. Vergeben Sie hier pro Ziel eine Priorität, z. B. zwischen 1 wie sehr hoch und 5 wie sehr gering.
[Hier klicken]</td></tr>
<tr><td></td><td>2</td><td>[Hier klicken]</td><td>[Hier klicken]</td></tr>
<tr><td>Beschreibung</td><td>#</td><td>Hauptaufgaben</td><td>Systeminformationen</td><td>Zugriff</td></tr>
<tr><td></td><td>1</td><td>⇨ Speziell für sekundäre Akteure kommen Sie häufig besser über eine Aufgabenliste zu Anwendungsfällen. Stellen Sie die Aufgaben hier zusammen.
[Hier klicken]</td><td>⇨ Welche Informationen des Systems muß der Akteur pro Aufgabe im Zugriff haben?
[Hier klicken]</td><td>⇨ lesend/ schreibend/ ändernd
[Hier klicken]</td></tr>
<tr><td></td><td>2</td><td>[Hier klicken]</td><td>[Hier klicken]</td><td>[Hier klicken]</td></tr>
<tr><td>Informationsbedarf des Akteurs</td><td colspan="3">⇨ Über welche systeminternen Veränderungen muß das System den Akteur unbedingt informieren?
[Hier klicken und Text eingeben]</td></tr>
<tr><td>Informationsbedarf des Systems</td><td colspan="3">⇨ Über welche äußeren Veränderungen muß der Akteur das System unbedingt informieren?
[Hier klicken und Text eingeben]</td></tr>
<tr><td>Profil</td><td colspan="3">⇨ Handelt es sich bei dem Akteur um einen Bediener am Bildschirm, dann beschreiben Sie hier sein Benutzerprofil: Ist er ein ständiger oder gelegentlicher Benutzer? Ist er im Umgang mit einem Dialogsystem geübt? Welche fachlichen Voraussetzungen bringt der Benutzer mit? ... etc.
[Hier klicken und Text eingeben]</td></tr>
</table>

Abb. 2: Beschreibungsschema für Akteure /objectiF 4.5, microTOOL GmbH/

Anwendungsfälle finden

Für den Anwender beschreibt ein Anwendungsfall eine sichtbare Funktionalität, durch die ein diskretes Ziel erreicht wird. Das bedeutet, daß aus jedem zuvor erkannten Ziel eines Akteurs ein Anwendungsfall abgeleitet werden sollte. Die nachfolgenden Fragen können bei der Identifizierung von Anwendungsfällen helfen:

- Welche Aufgaben verbindet ein Akteur mit seinen Zielen?
- Welche Funktionalität erwartet ein Akteur von dem zu realisierenden Anwendungssystem?

Oben wurde das Vorgehen beschrieben, falls kein Geschäftsprozeßmodell vorliegt. Existiert aber solch ein Modell, dann kann es zur Ermittlung von Anwendungsfällen genutzt werden. Und zwar können alle Aktivitäten eines Geschäftsprozesses, die nacheinander ohne Rollenwechsel ablaufen, zu einem Anwendungsfall-Kandidaten zusammengefaßt werden. Solche gefundenen Kandidaten können sehr komplex sein, so daß sie gegebenenfalls zerlegt werden müssen. Eine bewährte Möglichkeit zur Zerlegung ist, die Menge der Aktivitäten nach einem Muster, wie z.B. Vorbereitung, Freigabe und Durchführung, zu gliedern und die Aktivitäten entsprechend dieser Gliederung in mehrere Anwendungsfälle aufzuteilen. Aber auch der umgekehrte Fall kann vorkommen, d.h. eine Aktivität ist so komplex, daß sie auf mehrere Anwendungsfälle aufgeteilt werden muß.

Zur besseren Lesbarkeit empfiehlt es sich, den Anwendungsfällen funktionsorientierte Bezeichner zu geben, die jeweils aus einem Substantiv und einem Verb bestehen (Beispiel: *Reise buchen*).

Anwendungsfall beschreiben

Für die Entwicklung eines Anwendungssystems ist eine genaue Beschreibung der Anwendungsfälle notwendig. Das Anwendungsfalldiagramm ist für sich genommen ziemlich aussagearm. Bei der Beschreibung sollten folgende Punkte beachtet werden: Vor- und Nachbedingung eines Anwendungsfalls, das Hauptszenario, Nebenszenarien, Ausnahme- und Fehlerbedingungen. Die nächste Abbildung stellt ein Beschreibungsschema für Anwendungsfälle dar.

Bezeichner	⇨ Geben Sie hier die Bezeichnung des Anwendungsfalls ein. [**Hier** klicken und Text eingeben]
Ziel	⇨ Welches fachliche Ziel wird mit seiner Durchführung verfolgt? Auf diesen Eintrag könnte verzichtet werden, wenn der Bezeichner des Anwendungsfalls das Ziel bereits unmißverständlich ausdrückt. [**Hier** klicken und Text eingeben]
Kontext der Verwendung	⇨ In welchem fachlichen Zusammenhang tritt der Anwendungsfall auf? [**Hier** klicken und Text eingeben]
Rahmen	⇨ Definieren Sie hier das Anwendungssystem, das entworfen werden soll, das zunächst jedoch noch als Blackbox betrachtet wird. Grenzen Sie ab: Was gehört dazu und was nicht? [**Hier** klicken und Text eingeben]
Ebene	⇨ Auf welchem Detaillierungsniveau befindet sich der hier beschriebene Anwendungsfall? Handelt es sich um einen Überblick oder eine Zusammenfassung, eine Funktion oder eine Teilfunktion?

	[**Hier** klicken und Text eingeben]	
Primäre Akteure (optional)	⇨ Wer ist der Hauptakteur (Rollenname)? Gibt es daneben weitere primäre Akteure? Dieser Eintrag ist verzichtbar, wenn Sie den Zusammenhang zwischen Akteuren und Anwendungsfällen in Anwendungsfalldiagrammen grafisch modellieren.	
	[**Hier** klicken und Text eingeben]	
Sekundäre Akteure (optional)	⇨ Wen benötigt das System, um den Anwendungsfall auszuführen? Dieser Eintrag ist verzichtbar, wenn Sie den Zusammenhang zwischen Akteuren und Anwendungsfällen in Anwendungsfalldiagrammen grafisch modellieren.	
	[**Hier** klicken und Text eingeben]	
Interessens-gruppe & Interesse (optional)	**Interessensgruppe**	**Interesse**
	⇨ Beschreiben Sie hier, wer (zum Beispiel aus dem Management) über die rein funktionale Anforderung hinaus an der Realisierung des Anwendungsfalls interessiert ist.	⇨ Welcher Art ist das Interesse an dem Anwendungsfall? Geht es zum Beispiel um die Sicherung der Wettbewerbsfähigkeit um handfeste, finanzielle Vorteile oder Marketing-Interessen?
	[**Hier** klicken und Text eingeben]	[**Hier** klicken und Text eingeben]
Vorbedingungen	⇨ Welcher Zustand liegt vor dem Beginn des Anwendungsfalls vor? Es wird davon ausgegangen, daß der hier beschriebene Zustand für den Anwendungsfall immer gegeben ist und nicht innerhalb des Anwendungsfalls abgeprüft wird. Eine informale Beschreibung ist ausreichend.	
	[**Hier** klicken und Text eingeben]	
Nachbedingung im Positivfall	⇨ Welcher Zustand liegt nach dem erfolgreichem Abschluß des Anwendungsfalls vor? Welche Interessen wurden damit abgedeckt und welche Anwenderziele erreicht? Eine informale Beschreibung ist ausreichend.	
	[**Hier** klicken und Text eingeben]	
Nachbedingung im Negativfall	⇨ Auch im Fehlerfall, wenn das angestrebte Ziel nicht erreicht wird, muß dafür gesorgt werden, daß sich das System in einem definierten Zustand befindet. Beschreiben Sie, was bei erfolglosem Abschluß oder Abbruch des Anwendungsfalls gewährleistet werden muß. Verwenden Sie dazu am besten folgendes Schema: Wenn <Art des Abbruchs oder Fehlers>, dann <sicherzustellender Zustand>.	
	[**Hier** klicken und Text eingeben]	
Auslöser	⇨ Welche Aktion löst den Anwendungsfall aus? Es kann sich dabei auch um ein Zeitereignis handeln.	
	[**Hier** klicken und Text eingeben]	
Beschreibung	#	**Interaktionsschritte** ⇨ Beschreiben Sie hier die Schritte der Hauptszenarien, d.h. den erfolgreichen Durchlauf des Anwendungsfalls von der auslösenden Aktion bis zum gewünschten Ergebnis. Hilfreich ist dabei die Orientierung an dem Schema: System tut – Akteur tut – System tut – Akteur tut – .... Diese Technik geht auf Rebecca Wirfs-Brock zurück. Sie nennt sie treffend „a conversation“.
	1	[**Hier** klicken und Text eingeben]
	2	[**Hier** klicken und Text eingeben]
	3	[**Hier** klicken und Text eingeben]
Erweiterungen	#	**Abweichende Interaktionen** ⇨ Machen Sie hier jeweils Einträge mit folgender Struktur: <Bedingung, die zur Erweiterung der Hauptszenarien führt>: <Aktion oder Name des erweiternden Anwendungsfalls>
	1a	[**Hier** klicken und Text eingeben]

Variationen	**Von Schritt #**	**Abweichende Interaktionen** ⇨ Ein Interaktionsschritt kann ggf. technisch unterschiedlich realisiert werden. Beispiel: Bediener identifiziert sich beim System. Dies kann mit einer Karte und Eingabe einer PIN-Nummer, durch Eintippen eines Passworts, Scannen eines Finderabdrucks oder der Augenpartie geschehen. Führen Sie die technischen Variationen für einen Interaktionsschritt hier auf.
	1	[**Hier** klicken und Text eingeben]
Enthaltene Anwendungsfälle	**in Schritt #**	**Enthaltene Anwendungsfälle** ⇨ Wird anstelle eines Interaktionsschritts ein anderer Anwendungsfall ausgeführt? D. h., ist an der Stelle des angegebenen Schrittes ein separat definierter Anwendungsfall im Ablauf enthalten? Dann führen Sie hier seinen Namen an.
	1	[**Hier** klicken und Text eingeben]
Zusatz-informationen	⇨ Name des Anwendungsfalls [**Hier** klicken und Text eingeben]	
Kritikalität	⇨ Wie kritisch ist der Anwendungsfall für das System bzw. Ihre Organisation? [**Hier** klicken und Text eingeben]	
Performanz	⇨ Wie lange darf der Anwendungsfall dauern (die Performanz des Systems resultiert aus der Performanz der Anwendungsfälle)? [**Hier** klicken und Text eingeben]	
Häufigkeit	⇨ Wie oft wird der Anwendungsfall eintreten? [**Hier** klicken und Text eingeben]	
Übergeordneter Anwendungsfall	⇨ Name des Anwendungsfalls, der diesen enthält. [**Hier** klicken und Text eingeben]	
Untergeordnete Anwendungsfälle	⇨ Namen der Anwendungsfälle, die in diesem enthalten sind. [**Hier** klicken und Text eingeben]	
Übertragungs-kanal zum primären Akteur	⇨ Beschreiben Sie über welches Medium mit dem Akteur kommuniziert wird: im Dialog, per e-Mail, durch Austausch von Dateien, über Datenbanken etc. Wie sind die zu übertragenden Daten formatiert? Wie sehen die Übertragungsprotokolle aus? [**Hier** klicken und Text eingeben]	
Übertragungska-näle zu sekundä-ren Akteuren	⇨ Beschreiben Sie, über welches Medium mit dem Akteur kommuniziert wird: im Dialog, per e-Mail, durch Austausch von Dateien, über Datenbanken etc. Wie sind di zu übertragenden Daten formatiert? Wie sehen die Übertragungsprotokolle aus? [**Hier** klicken und Text eingeben]	
Offene Punkte	⇨ Zählen Sie in Form einer Liste auf, welche Entscheidungen bzw. Besonderheiten zum Anwendungsfall noch offen sind. [**Hier** klicken und Text eingeben]	
Geplant für	⇨ Wann oder in welchem Release soll der Anwendungsfall maschinell unterstützt sein? [**Hier** klicken und Text eingeben]	
Management-informationen	⇨ ...nach Bedarf [**Hier** klicken und Text eingeben]	

Abb. 3: Beschreibungsschema für Anwendungsfälle /objectiF 4.5, microTOOL GmbH/

Anwendungsfalldiagramm erstellen

Die Funktionalität eines Anwendungssystems wird in Teilbereichen zerlegt und durch Anwendungsfalldiagramme beschrieben. Damit sollte ein Anwendungsfalldiagramm nur Anwendungsfälle enthalten, die eine gemeinsame Leistung erbringen. Da das menschliche Aufnahmevermögen begrenzt ist, sollte ein Anwendungsfalldiagramm nicht mehr als drei Akteure aufweisen, die jeweils mit maximal drei bis fünf Anwendungsfällen kommunizieren.

In der Regel werden Anwendungsfälle aufgrund von Ausnahmen oder Fehlerbedingungen erweitert. Diese Erweiterung kann als Text oder als eigener Anwendungsfall mit Hilfe der *Erweiterungsbeziehung* modelliert werden. Wenn mehrere Anwendungsfälle gleiches Verhalten umfassen, so kann dieses Verhalten als ein eigenständiger Anwendungsfall mit Hilfe der *Enthält-Beziehung* modelliert werden. Das heißt, daß ein Anwendungsfall, der nur ein einziges mal „benutzt“ wird, auf eine fehlerhafte Modellierung deutet. Die Abstraktion von mehreren Anwendungsfällen zu einem verallgemeinerten Anwendungsfall kann mit der *Generalisierung* modelliert werden.

Ein Anwendungsfalldiagramm kann mit dem folgenden Beschreibungsschema dokumentiert werden.

Bezeichner	⇨ Geben Sie hier die Bezeichnung des Anwendungsfalldiagramms ein. [**Hier** klicken und Test eingeben]
Zweck	⇨ Beschreiben Sie den funktionalen Kontext, in dem die in diesem Diagramm dargestellten Anwendungsfälle stehen. Begründen Sie Ihre Entwurfsentscheidung. [**Hier** klicken und Text eingeben]
Annahmen und Restriktionen	⇨ Liegen der Gruppierung der Anwendungsfälle und der Darstellung des Anwendungsfalldiagramms spezielle Annahmen oder Restriktionen zu Grunde? [**Hier** klicken und Text eingeben]
Erkenntnisse und Konsequenzen	⇨ Was haben Sie bei der Gestaltung des Anwendungsfalldiagramms erkannt, welche Konsequenzen sind insbesondere hinsichtlich der Realisierung und Gliederung des Anwendungssystems daraus zu ziehen? [**Hier** klicken und Text eingeben]
Offene Fragen	⇨ Welche Gestaltungsentscheidungen stehen noch aus? [**Hier** klicken und Text eingeben]
Bearbeitungs-status	⇨ in Bearbeitung, abgenommen, etc. [**Hier** klicken und Text eingeben]

Abb. 4: Beschreibungsschema für Anwendungsfalldiagramme /objectiF 4.5, microTOOL GmbH/

Aktivitätsdiagramme zu Anwendungsfällen erstellen

Es kann auch vorkommen, daß ein Anwendungsfall sich nicht durch eine einfache Sequenz von Aktivitäten beschreiben läßt. In solch einem Fall wird zur Spezifikation ein Aktivitätsdiagramm verwendet. Ziel ist es hierbei, den funktionalen Umfang des Anwendungsfalls abzugrenzen. Einen komplexen Anwendungsfall, wie z.B. „Reise reservieren“, würde man durch ein Aktivitätsdiagramm näher spezifizieren, um alle Sonderfälle berücksichtigen zu können. Beim Entwurf werden die Anwendungsfälle mit Klassen-diagrammen technisch spezifiziert. Denn ein Anwendungsfall stellt eine Systemfunktion für den Anwender dar, die technisch realisiert werden muß.

Prototypen zu Anwendungsfällen erstellen

Um Mißverständnisse und Unklarheiten aufzudecken, kann es sinnvoll sein, für einige Anwendungsfälle einen Prototypen zu bauen. Damit kann dem Anwender die Funktionalität des zu entwickelnden Anwendungssystems anschaulich mit Hilfe einer Benutzeroberfläche präsentiert werden. Das führt in der Regel zu einer frühzeitigen Akzeptanz des zu realisierenden Anwendungssystems und verringert Risiken. Ob ein explorativer, experimenteller oder evolutionärer Prototyp entwickelt wird, hängt vom Anwendungssystem und Auftraggeber ab. Detaillierte Informationen über Prototyping können in /Balzert 1998/ und /Oestereich 1999/ nachgelesen werden.

Erkannte Risiken dokumentieren

Wenn bei der Modellierung der Anwendungsfälle Risiken erkannt werden, wie z.B. ein größerer Umfang des Anwendungssystems als zunächst erwartet, sollten diese dokumentiert werden, damit sie bei der Risikomanagementplanung berücksichtigt werden können.

Anwendungsfallmodell prüfen

Bevor die erzielten Ergebnisse in die Qualitätssicherung gehen, sollte eine Selbstprüfung bezüglich der Vollständigkeit und Korrektheit unternommen werden. Auf folgende formale Aspekte sollte geachtet werden:

- Sind alle Akteure und Anwendungsfälle textuell beschrieben worden?
- Gibt es Anwendungsfälle oder Akteure ohne Beziehungen?

Bei der inhaltlichen Prüfung sollten folgende Punkte berücksichtigt werden:

- Gibt es für jedes Ziel eines Akteurs auch mindestens einen Anwendungsfall?
- Gibt es funktionale Anforderungen, für die kein Anwendungsfall existiert?
- Sind die Interaktionsschritte für die einzelnen Szenarien korrekt?
- Sind die Erweiterungs- und Enthältbeziehungen sowie die Generalisierung semantisch korrekt verwendet worden?

Ergebnisse

Das Ergebnis der Anforderungsanalyse sind dokumentierte und geprüfte Anwendungsfalldiagramme, die zum jetzigen Zeitpunkt noch unvollständig sein können.

Hinweise und Tipps

- Bei der Wahl von Akteuren sollte beachtet werden, das ein Akteur niemals Teil des Anwendungssystems ist, sondern stets außerhalb des Anwendungssystems steht.
- Die Erweiterungs- und Enthältbeziehungen dürfen nicht zur funktionalen Dekomposition verwendet werden. Anwendungsfalldiagramme beinhalten nach Definition niemals eine funktionale Dekomposition.
- Ein Anwendungsfall, der nur ein einziges mal „benutzt“ wird, deutet auf einen Modellierungsfehler hin.

2.1.1.2 Nicht-funktionale Anforderungen definieren

Zweck

Alle Anforderungen, die sich nicht auf die Funktionalität des zu realisierenden Anwendungssystems beziehen, werden in diesem Schritt ermittelt. Zu den nicht-funktionalen Anforderungen eines Anwendungssystems zählen:

- Vorgaben für die Erfüllung von Qualitätsmerkmalen, wie z.B. Sicherheit und Zuverlässigkeit,
- Hard- und Software Restriktionen,
- Vorgaben an die Software-Architektur, z.B. Verteil- und Skalierbarkeit,
- Vorgaben zur Wiederverwendung oder Kauf von Fertigsoftware,
- Vorgaben zu Software-Ergonomie.

Beteiligte

Ausführend: Systemanalytiker
Mitwirkend: Anwender

Vorgehen

Die folgenden vier Schritte beschreiben den Weg zur Definition von nicht-funktionalen Anforderungen:

- Nicht-funktionale Anforderungen ermitteln,
- Anforderungen an Systemarchitektur und Zielumgebung dokumentieren,
- erkannte Risiken dokumentieren,
- nicht-funktionale Anforderungen prüfen.

Nicht-funktionale Anforderungen ermitteln

Dieser Aktivitätstyp steht für die Ermittlung von Merkmalen im Rahmen der Zusammenarbeit des Anwenders mit dem Anwendungssystem. Zum Erfassen dieser Anforderungen muß gemeinsam mit dem Anwender für jede Kommunikationsbeziehung zwischen einem Akteur und einem Anwendungsfall die in der nächsten Abbildung dargestellte Tabelle ausgefüllt werden.

Anforderungen	⇨ Anforderungen an die Kommunikation zwischen <Akteur> und <Anwendungsfall> [**Hier** klicken und Text eingeben]
Antwortzeit-verhalten	⇨ Wie schnell erwartet der Akteur das Ergebnis des Anwendungsfalls? Quantifizieren Sie hier das Antwortzeitverhalten, sofern Sie es nicht bereits unter den Zusatzinformationen zum Anwendungsfall getan haben. [**Hier** klicken und Text eingeben]
System-verfügbarkeit	⇨ Von wann bis wann erwartet der Akteur die Verfügbarkeit des Systems? [**Hier** klicken und Text eingeben]

Zuverlässigkeit	⇨ Welche Zeit zwischen dem Auftreten von Fehlern ist für den Akteur akzeptabel? [**Hier** klicken und Text eingeben]
Sicherheit	⇨ Welche Backup- und Recovery-Mechanismen sind für die erfolgreiche Arbeit des Akteurs unerläßlich? [**Hier** klicken und Text eingeben]
Schutz	⇨ In welchem Umfang ist Schutz vor unberechtigten Benutzern im Rahmen des Anwendungsfalls vorzusehen? [**Hier** klicken und Text eingeben]
Ressourcen-belegung	⇨ Welche Ressourcen wie Hauptspeicher, Plattenplatz, Leitungen etc. stehen einem Akteur zu bzw. dürfen im Rahmen des Anwendungsfalls in welchem Umfang belegt werden? [**Hier** klicken und Text eingeben]
Dokumentation	⇨ In welchem Umfang benötigt ein Akteur Hilfe zu einem Anwendungsfall und in welcher Form (Online-Hilfe, Referenzkarte, Handbuch etc.) soll sie angeboten werden? [**Hier** klicken und Text eingeben]
Erweiterbarkeit	⇨ Welche zukünftigen Erweiterungen des Anwendungsfalls müssen in Hinsicht auf diese Kommunikationsbeziehung berücksichtigt werden? [**Hier** klicken und Text eingeben]
Portabilität	⇨ Auf welchen Systemen muß der Anwendungsfall ablauffähig sein? [**Hier** klicken und Text eingeben]
Sonstige	[**Hier** klicken und Text eingeben]

Abb. 5: Nicht-funktionale Anforderungen /objectiF 4.5, microTOOL GmbH/

Anforderungen an Systemarchitektur und Zielumgebung dokumentieren

Der in Abbildung 6 vorgestellte Gliederungsvorschlag kann verwendet werden, um die nicht-funktionalen Anforderungen zu beschreiben. Die Punkte 2 bis 10 können als Checkliste für die Erfassung aller Anforderungen, die mit der technischen Umgebung und Realisierung der Lösung zu tun haben, verwendet werden (Punkt 1 verweist auf die qualifizierbaren nicht-funktionalen Anforderungen).

1 Anforderungen an das Systemverhalten
2 Anforderungen an die Hardware
3 Anforderungen an die Systemsoftware
4 Anforderungen an die Software-Architektur
5 Anforderungen an die Wiederverwendung
6 Anforderungen zur Portabilität des Systems
7 Anforderungen an die Gestaltung der Benutzeroberfläche
8 Einzuhaltende Standards
9 Anforderungen an die Erweiterbarkeit des Systems
10 Sonstige

Abb. 6: Beschreibungsschema für nicht-funktionale Anforderungen /IBM 1997/

Erkannte Risiken dokumentieren

Wenn bei der Definition der nicht-funktionalen Anforderungen Risiken erkannt werden, so sind diese zu dokumentieren. Die erkannten Risiken dienen als Planungsgrundlage für das Risikomanagement.

Nicht-funktionale Anforderungen prüfen

Hier sind einige Punkte, die bei der Prüfung der nicht-funktionalen Anforderungen beachtet werden sollten:

- Wurden alle Kommunikationsbeziehungen bei der Definition der nicht-funktionalen Anforderungen berücksichtigt?
- Sind alle formulierten nicht-funktionalen Anforderungen notwendig und sinnvoll?
- Sind die formulierten nicht-funktionalen Anforderungen widerspruchsfrei?

Ergebnisse

Die Beschreibungsschema in Abbildung 5 und 6 sind auszufüllen.

2.1.1.3 Anforderungen prüfen

Zweck

Die funktionalen und nicht-funktionalen Anforderungen sollten hinsichtlich ihrer Erfüllbarkeit und Dringlichkeit mit dem Anwender bzw. Auftraggeber bewertet werden, um Mißerfolge und deren mögliche Konsequenzen zu vermeiden. Denn in der Regel können nicht alle Anforderungen im Rahmen des Zeit- bzw. Geld-Budgets realisiert werden. Daher ist es die Aufgabe des Projektmanagers, mit dem Anwender bzw. Auftraggeber einen Konsens bezüglich der Wirtschaftlichkeit und Realisierbarkeit der Anforderungen zu finden.

Beteiligte

Ausführend: Auftraggeber bzw. Anwender, Projektmanager
Mitwirkend: Projektleiter, Systemanalytiker

Vorgehen

Folgende Schritte müssen durchlaufen werden:

- Anforderungen analysieren und bewerten,
- über Anforderungen entscheiden,
- Entscheidungen dokumentieren.

Anforderungen analysieren und bewerten

Alle Anforderungen sollten kritisch analysiert und bewertet werden. Die nachfolgenden Fragen sollen hierbei unterstützen:

- Wie wichtig ist die Anforderung für das zu entwickelnde Anwendungssystem?
- Welche Anforderungen haben Priorität, falls konkurrierende Anforderungen vorliegen?
- Wann wird die geforderte Funktionalität benötigt?
- Wie komplex sind die Anforderungen und welche Risiken bergen sie?
- Was kostet die Realisierung der einzelnen Anforderungen?

- Kann statt der Realisierung auf gekaufte oder wiederverwendete Software zurückgegriffen werden?

Aufgrund dieser Analyse müssen gegebenenfalls Anforderungen modifiziert und reduziert werden, die Entwicklungsziele sollten aber nicht gefährdet werden.

Über Anforderungen entscheiden

Es muß festgelegt werden, welche Anforderungen in welchem Umfang und in welchem Zyklus realisiert werden sollen. Dazu dient die Bewertung der Anforderungen im vorherigen Schritt.

Entscheidungen dokumentieren

Die getroffenen Entscheidungen beeinflußen die Projektplanung, so daß insbesondere der Risikomanagementplan und der Qualitätssicherungsplan überarbeitet werden sollten. Die Bewertung der Anforderungen kann mit den folgenden Tabellen dokumentiert werden.

Anwendungsfälle (funktionale Anforderungen)	Bedeutung	Dringlichkeit	Komplexität	Risiko	Alternativen	Realisierender Zyklus
Reise buchen	2	2	1	1	Keine	1

Abb. 7: Funktionale Anforderungen /IBM 1997/

Nicht-funktionale Anforderungen	Bedeutung	Dringlichkeit	Komplexität	Risiko	Alternativen	Realisierender Zyklus
Netzwerkfähig	3	3	2	2	Keine	2

Abb. 8: Nicht-funktionale Anforderungen /IBM 1997/

Bedeutung:	1 = entscheidend, 2 = wichtig, 3 = wäre schön, 4 = wäre schön, wenn verfügbar
Dringlichkeit:	1 = wird umgehend benötigt, 2 = dringend, 3 = kann warten
Komplexität:	1 = gering, 2 = hoch, 3 = unbeherrschbar
Risiko:	1 = gering, 2 0 hoch, 3 = untragbar
Alternativen:	Welche Alternativen gibt es noch?
Realisierender Zyklus:	In welchem Zyklus soll die Anforderung realisiert werden?

Ergebnisse

Die Beschreibungsschema in der Abbildung 7 und 8 sollten ausgefüllt werden.

Hinweise und Tipps

- Wenn die Anforderungen durchweg mit „mittel“ bewertet werden, so sind differenziertere Skalen zu wählen.
- Komplexe Anwendungsfälle mit hoher Priorität sollten sorgfältig untersucht werden, denn möglicherweise sind Enthält- oder Erweiterungsbeziehungen übersehen worden.

2.1.1.4 Vorgaben definieren

Zweck

Damit eine einheitliche und leichtverständliche Lösung erarbeitet werden kann, sollten verbindliche Vorgaben, wie z.B. Richtlinien für die Benutzeroberfläche, festgelegt werden. Einheitliche Standards erleichtern auch eine schnelle Einarbeitung von später hinzukommenden Teammitgliedern.

Beteiligte

Ausführend: Projektleiter
Mitwirkend: Systemanalytiker, Softwareentwickler

Vorgehen

Es sind folgende Tätigkeiten auszuführen:

- Stereotypen festlegen,
- Einsatz von Mustern und wiederzuverwendender Software regeln
- Richtlinien für die Benutzeroberfläche definieren

Stereotypen festlegen

Stereotypen können verwendet werden, um die Ausdrucksmittel der UML zu erweitern. Wenn solche Erweiterungen genutzt werden, dann sollte dafür gesorgt werden, daß alle Teammitglieder mit demselben Vorrat an Stereotypen arbeiten. So wird die Voraussetzung für ein einheitliches Arbeiten geschaffen. Die Liste der Stereotypen wird in der Regel während der Entwurfs-Phase erweitert werden. Folgende Punkte sollten für jeden Stereotyp beachtet werden:

- Der Zweck eines Stereotyps sollte definiert werden.
- Stereotypen sind Spezialisierungen von UML-Modellelementen. Daher sollte beschrieben werden, welche zusätzliche Semantik dem vorhandenen Modellelement hinzugefügt wird.
- Oft ist es hilfreich, wenn Beispiele angegeben werden, die die Verwendung der definierten Stereotypen veranschaulichen.

Einsatz von Mustern und wiederzuverwendender Software regeln

In diesem Schritt sollte geprüft werden, ob die Verwendung von Mustern und die Wiederverwendung von Software für das zu realisierende Anwendungssystem erfolgsversprechend sein könnten. Sollte das der Fall sein, so müssen entsprechende Quellen dem Team verfügbar gemacht werden.

Richtlinien für die Benutzeroberfläche definieren

Dieser Aktivitätstyp dient dazu, Richtlinien für die Gestaltung der grafischen Benutzeroberfläche zu definieren. Das Ziel sollte es sein, eine benutzerfreundliche Oberfläche

zu entwickeln. Dazu müssen Vorgaben für den Dialogfluß und das Bildschirm-Layout spezifiziert werden und gegebenenfalls durch Screenshots demonstriert werden.

Ergebnisse

Die definierten Stereotypen und die Richtlinien für die Benutzeroberflächen werden in der Systemdokumentation (Abb. 9) dokumentiert. Die Quellen für Muster und wiederverwendete Software werden ebenfalls in der Systemdokumentation festgehalten.

Hinweise und Tipps

- Zu viele Stereotypen sollten vermieden werden, sonst werden die Modelle zu sehr überladen und unverständlicher.
- Die Liste der Stereotypen kann zu jedem Zeitpunkt im Projekt ergänzt werden.

2.1.1.5 Systemdokumentation anlegen

Zweck

Parallel zu den Aktivitätstypen der Analyse-Phase wird eine Systemdokumentation angelegt, in der die ermittelten Ergebnisse systematisch zusammengefaßt werden. Als praxisrelevant hat sich z.B. der Standard „IEEE STD 830-1993“ bewährt, so daß es sich empfiehlt, sich an diesem Standard zu orientieren

Beteiligte

Ausführend: Systemanalytiker, Projektleiter
Mitwirkend: Anwender

Vorgehen

Die Systemdokumentation wird in der Analyse-Phase angelegt und in den nächsten Phasen kontinuierlich ausgebaut. In Anlehnung an den „IEEE STD 830-1993“ Standard schlagen wir das in Abbildung 9 dargestellte Gliederung, vor. Das Dokument besteht aus einem Einleitungsteil, in dem die Projektziele und organisatorischen Aspekte behandelt werden. Im Hauptteil wird das zu realisierende Anwendungssystem beschrieben, indem z.B. die Zielumgebung, die Schnittstellen zu Fremdsystemen oder die Systemarchitektur dokumentiert werden. Im Schlußteil befindet sich der Index und der Anhang. Nicht alle Abschnitte können bereits während der Analyse-Phase ausgefüllt werden, wie z.B. Abschnitt 5.2, wo der Ausgang des Akzeptanztests beschreiben wird. Das Dokument muß nach und nach vervollständigt und verfeinert werden.

1 Einleitung
1.1 Zielsetzung
1.2 Scope
1.3 Definitionen, Akronyme und Abkürzungen
1.4 Referenzen
1.5 Überblick

2 Allgemeine Beschreibung
2.1 Produktumgebung
2.2 Systemschnittstellen
2.2.1 Benutzerschnittstellen
2.2.2 Hardwareschnittstellen
2.2.3 Softwareschnittstellen
2.2.4 Kommunikationsschnittstellen
2.2.5 Speicher-Restriktionen
2.2.6 Operationen
2.2.7 Standort-abhängige Anpassungen
2.3 Produktfunktionen
2.4 Restriktionen
2.5 Annahmen und Abhängigkeiten
2.6 Benutzereigenschaften

3 Anforderungen
3.1 Funktionale Anforderungen
3.2 Nicht-funktionale Anforderungen

4 Bausteinstruktur
4.1 Komponentenstruktur
4.2 Subsystemstruktur
4.3 Systemarchitektur

5 Integration und Test
5.1 Integrationsstrategie
5.2 Integrationsbericht
5.3 Testbericht
5.4 Akzeptanzbericht
5.5 Inbetriebnahmebericht

6 Einsatz
6.1 Stabilisierungsbericht
6.2 Optimierungsbericht
6.3 Anpassungsbericht
6.4 Erweiterungsbericht

7 Richtlinien
7.1 Projekt-Richtlinien
7.2 Programmier-Richtlinien

8 Anhang
9 Index

Abb. 9: Gliederung für die Systemdokumentation

Ergebnisse

Die Systemdokumentation sollte angelegt und soweit wie möglich ausgefüllt werden. Dabei sind die in den vorherigen Schritten produzierten Ergebnisse einzubinden.

2.1.2 Entwurf

Ziel der Entwurfs-Phase ist es, eine fachliche Lösung für das geplante Anwendungssystem zu entwickeln. Ausgangspunkt sind die in der Analyse-Phase ermittelten funktionalen und nicht-funktionalen Anforderungen. Diese werden zur Definition von Komponenten genutzt, um das System zu strukturieren (Abb. 10).

Deren genaue Kenntnis ist eine weitere notwendige Voraussetzung für die Systemarchitektur. Ausgehend von der Zielumgebung und der definierten Integrationsstrategie wird die Systemarchitektur entworfen. Parallel zu den genannten Aktivitätstypen muß die Systemdokumentation weiter fortgeschrieben werden.

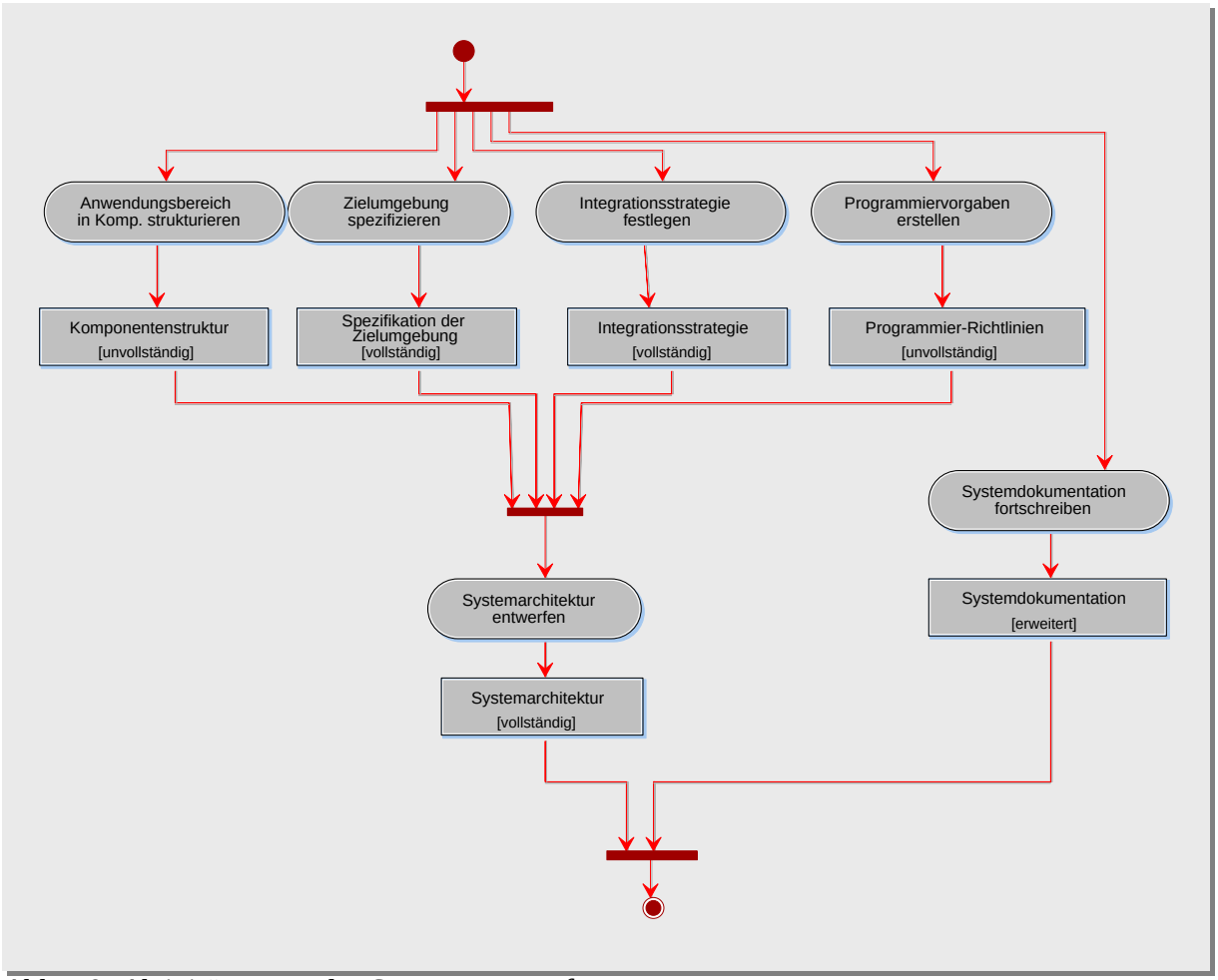


Abb. 10: Aktivitätstypen des System-Entwurfs

2.1.2.1 Anwendungsbereich in Komponenten strukturieren

Zweck

Damit die Komplexität der fachlichen Lösung beherrschbar bleibt, werden größere Modelleinheiten für die Klassen des Anwendungsbereichs gebildet. In der UML heißen diese

Pakete (*Packages*). Ein Paket enthält jeweils eine Gruppe von Klassen, die semantisch zusammengehören, aber durchaus Beziehungen zu anderen Gruppen von Klassen haben können. Wir sprechen in diesem Zusammenhang von Komponenten und behalten in den nächsten Abschnitten diesen Sprachgebrauch bei (Komponente = Paket). Die Strukturierung des Systems in Komponenten hat neben der Reduktion von Komplexität organisatorische Vorteile, wie z.B. Handhabbarkeit der großen Anzahl von Klassen, Unterstützung der Wiederverwendung oder Erleichterung der Projektplanung. Mit der Identifikation einer Komponente wird auch sein Entwicklungszyklus angestoßen.

Beteiligte

Ausführend: Systemanalytiker
Mitwirkend: Anwendungsbereichsexperte

Vorgehen

Folgende drei Schritte sind zu durchlaufen:

- Komponenten definieren und abgrenzen,
- Komponenten beschreiben,
- Beziehungen definieren,
- Komponenten-Gliederung prüfen.

Komponenten definieren und abgrenzen

Intern bestehen Komponenten aus einer Menge von Klassen, die eine definierte Leistung erbringen. Bevor die interne Klassenstruktur von Komponenten gestaltet werden kann, müssen zunächst Komponenten identifiziert und fachlich gegeneinander abgegrenzt werden. Dazu ist eine Blackbox-Sicht der Komponente sehr hilfreich. Denn so können die entscheidenden zwei Fragen beantwortet werden:

- Welche funktionalen und nicht-funktionalen Anforderungen können mit der Komponente abgewickelt werden?
- Welche Schnittstellen in Form von öffentlichen Klassen bietet die Komponente?

Der Kern der Aktivität dieses Typs besteht in der Ermittlung von Anforderungen, die in einem engen fachlichen Kontext stehen. Sind solche Anforderungen identifiziert, so kann von ihnen zu Komponenten abstrahiert werden. Bei diesem Abstraktionsschritt wird der fachliche Gegenstand in Form von Anwendungsfällen (funktionale Anforderungen) bzw. Tabellen (nicht-funktionale Anforderungen) in einen technischen Gegenstand in Form von Komponenten abgebildet. Ein Beispiel für diese Vorgehensweise für die funktionalen Anforderungen zeigt die nächste Abbildung.

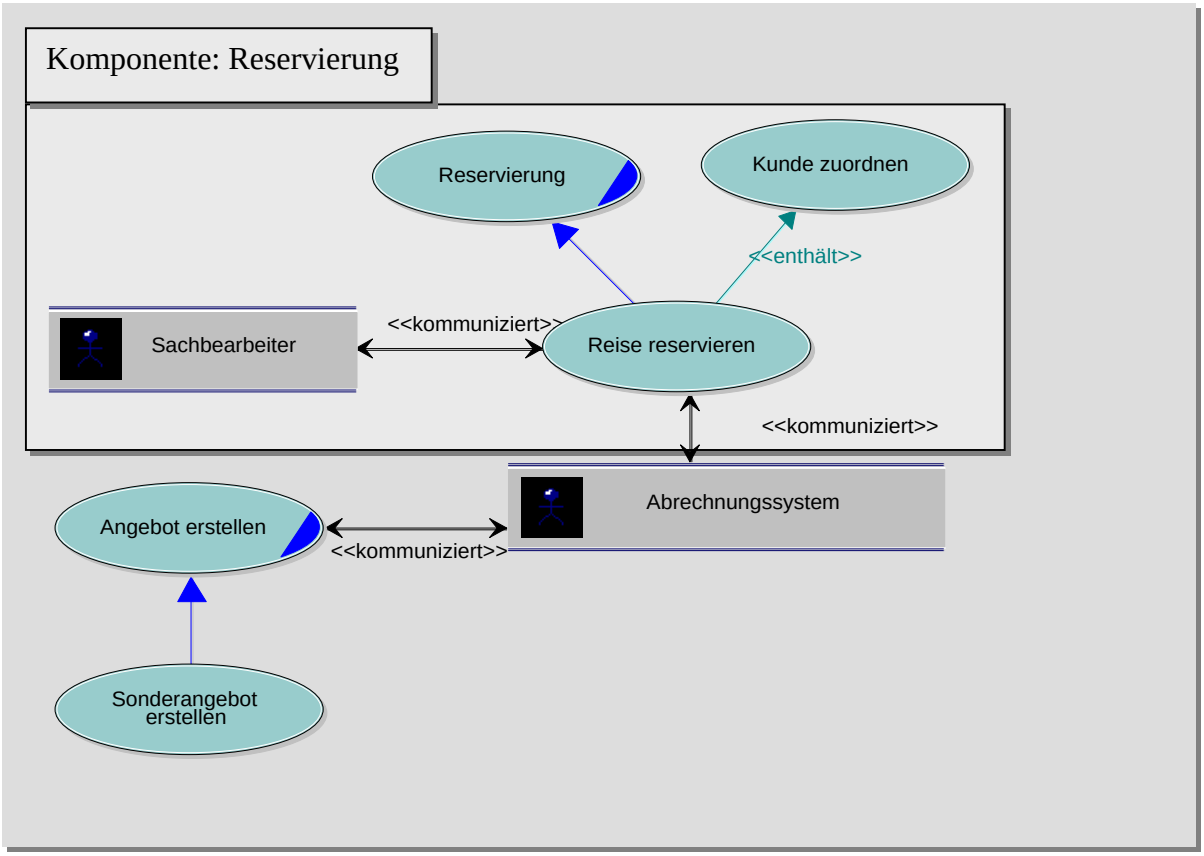


Abb. 11: Komponenten definieren

Wenn später die Klassen identifiziert werden, kann auch bottom up vorgegangen werden, d.h. ausgehend von den gefundenen Klassen werden Komponenten definiert. Sollte ein Datenmodell vorliegen, so kann es ebenfalls als Grundlage zur Definition von Komponenten verwendet werden. Nachdem die Komponenten soweit wie möglich identifiziert worden sind, müssen die gegenseitigen Abhängigkeiten modelliert werden. Die Package-Diagramme von UML können verwendet werden, um die gefundenen Komponenten und ihre Abhängigkeiten zu beschreiben.

Komponenten beschreiben

Für die Beschreibung von Komponenten kann das folgende Muster verwendet werden, das im Verlauf des Entwicklungsprozesses vervollständigt werden muß:

Bezeichnung	⇨ Geben Sie hier die Bezeichnung der Komponente an. [**Hier** klicken und Text eingeben]
Art	⇨ Klassifizieren Sie die Komponente: Handelt es sich um ein Komponente mit Anwendungsklassen? Ist es eine Klassenbibliothek, ein Framework oder eine gekaufte Komponente? Alternativ können Sie dazu auch Stereotypen verwenden. [**Hier** klicken und Text eingeben]
Quelle	⇨ Wenn es sich um eine Klassenbibliothek, ein Framework oder eine gekaufte Komponente handelt, geben Sie hier Hersteller, Autor, Version, Datum etc. an. Alternativ können Sie dazu auch benutzerdefinierte Eigenschaften verwenden. [**Hier** klicken und Text eingeben]
Zweck	⇨ Stellen Sie kurz dar, wozu die Komponente im Kontext der gesamten Anwendung dienen wird, und begründen Sie Ihre Entwurfsentscheidung. Stellen Sie dabei einen

	Bezug zu den Anwendungsfällen her. [**Hier** klicken und Text eingeben]
Leistung	⇨ Listen Sie die wesentlichen Leistungen auf, die die Komponente aus äußerer Sicht erbringen soll. Ergänzen Sie diese Beschreibung im Laufe der folgenden Entwicklungsschritte um die innere Sicht, also die Angabe der Klassen, die die Leistungen tatsächlich liefern. [**Hier** klicken und Text eingeben]
Schnittstellen	⇨ Fügen Sie hier eine kommentierte Liste der öffentlichen Klassen ein. [**Hier** klicken und Text eingeben]
Abhängigkeiten	⇨ Beschreiben und begründen Sie hier im Laufe der folgenden Entwicklungsschritte Abhängigkeiten zu anderen Komponenten. [**Hier** klicken und Text eingeben]
Verwendungs-hinweise	⇨ Hier können Sie Hinweise folgender Art hinterlegen: „Letzte Änderung erfolgt am ...“, „Neue Version in Vorbereitung/angekündigt für ...“, „Bekannte Fehler in ...“, „Achtung: Wartung vom Hersteller eingestellt“, „Erweiterungen vorgesehen für ...“, etc. [**Hier** klicken und Text eingeben]

Abb. 12: Beschreibungsschema für Komponenten /objectiF 4.5, microTOOL GmbH/

Beziehungen definieren

Zwischen zwei Komponenten kann eine Assoziation, Aggregation oder Generalisierung definiert werden. Diese Beziehungstypen haben die gleiche Bedeutung wie die entsprechenden Klassenbeziehungen. Es besteht eine *Assoziation* zwischen zwei Komponenten, wenn mindestens eine Klasse A der ersten Komponente mit einer Klasse B aus der zweiten Komponente in einer Assoziations- oder Aggregationsbeziehung ist. Wenn eine Komponente A weitere Komponenten umfaßt, so spricht man von einer *Aggregation*. Die beteiligten Komponenten sind nicht gleichrangig. Die Komponente A repräsentiert das Ganze und die anderen die Teile. Eine Komponente kann als *Spezialisierung* einer anderen – allgemeineren – Komponente angesehen werden, wenn ihre Schnittstelle der der allgemeineren Komponenten entspricht.

Zur Definition der genannten Beziehungen ist die Klassenstruktur der Komponenten notwendig. Die Klassenstruktur einer Komponente liegt aber erst vor, wenn sie entworfen ist. Mit Hilfe der folgenden Regel können trotzdem erste Beziehungen zwischen Komponenten angelegt werden:

- Besteht zwischen einem Anwendungsfall einer Komponente A und einem Anwendungsfall einer zweiten Komponente B eine Beziehung, so liegt zwischen A und B mindestens eine Assoziation vor.

Die angelegten Beziehungen zwischen den identifizierten Komponenten müssen im Verlauf des Entwicklungsprozesses verfeinert werden.

Komponenten-Gliederung prüfen

Hier folgen noch einige Kriterien, die zur Bewertung von Komponenten und zur Verbesserung der Komponenten-Gliederung genutzt werden können /Balzert 1995/.

- Komponenten sollten eine hohe interne Bindung haben, d.h. sie sollten keine lose Sammlung von Klassen sein, sondern aus Klassen mit hohem fachlichen Kontext bestehen.
- Komponenten sollten zur Reduktion von Komplexität verhelfen.
- Komponenten sollte schmale Schnittstellen zu anderen besetzen - hierbei spricht man von einer losen Kopplung - damit die Wartung und die Wiederverwendung erleichtert wird.
- Zyklische Abhängigkeiten wie in Abb. 13 sollten wenn möglich vermieden werden.

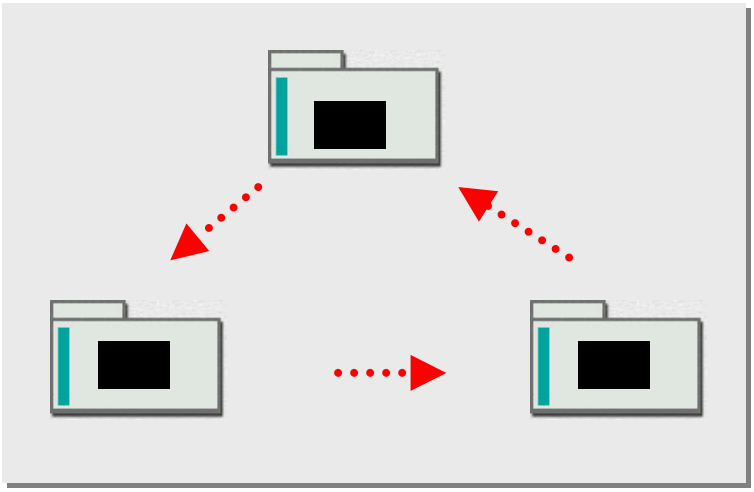


Abb. 13: Zyklische Abhängigkeit von Komponenten

Die obige Abbildung beschreibt drei voneinander abhängige Komponenten, die in der Bearbeitung sind. Komponente A benötigt B, damit sie weiter verarbeitet werden kann. Komponente B benötigt wiederum Komponente C. Da die Komponente C auf A angewiesen ist, entsteht ein Deadlock. Solche Deadlocks entstehen durch zyklische Abhängigkeiten und erschweren die Wartung.

Es gibt zwei Techniken zum Auflösen von zyklischen Abhängigkeiten, nämlich „Auflösung durch Auslagerung“ und „Auflösung durch Abhängigkeitsumkehr“. Bei der ersten Technik werden die Klassen, von denen mehrere Komponenten abhängen, in eine neue Komponente ausgelagert und der Zyklus wird gebrochen. Im zweiten Fall werden Interfaces verwendet, um eine Abhängigkeit umzukehren. Sollte also eine Komponente „A“ von einer Klasse „c“ einer zweiten Komponente „B“ abhängen, so wird in „A“ ein Interface zu c erstellt, das von der Komponente „B“ implementiert werden muß. Damit wird die Abhängigkeit umgedreht und der Zyklus gebrochen.

Ergebnisse

Die einzelnen Komponenten werden mit dem Beschreibungsschema in Abb. 12 vollständig beschrieben. Package-Diagramme sollten erstellt werden, um die Abhängigkeiten der Komponenten zu modellieren.

Hinweise und Tipps

- Nicht-funktionale Anforderungen sollen die Komponenten-Gliederung nicht beeinflussen, denn Qualitätsmerkmale oder Benutzeroberflächen sind keine Gliederungskriterien.

- Komponenten sollten idealer Weise nicht mehr als 20 bis 30 Klassen umfassen. Gegebenenfalls sollte die Möglichkeit des Schachtelns von Komponenten genutzt werden.

2.1.2.2 Zielumgebung spezifizieren

Zweck

Bevor die Systemarchitektur entworfen wird, sollte die Zielumgebung des geplanten Anwendungssystems vollständig untersucht werden. Das ist der Zweck dieses Aktivitätstyps, in der die Elemente der Zielumgebung, wie z.B. Hardware, Netzwerk, Betriebssysteme, festgelegt und spezifiziert werden.

Beteiligte

Ausführend: Projektleiter
Beteiligt: Systemarchitekt

Vorgehen

Die Grundlage für die Spezifikation der Zielumgebung bilden die bereits analysierten nicht-funktionalen Anforderungen. Folgende Schritte sind zu durchlaufen:

- Elemente der Zielumgebung ermitteln und dokumentieren,
- Spezifikation der Zielumgebung prüfen.

Elemente der Zielumgebung ermitteln und dokumentieren

Beim Untersuchen der Zielumgebung sollte mit den Hardware-Plattformen und Betriebssystemen begonnen werden. Weiterhin sind oft das Netzwerk und die entsprechende Software für die Spezifikation der Zielumgebung von Interesse. Bei verteilten Anwendungen muß geklärt werden, auf welcher technischen Grundlage die Verteilung von Objekten realisiert werden soll. Falls ein Datenbanksystem für das Anwendungssystem vorgesehen ist, muß geklärt werden, welche Datenbanktechnik verwendet werden soll. Die ermittelten Elemente der Zielumgebung können mit dem folgenden Muster dokumentiert werden.

1	Hardware-Plattformen
2	Netzwerk/Netzwerkprotokoll
3	Object Request Broker
4	Datenbanksysteme
5	Laufzeitsysteme
6	Sonstige systemnahe Software

Abb. 14: Beschreibungsschema für Zielumgebung

Spezifikation der Zielumgebung prüfen

- Anhand der operationalen Abhängigkeiten der Elemente sollte festgestellt werden, ob die ermittelten Elemente vollständig sind.

- Es sollte überprüft werden, ob die spezifizierte Zielumgebung den nicht-funktionalen Anforderungen genügen wird.

Ergebnisse

Die Spezifikation der Zielumgebung wird mit dem Beschreibungsschema in der Abbildung 14 festgehalten werden.

2.1.2.3 Integrationsstrategie festlegen

Zweck

Im Rahmen dieses Aktivitätstyps wird eine Strategie zur Integration des Systems aus den Komponenten bzw. Subsystemen ausgearbeitet.

Beteiligte

Ausführend: Softwaredesigner, Systemarchitekt
Mitwirkend: Softwareentwickler, Qualitätsprüfer

Vorgehen

Die Systemintegration ist in der Regel ein Bottom-up-Prozeß, der von den Abhängigkeiten der Komponenten bzw. Subsysteme stark beeinflußt wird. Auf der Komponenten-Ebene werden wir unterschiedliche Integrationsstrategien, wie z.B. *Bottom-up*, *Top-down*, *Inside-out*, *Hardest-first*, *Outside-in*, *Big-bang*, *Geschäftsprozeß-orientiert*, *Funktions-orientiert*, *Verfügbarkeits-orientiert*, vorstellen. Die Auswahl einer Strategie hängt vom zu realisierenden Anwendungssystem ab. Oft ist aber eine Kombination solcher Strategien sinnvoll.

Ergebnisse

Die festgelegte Integrationsstrategie sollte beschrieben und begründet werden.

2.1.2.4 Programmiervorgaben erstellen

Zweck

Damit eine von jeweiligen Autor unabhängige Wartbarkeit und eine hohe Qualität des Quellcodes gewährleistet werden kann, sollten sich Softwareentwickler an einheitlichen Richtlinien orientieren. Falls solch ein Regelwerk als unternehmensweiter Standard nicht vorliegt, so sind entsprechende Programmierrichtlinien, die z.B. Namenskonventionen enthalten, festzulegen.

Beteiligte

Ausführend: Projektleiter, Softwareentwickler
Mitwirkend: Softwaredesigner

Vorgehen

Programmierrichtlinien helfen die Wartung und Pflege zu erleichtern. Dabei handelt es sich um Richtlinien, bei denen Verstöße durch den Compiler nicht festgestellt werden können, jedoch z.B. durch Code-Inspektionen. Beispiele sind:

- Richtlinien für Bezeichner,
- Richtlinien für die Formatierung,
- Richtlinien für Methodenaufrufe,
- Richtlinien für Fehlerbehandlung.

Falls ein unternehmensweiter Standard vorliegt oder auf Richtlinien früherer Projekte zurückgegriffen werden kann, so sind diese auf die Erfordernisse des aktuellen Projekts anzupassen. Dabei ist zu berücksichtigen, welche der Richtlinien durchsetzbar waren und welche als Standard nicht akzeptiert wurden.

Ein übersichtliches Regelwerk trägt zur Akzeptanz durch das Projektteam bei, während ein 500 Seiten langes Dokument abschreckend wirkt und nicht eingehalten wird. Damit die Richtlinien auch angenommen werden, sollten diese mit den beteiligten Personen kritisch diskutiert werden. Außerdem sollte die Möglichkeit vorgesehen werden, die Richtlinien im Verlauf der Implementierung zu korrigieren und zu erweitern, so daß die Richtlinien für die nächsten Zyklen und Bausteine verbessert werden können.

Ergebnisse

Programmierrichtlinien können wie folgt tabellarisch dokumentiert werden.

Bezeichnung	Ein eindeutiger Bezeichner zur Identifikation der Richtlinie.
Thema	Auf welches Entwurfselement, sprachenspezifische Konstrukt oder Implementierungsziel bezieht sich die Regel (z.B. Vererbung, Konstruktor/Destruktor)?
Regel	Hier sollten die Programmierregeln in knapper Form beschrieben werden.
Begründung	Die Regel sollte hier kurz motiviert werden.
Beispiel	Zum besseren Verständnis sollte ein Beispiel hinzugefügt werden.
Hinweise	Hier sollten spezielle Aspekte, wie z.B. der Zusammenhang mit anderen Regeln, diskutiert werden.

Abb. 15: Beschreibungsschema für Programmierrichtlinien

Hinweise und Tipps

Wenn auf vorhandene Richtlinien nicht zurückgegriffen werden kann, so bieten sich externe Quellen wie z.B. /Cline, Lomow 1995/ und /Sanchez, Canton 1998/ als Anregung an.

2.1.2.5 Systemarchitektur entwerfen

Zweck

Ziel dieses Aktivitätstyps ist es, eine Systemarchitektur zu entwerfen, um die fachliche Lösung in der vorhandenen Zielumgebung technisch zu realisieren. Ein geeignetes Architekturkonzept auszuarbeiten, ist eine schwierige und folgenreiche Entscheidung im Prozeß der Software-Entwicklung. Denn von der Architektur hängt es z.B. ab, ob das System die Wiederverwendbarkeit, Erweiterbarkeit und Wartbarkeit gut unterstützt. Damit ist die Wahl einer geeigneten Systemarchitektur eine langfristige und weitreichende Entscheidung.

Beteiligte

Ausführend: Systemarchitekt
Mitwirkend: Systemdesigner, Projektleiter

Vorgehen

Nach folgendem Schema sollte bei der Entwicklung einer Systemarchitektur vorgegangen werden:

- Architekturziele festlegen,
- Systemgliederung entwickeln:
  - Schichten spezifizieren und konstruieren,
  - Komponenten in Schichten gliedern,
  - Schnittstellen der Schichten spezifizieren,
  - Fehlerbehandlungsstrategie entwickeln,
- Systemarchitektur prüfen.

Architekturziele festlegen

Auf der Suche nach der geeigneten Systemarchitektur sollte zunächst definiert werden, welche Ziele mit der Systemarchitektur erreicht werden sollen. Für das Anwendungssystem anzustrebende Architekturziele, wie z.B. Erweiterbarkeit oder Skalierbarkeit, sollten gesammelt, dokumentiert und bewertet werden.

Systemgliederung entwickeln

Die Systemgliederung in Komponenten hängt von den festgelegten Architekturzielen ab. Welche Gliederung aber die richtige ist, kann leider nicht allgemein beantwortet werden. Im folgenden stellen wir kurz einige Architekturmuster zur Gliederung eines Systems vor, die Lösungen für spezielle Architekturziele bieten. Details hierzu können in /Buschmann 1998/ nachgelesen werden.

- Pipes-and-Filters-Muster
  Bei diesem Ansatz werden die Verarbeitungsschritte in Filtern (*Filters*) gekapselt und die Daten durch Kanäle (*Pipes*) weitergereicht. Durch die Kombinationen von Filtern wird ein System realisiert.

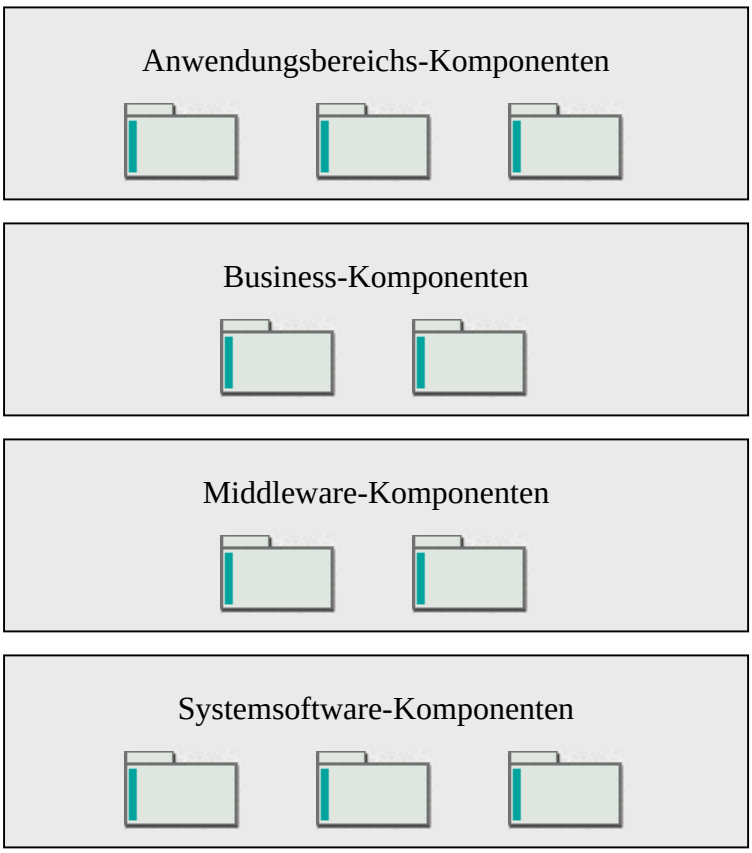
- Blackboard-Muster
  In diesem Architekturkonzept teilen sich mehrere Komponenten ihr Wissen, um nicht-deterministische Probleme zu lösen.

- Broker-Muster
  Dieser Ansatz bietet eine Lösung für verteilte Systeme und beschreibt eine Architektur aus entkoppelten Komponenten, die über eine Vermittlerkomponente (*Broker*) miteinander interagieren.

- Model-View-Controller-Muster
  Bei diesem Architekturvorschlag für interaktive Systeme werden die Komponenten in drei Typen aufgeteilt:
  1) Model-Komponenten, die die Kernfunktionalität und Datenverarbeitung abbilden,
  2) View-Komponenten, die die Benutzeroberflächen präsentieren,
  3) Controller-Komponenten, die die Benutzereingaben behandeln.

- Micro-Kernel-Muster
  Es bietet eine hohe Anpassungsfähigkeit, in dem die Kernfunktionalität abgespalten wird und spezifische Funktionen hiervon abgeleitet werden.

Wie soll jedoch vorgegangen werden, wenn keines der oben genannten Architektur-Muster verwendet werden kann? Erfahrungen aus der Praxis haben gezeigt, daß die Orientierung an einer Schichtenarchitektur (Abb. 16) in den meisten Fällen eine gute Ausgangsbasis für die Systemgliederung bietet, die sich sehr gut bewährt hat.



Komponenten, die dem Endanwender Anwendungsfälle realisieren.

Komponenten, die allgemein gültige fachliche Leistungen zur Wiederverwendung anbieten.

Komponenten, die von der Plattform unabhängige Technologieleistungen anbieten.

Komponenten, die von der Plattform abhängige Systemsoftware anbieten.

Abb. 16: Schema für eine Schichtenarchitektur /Jacobson 1997/

Das Architekturschema aus Abbildung 16 geht auf Jacobson zurück und bildet auch die Grundlage für moderne Architekturstandards wie J2EE (Java 2 Plattform Enterprise Edition). Die Komponenten einer Schicht können direkt miteinander interagieren, während die Kommunikation zwischen zwei Schichten nach dem *Pull-Prinzip* erfolgen sollte. Das bedeutet, daß eine höhere Schicht Leistungen aus einer darunter liegenden Schicht in Anspruch nimmt, aber nicht umgekehrt (*Push-Prinzip*).

In der *Anwendungsbereichs-Komponenten* Schicht befinden sich Komponenten, die dem Endanwender jeweils eine Menge eng zusammengehöriger Anwendungsfälle anbieten. Der Zweck der *Business-Komponenten* Schicht ist es, Wissen über fachliche Zusammenhänge systematisch wiederzuverwenden. Die Komponenten dieser Schicht realisieren oft selbst Anwendungsfälle genereller Art und bieten dem Entwickler von Anwendungsbereichs-Komponenten allgemein gültige fachliche Leistungen an. Die Schicht für die *Middleware-Komponenten* bietet von der Plattform unabhängige Technologieleistungen, wie z.B. Transaktionsmanagement oder Datenbankanbindungs-Schnittstellen, an. Plattformabhängige Systemsoftware, wie z.B. Betriebssysteme, Netzwerke, Datenbanksysteme oder spezifische Hardwareschnittstellen, werden in der Schicht für *Systemsoftware-Komponenten* gekapselt.

In Anlehnung an /Buschmann 1998/ beschreiben wir im folgenden die Entwurfsschritte für das vorgestellte Architekturschema:

Schichten spezifizieren und konstruieren:

Orientiert an dem Abstraktionskriterium der Schichten - d.h. je tiefer eine Schicht angesiedelt ist, umso allgemein gültiger und weniger anwendungsspezifisch ist sie – sollten für das Anwendungssystem relevante Schichten definiert und kurz beschrieben werden.

Komponenten in Schichten gliedern:

Aus den Leistungen, die eine Schicht anbieten soll, können ihr die entsprechenden Komponenten zugeordnet werden. Nach diesem Prinzip können nach und nach alle Komponenten einer Schicht zugeordnet werden.

Schnittstellen der Schichten spezifizieren:

In diesem Schritt sollen die definierten Schichten so weit beschrieben werden, daß sie weitgehend unabhängig voneinander konstruiert werden können. Daher liegt es nahe, einen *Black-Box-Ansatz* zu verfolgen und das Innere einer Schicht nach außen zu verbergen. Hierzu bietet sich das Fassadenmuster /Gamma 1995/ an, bei dem die Komponenten einer Schicht über eine Fassadenklasse abgeschirmt werden.

Fehlerbehandlungsstrategie entwickeln:

Bei einer Schichtenarchitektur muß geklärt werden, wo ein aufgetretener Fehler behandelt werden soll. Eine Faustregel besagt, daß ein Fehler nach Möglichkeit in der Schicht behandelt werden soll, in der er aufgetreten ist. Damit wird ein aufwendiges Durchreichen von Fehlern in höhere Schichten vermieden. Oft können nicht alle Fehler einer Schicht auch in ihr behandelt werden, so daß aufgetretene Fehler in einer höheren Schicht abgefangen werden müssen. Um zu vermeiden, daß höhere Schichten von hochgereichten Fehlern „überschwemmt“ werden, empfiehlt es sich, Fehler zu Fehlergruppen zu abstrahieren und eine Fehlerbehandlung pro Fehlergruppe zu implementieren.

Systemarchitektur prüfen

Folgende Merkmale der Architekturlösung sollten überprüft werden:

- Sind die Architekturziele wirklich aus den funktionalen und nicht-funktionalen Anforderungen sowie den Rahmenbedingungen der Zielumgebungen abgeleitet worden?
- Werden die definierten Architekturziele auch wirklich mit der angegebenen Lösung erreicht?
- Sind so viel Schichten wie nötig und so wenige wie möglich definiert worden? Denn zu viele Schichten erhöhen den organisatorischen Aufwand und zu wenige erschweren die Wartung.
- Ist die Kopplung der Schichten einseitig, d.h. eine Schicht benötigt nur Leistungen aus niederen Schichten? Sind die Schnittstellen der Schichten so schmal wie möglich?
- Haben die Komponenten einer Schicht auch eine hohe Kohäsion, d.h. hohe interne Bindung?
- Sind zyklische Abhängigkeiten, wie z.B. in Abbildung 13 beschrieben, auch wirklich vermieden worden?

Ergebnisse

Die zentralen Architekturentscheidungen sollten in einem Dokument zusammengefaßt werden. Die folgende Gliederung kann dabei hilfreich sein:

1	Architekturziele
2	Architektur: Schichten des Systemaufbaus (Leistungen, Schnittstellen, Komponenten)
3	Kommunikation zwischen den Schichten
4	Fehlerbehandlungskonzept
5	Sonstige Bemerkungen

Abb. 17: Dokumentation des Architekturkonzepts

Hinweise und Tipps

Weitere Architekturmuster können in /Martin 1998/ gefunden werden. Bei auf CORBA-basierenden Anwendungssystemen können in /Mowbray 1997/ nützliche Anregungen, wie z.B. die Integration von Altsystemen, gefunden werden.

Folgende Prinzipien helfen, eine optimale Systemarchitektur hinsichtlich der Wiederverwendung und Wartbarkeit zu entwickeln /Martin 1996/.

- Das Prinzip der Äquivalenz von Wiederverwendung und Freigabe (*reuse/release equivalence principle*):
  Eine Freigabeeinheit entspricht einer Einheit zur Wiederverwendung. Dieses Prinzip bildet die Grundlage für eine erfolgreiche und risikoarme Wiederverwendung von Komponenten.

- Das Prinzip der gemeinsamen Wiederverwendung (*common reuse principle*):

Eine Komponente wird samt ihrer Klassen wiederverwendet. Damit wird die Zuordnung von Klassen zu Komponenten gegebenenfalls erleichtert.

- Das Prinzip der gemeinsamen Abgeschlossenheit (*common closure principle*):
  Die Klassen einer Komponenten sollten gegen dieselbe Art von Änderungen abgeschlossen sein. Damit führen Änderungen zu einer neuen Version, wodurch die Wartbarkeit erleichtert wird.

- Das Prinzip der azyklischen Abhängigkeiten (*acyclic dependencies principle*):
  Die Abhängigkeiten von Komponenten müssen sich als gerichtete, azyklische Graphen darstellen lassen. Mit diesem Prinzip wird die Wartbarkeit eines Systems erleichtert (siehe Abbildung 13).

- Das Prinzip der stabilen Abstraktionen (*stable abstractions principle*):
  Komponenten mit maximaler Stabilität sollten abstrakt sein, und instabile Komponenten sollten möglichst konkret sein. Für das Messen der Stabilität und Abstraktion von Komponenten sind bereits entsprechende Metriken bekannt.

- Das Prinzip der stabilen Abhängigkeiten (*stable dependencies principle*):
  Eine Komponente sollte immer von Komponenten abhängen, die stabiler sind. Auch mit diesem Prinzip wird eine hohe Wartbarkeit gewährleistet.

2.1.3 Implementierung

Gegenstand der Implementierung auf der Systemebene ist im engeren Sinne die Systemintegration und der Systemtest. Im einzelnen sind die in der nächsten Abbildung dargestellten Aktivitäten durchzuführen. Die Systemdokumentation muß weiterhin gepflegt und verfeinert werden (Abb. 18).

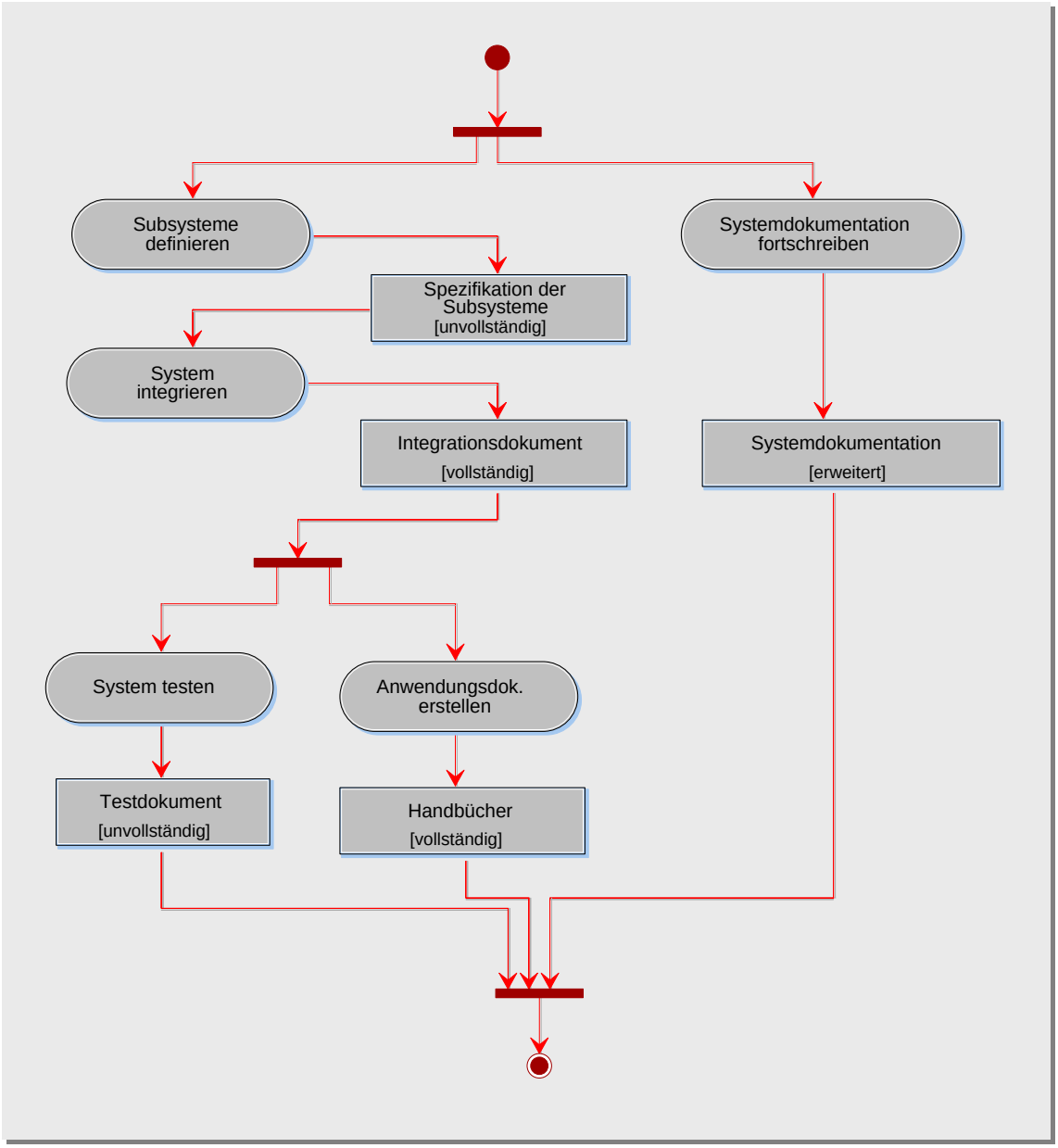


Abb. 18: Aktivitätstypen der System-Implementierung

2.1.3.1 Subsysteme definieren

Zweck

Ein Subsystem ist eine nicht-disjunkte Zusammenfassung von Klassen, die zu Test- oder Integrationszwecken gemeinsam zum Ablauf gebracht werden. Im Rahmen dieses Aktivitätstyps werden die für das Anwendungssystem benötigten Subsysteme definiert.

Beteiligte

Ausführend: Softwaredesigner, Systemarchitekt
Mitwirkend: Softwareentwickler, Qualitätsprüfer

Vorgehen

Die Definition von Subsystemen kann nach der folgenden Methode erfolgen /Balzert 1995/:

Ein Subsystem sollte eine logische Einheit bilden, das bedeutet:

- Es sollte nur Klassen enthalten die gemeinsam zum Ablauf gebracht werden sollen.
- Es sollte einen Themenbereich enthalten, der für sich allein betrachtet und verstanden werden kann.
- Es sollte Klassen enthalten, die logisch zusammengehören.
- Es sollte eine wohldefinierte Schnittstelle zu seiner Umgebung haben.
- Es sollte Vererbungsstrukturen nur in vertikaler Richtung schneiden, d.h. alle Oberklassen einer Unterklasse sollten in dem Subsystem enthalten sein.
- Die Durchtrennung von Aggregationen sollte vermieden werden.
- Ein Subsystem sollte so wenige Assoziationen enthalten wie möglich.

Die nächste Abbildung zeigt ein Beispiel für ein Subsystem, das Komponenten und Klassen über mehrere Schichten zusammenfaßt.

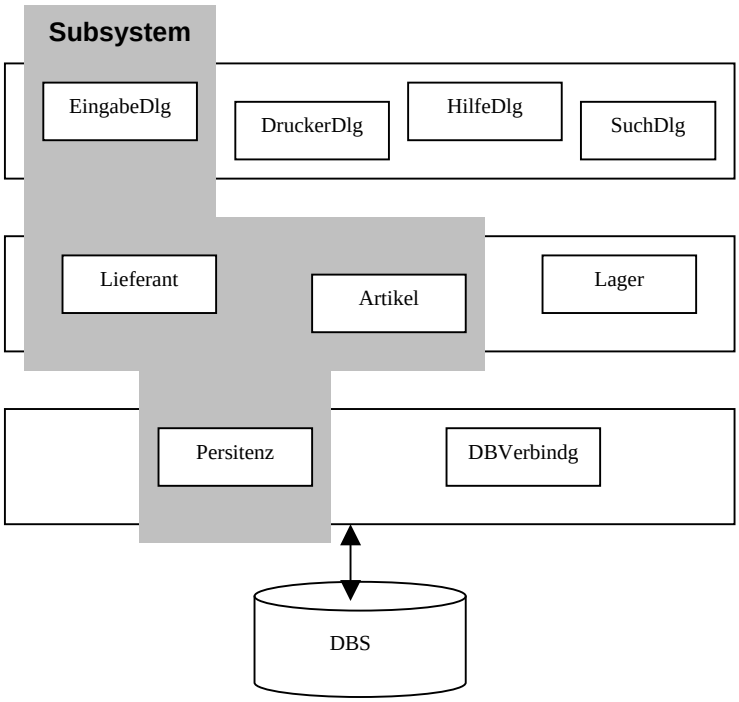


Abb. 19: Definition eines Subsystems

Ergebnisse

Die definierten Subsysteme sollten dokumentiert werden.

2.1.3.2 System integrieren

Zweck

Ein komponentenbasiertes Anwendungssystem setzt sich aus neuentwickelten, wiederverwendeten bzw. von fremder Seite hergestellten Komponenten zusammen. Ziel der Systemintegration ist es, diese Komponenten zu einem Anwendungssystem zusammenzuführen und in der Zielumgebung zum Laufen zu bringen. Hierbei ist der Nachweis zu erbringen, daß die Komponenten bzw. Subsysteme fehlerfrei über ihre Schnittstellen zusammenarbeiten.

Beteiligte

Ausführend: Softwareentwickler, Qualitätsprüfer
Mitwirkend: Softwaredesigner, Systemarchitekt

Vorgehen

Die Integration von Komponenten bzw. Subsystemen entspricht weitgehend dem Vorgehen zur Integration von Klassen zu Komponenten, das weiter unten beschrieben wird (siehe 2.2.3). Das bedeutet, daß folgende Schritte zu durchlaufen sind:

- Integration der Komponenten bzw. Subsysteme planen,
- Testfälle für den Integrationstest definieren,
- Integrationsumgebung erstellen,
- Integrationstest der Komponenten bzw. Subsysteme durchführen,
- Fehler analysieren und beheben.

Wie die Klassenintegration (siehe 2.2.3), ist die Integration von Komponenten bzw. Subsystemen in der Regel ein Bottom-up-Prozeß, der von den Abhängigkeiten der Komponenten bzw. Subsysteme gesteuert wird. Die in der Entwurfs-Phase erstellten Package-Diagramme können als Grundlage für die Planung des Integrationsprozesses herangezogen werden. Zur Dokumentation können die im Rahmen der UML definierten Komponenten- und Verteilungsdiagramme verwendet werden.

Ergebnisse

Für die Dokumentation von Testfällen, Testanweisung und den Testablauf können die Beschreibungsschemata für die Klassenintegration verwendet werden (siehe 2.2.3). Um die Abhängigkeiten zwischen den Komponenten bzw. Subsystemen zu veranschaulichen, sollten entsprechende Komponenten- und Verteilungsdiagramme angefertigt werden.

Hinweise und Tipps

Auf der Komponenten-Ebene werden eine Reihe von Integrationsstrategien, wie z.B. *Bottom-up*, *Top-down*, *Inside-out*, *Hardest-first*, *Outside-in*, *Big-bang*, *Geschäftsprozessorientiert*, *Funktionsorientiert*, *Verfügbarkeitsorientiert*, erläutert (siehe 2.2). Welche Integrations-

strategie oder ob eine Kombination sinnvoll ist, hängt vom vorliegenden Anwendungssystem bzw. der Abhängigkeit der Komponenten/Subsysteme ab und muß im Einzelfall geprüft werden.

2.1.3.3 System testen

Zweck

Mit dem Systemtest soll nachgewiesen werden, daß das Anwendungssystem in der Zielumgebung die Leistungen erbringt, die in der Analyse-Phase spezifiziert worden sind. Es werden eine Reihe von Black-Box-Tests ausgeführt, die zwar aus der Anwendersicht, aber noch ohne direkte Anwenderbeteiligung stattfinden. Mögliche zu überprüfende Qualitätsmerkmale sind:

- die funktionale Vollständigkeit bezüglich der definierten Anforderungen,
- die Leistungsfähigkeit bzgl. Antwortzeit, Durchsatz, Belastbarkeit, Robustheit, Korrektheit, Sicherheit und Interoperabilität,
- die Verständlichkeit und Bedienbarkeit,
- die Installations- und Recovery-Fähigkeit.

Beteiligte

Ausführend: Qualitätsprüfer
Mitwirkend: Softwareentwickler

Vorgehen

Folgende Schritte sind beim Systemtest zu durchlaufen:

- Systemtest planen,
- Testfälle definieren,
- Systemtest durchführen,
- Fehler analysieren und beheben.

Systemtest planen

Der Umfang und die Art des Systemtests hängen davon ab, welche Qualitätsmerkmale priorisiert werden. Daher sind zunächst die Qualitätsmerkmale zu definieren und zu Prioritäten zu bilden, von denen ausgehend die durchzuführenden Tests zu bestimmen sind. Die oben genannten Qualitätsmerkmale können durch folgende Tests nachgewiesen werden.

- *Funktionstest:* Ausgehend vom Anwendungsfallmodell werden Testfälle abgeleitet, die zum Nachweis der funktionalen Vollständigkeit und Korrektheit des Systems genutzt werden können.
- *Leistungstest:* Aus den nicht-funktionalen Anforderungen ergeben sich die entsprechenden Testfälle für die Leistungsfähigkeit. Zum Leistungstest gehören:
  - *Zeittest:* Die Einhaltung von Zeitrestriktionen wird überprüft.
  - *Massentest:* Es wird sichergestellt, daß das System auch große Datenmengen verarbeiten kann.

  - *Lasttest:* Das System wird über einen längeren Zeitraum unter Spitzenbelastung (aber keine Überlastung) geprüft, um die Zuverlässigkeit zu überprüfen.
  - *Stresstest:* Für kurze Zeit wird das System überbelastet, um die Robustheit und Fehlertoleranz zu ermitteln.
  - *Sicherheitstest:* Es wird überprüft, ob die Sicherheitsanforderungen erfüllt werden.
  - *Umgebungstest:* Das System wird in der Zielumgebung unter den dort gegebenen technischen und physikalischen Bedingungen überprüft.
  - *Konfigurationstests:* Es wird überprüft, ob das System in allen geforderten Hardware- und Softwarekonfigurationen ablauffähig ist.
- *Benutzbarkeitstest:* Es wird überprüft, ob die Systembedienung intuitiv erfolgen kann. Der Anwender sollte nicht über- bzw. unterfordert werden. Seine fachliche Terminologie sollte auf den Benutzeroberflächen korrekt wiedergegeben worden sein.
- *Installations- und Wiederanlauftest:* Es wird überprüft, ob das System in der Zielumgebung erfolgreich installiert werden kann und, ob sich das System im Falle eines Absturzes problemlos wieder in Betrieb nehmen lassen kann.

Testfälle definieren

Ausgehend vom Anwendungsfallmodell können Testfälle für den Funktionstest definiert werden, die nach dem weiter unten angegebenen Schema (siehe Komponenten-Ebene) beschrieben werden können. Aus den nicht-funktionalen Anforderungen können Testfälle für den Leistungstest ermittelt werden, die mit dem folgenden Schema dokumentiert werden können.

Testnummer	⇨ Da ein Testfall in der Regel mehrfach getestet wird, empfiehlt sich die Vergabe eines identifizierenden Merkmals pro Test. [**Hier** klicken und Text eingeben]
Testdatum	⇨ Geben sie hier an, wann der Test stattgefunden hat. [**Hier** klicken und Text eingeben]
Testteam	⇨ Führen Sie hier die Namen der am Test beteiligten Personen auf. [**Hier** klicken und Text eingeben]
ID	⇨ Vergeben Sie hier ein eindeutiges Merkmal zur Identifikation des Testfalls, denn pro Szenario kann es mehrere Testfälle geben. [**Hier** klicken und Text eingeben]
Gegenstand	⇨ Machen Sie deutlich, was der Gegenstand des Tests ist, indem Sie die entsprechende Testform kennzeichnen. ☐ Zeittest ☐ Massentest ☐ Lasttest ☐ Stresstest ☐ Sicherheitstest ☐ Umgebungstest ☐ Konfigurationstest ☐ Funktionstest
Zweck	⇨ Was ist der Zweck des Testfalls? Was soll konkret nachgewiesen werden? Welcher Anwendungsfall wird betrachtet? [**Hier** klicken und Text eingeben]
Spezifikation	⇨ Beschreiben Sie den Testfall ausführlich und nachvollziehbar.

	[**Hier** klicken und Text eingeben]	
Lfd. Nummer	⇨ Listen Sie hier alle Rahmenbedingungen auf, unter denen der Test ablaufen soll. [**Hier** klicken und Text eingeben]	
Testanweisung	**Erwartetes Ergebnis**	**Tatsächliches Ergebnis**
⇨ Beschreiben Sie hier jeden einzelnen Testschritt, den der Tester durchzuführen hat.	⇨ Führen Sie in dieser Liste pro Testschritt das erwartete Ergebnis.	⇨ Dokumentieren Sie das tatsächlich beobachtete Ergebnis jedes Testschritts.
[**Hier** klicken]	[**Hier** klicken und Text eingeben]	[**Hier** klicken und Text eingeben]
Hinweis	⇨ Sammeln Sie hier weitere Hinweise und Bemerkungen, die hilfreich sind, um den Testfall zu verstehen. [**Hier** klicken und Text eingeben]	
Zusammen-fassung	⇨ Geben Sie hier eine Zusammenfassung von Testverlauf und Ergebnissen. [**Hier** klicken und Text eingeben]	
Bewertung/ Empfehlung	☐ fehlerfreier Ablauf ☐ leichte Fehler ☐ schwere Fehler ☐ fatale Fehler	☐ Ergebnis akzeptieren, Test abschließen ☐ Korrektur ohne Testwiederholung ☐ Korrektur und Testwiederholung
Hinweise zur Fehlerbehebung	⇨ Sammeln Sie hier alle Hinweise und Bemerkungen, die bei der Ermittlung der Fehlersuche und bei der Fehlerbehebung von Nutzen sein können. [**Hier** klicken und Text eingeben]	

Abb. 20: Beschreibungsschema für Testfälle, Testanweisungen und Testprotokolle /Pressman 1997/

Das obige Schema kann auch für den Installations- und Wiederanlauftest verwendet werden.

Systemtest durchführen

Für die Durchführung des Systemtests muß in der Regel das Anwendungssystem in der Zielumgebung installiert werden, ohne den laufenden Betrieb zu stören. Für den Leistungstest müssen gegebenenfalls Vorkehrungen getroffen werden, wie z.B. die Bereitstellung von Massendaten. Solche Massendaten sollten durch Testdatengeneratoren automatisch erzeugt werden, oder es muß auf vorhandene Datenbestände zurückgegriffen werden. Der Testverlauf und die Ergebnisse sollten auf jeden Fall protokolliert werden.

Fehler analysieren und beheben

Die gefundenen Fehler sollten priorisiert werden und nach der Dringlichkeit behoben werden. Allerdings kommt nicht nur das entwickelte Anwendungssystem als mögliche Fehlerquelle in Frage, sondern auch die Hardware- und Softwareumgebung.

Ergebnisse

Die Testspezifikation (Testfälle und die Ergebnisse ihrer Durchführung) sollte mit dem vorgestellten Beschreibungsschema dokumentiert werden.

2.1.3.4 Anwendungsdokumentation erstellen

Damit das Anwendungssystem erfolgreich eingeführt werden kann, sind eine Reihe von Maßnahmen, wie z.B. „Anwender schulen“ oder „Infrastruktur beschaffen“, notwendig. Die Entwicklung und Umsetzung einer Einführungsstrategie ist aber in erster Linie die Aufgabe des Auftraggebers und nicht mehr als unmittelbarer Bestandteil des Softwareentwicklungsprozesses anzusehen. Das Projektteam kann aber die Einführung durch die Entwicklung einer Anwendungsdokumentation und durch technische Hilfestellung bei der Inbetriebnahme unterstützen.

Zweck

Dieser Aktivitätstyp umfaßt das systematische Sammeln und didaktische Aufbereiten aller Informationen, die der Anwender zum Bedienen, Administrieren und für die Fehlerdiagnose des Anwendungssystems benötigt. Diese Informationen werden in einem oder mehreren Handbüchern, in digitaler oder bedruckter Form, zusammengefaßt.

Beteiligte

Ausführend: technischer Autor
Beratend: Anwendungsbereichsexperte, Systemanalytiker, Softwaredesigner, Softwareentwickler

Vorgehen

Die Entwicklung der Anwenderdokumentation und der Software laufen weitgehend parallel. Denn bereits im Verlauf der Anforderungsanalyse sollten Informationen für die Anwenderdokumentation gesammelt werden. Neben der Recherche sind folgende Schritte zu durchlaufen:

- Zielgruppe analysieren,
- Anwenderdokumentation gliedern,
- Dokumentationsvorgaben erstellen,
- Anwenderdokumentation entwickeln.

Zielgruppe analysieren

Je besser die Qualifikation und Vorkenntnisse der unterschiedlichen Zielgruppen bekannt ist, desto besser kann die Dokumentation auf die Zielgruppen abgestimmt werden. Daher ist eine Analyse der Zielgruppen für die Entwicklung der Dokumentation Voraussetzung. Die unterschiedlichen Zielgruppen können durch Profile, wie z.B. Anfänger oder Experte, beschrieben werden.

Anwenderdokumentation gliedern

Für die Dokumentation können grundsätzlich zwei Gliederungsprinzipien kombiniert werden:

- *die Anleitung:* Der Anwender wird sowohl bei der täglichen Arbeit als auch in kritischen Situationen unterstützt. Oft wird eine Anleitung in Form eines Tutorials zum Selbststudium geschrieben und eignet sich sehr gut für Anfänger.
- *das Nachschlagewerk:* Es ist so organisiert, daß ein schneller Zugriff auf eine spezifische Information ermöglicht wird. Zu allen Systemfunktionen werden vollständige Informationen angegeben, die in der Regel für Experten gedacht sind.

Vorwort

⇨ Im Vorwort sollten Sie den Gegenstand und Zweck des Handbuches nennen. Die aktuelle Version des Handbuches und die Zielgruppe sollten beschrieben werden.

[**Hier** klicken und Text eingeben]

Inhaltsverzeichnis

⇨ Das Inhaltsverzeichnis soll klar und übersichtlich sein und die wesentlichen Teile des Handbuches deutlich werden lassen. Daher empfiehlt es sich nicht mehr als drei Gliederungsebenen zu benutzen.

[**Hier** klicken und Text eingeben]

1 Einleitung

⇨ Hier sollten Sie den Aufbau und den Umgang mit dem Handbuch beschreiben.

[**Hier** klicken und Text eingeben]

2 Inhaltlicher Teil

⇨ Die aufgaben- oder produktorientiert strukturierten Inhalte des Dokuments werden hier beschrieben.

[**Hier** klicken und Text eingeben]

Anhang

⇨ Alle Informationen, die sich ohne vertiefende Erklärung tabellarisch darstellen lassen, wie z.B. Fehlermeldungen, können hier beschreiben werden.

[**Hier** klicken und Text eingeben]

Glossar

⇨ Enthält alphabetisch sortiert wichtige Begriffserläuterungen, die für das Verständnis erforderlich sind.

[**Hier** klicken und Text eingeben]

Abkürzungsverzeichnis

⇨ Alle verwendeten Abkürzungen sind in alphabetischer Reihenfolge beschrieben.

[**Hier** klicken und Text eingeben]

Literaturverzeichnis

⇨ Erwähnt ergänzende Literatur zu dem Produkt.

[**Hier** klicken und Text eingeben]

Stichwortverzeichnis

⇨ Für einen schnellen Zugriff sind in alphabetischer Reihenfolge die wichtigsten Begriffe und ihr Auftreten im Text aufgelistet.

[**Hier** klicken und Text eingeben]

Abb. 21: Gliederungsschema für Handbücher /Balzert 1996/

Nachdem über die Form der Gliederung entschieden worden ist, muß für die Dokumentation eine Gliederung ausgearbeitet werden. Auch hier stehen prinzipiell zwei Möglichkeiten zur Verfügung, nämlich *die aufgabenorientierte* und *die produktorientierte* Gliederung. Während bei der aufgabenorientierten Gliederung die vom Anwendungssystem unterstützten fachlichen Aufgaben zur Strukturierung verwendet werden, dienen bei der produktorientierten Gliederung die Funktionen des Anwendungssystems als Gliederungsgrundlage. Die

aufgabenorientierte Gliederung eignet sich in der Regel für Anfänger sehr gut, während die produktorientierte Gliederung für Experten geeignet ist.

Dokumentationsvorgaben erstellen

Wenn kein unternehmensweiter Dokumentationsstandard vorliegt, so sollten eigene Vorgaben in Form eines Musters erstellt werden. Denn wie so oft erleichtern Muster (Abb. 21) den Lesern und den Autoren die Orientierung und Navigation innerhalb eines Dokuments. Die optische und sprachliche Gestaltung des Musters sollte durch Regeln, wie z.B.

- Hervorhebung durch blaue Farbe,
- kurze Sätze formulieren,
- viele Verben benutzen,
- Substantive und Passivformen vermeiden,
- Gleiches auch gleich benennen,

festgelegt werden. Oft empfiehlt es sich, ein Wörterbuch für den internen Gebrauch aufzubauen, das neben der Erläuterung von Fachbegriffen auch ihre Schreibweise verbindlich festlegt.

Anwenderdokumentation entwickeln

Die im vorigen Schritt ausgearbeitete Gliederung wird nun mit Text und Abbildungen ausgefüllt. Hierbei sollte der Leser immer motiviert werden, indem der Zweck und der Nutzen des Beschreibungsgegenstands deutlich gemacht und anhand eines Beispiels veranschaulicht werden (*Motivation-Information-Beispiel-Prinzip*). Außerdem sollten zuerst die typischen Abläufe beschrieben werden und danach die Sonderfälle (*Regel-Ausnahme-Prinzip*).

Ergebnisse

Das oben beschriebene Gliederungsschema sollte als Grundlage zur Erstellung von Handbüchern verwendet werden. Es empfiehlt sich, mehrere Trainings- und Referenz-Handbücher anzufertigen.

2.1.4 Operationeller Einsatz

Nach dem erfolgreichen Systemtest beginnt die Phase des operationellen Einsatzes, bei dem die Aspekte der Erprobung, Nutzung und Revision betrachtet werden. In dieser Phase wird der Akzeptanztest durchgeführt und das Anwendungssystem in Betrieb genommen.

Nach der Inbetriebnahme des Anwendungssystems

- treten in der Regel Betriebsfehler auf,
- ändern sich die Umfeldbedingungen, wie z.B. durch den Einsatz neuer Hardware oder neuer Systemsoftware,
- entstehen neue Wünsche und Anforderungen, wie z.B. nach neuen Funktionen oder erhöhter Geschwindigkeit.

Um diese Korrekturen und Verbesserungen vorzunehmen, sind entsprechende Aktivitäten durchzuführen, die in der nächsten Abbildung dargestellt sind.

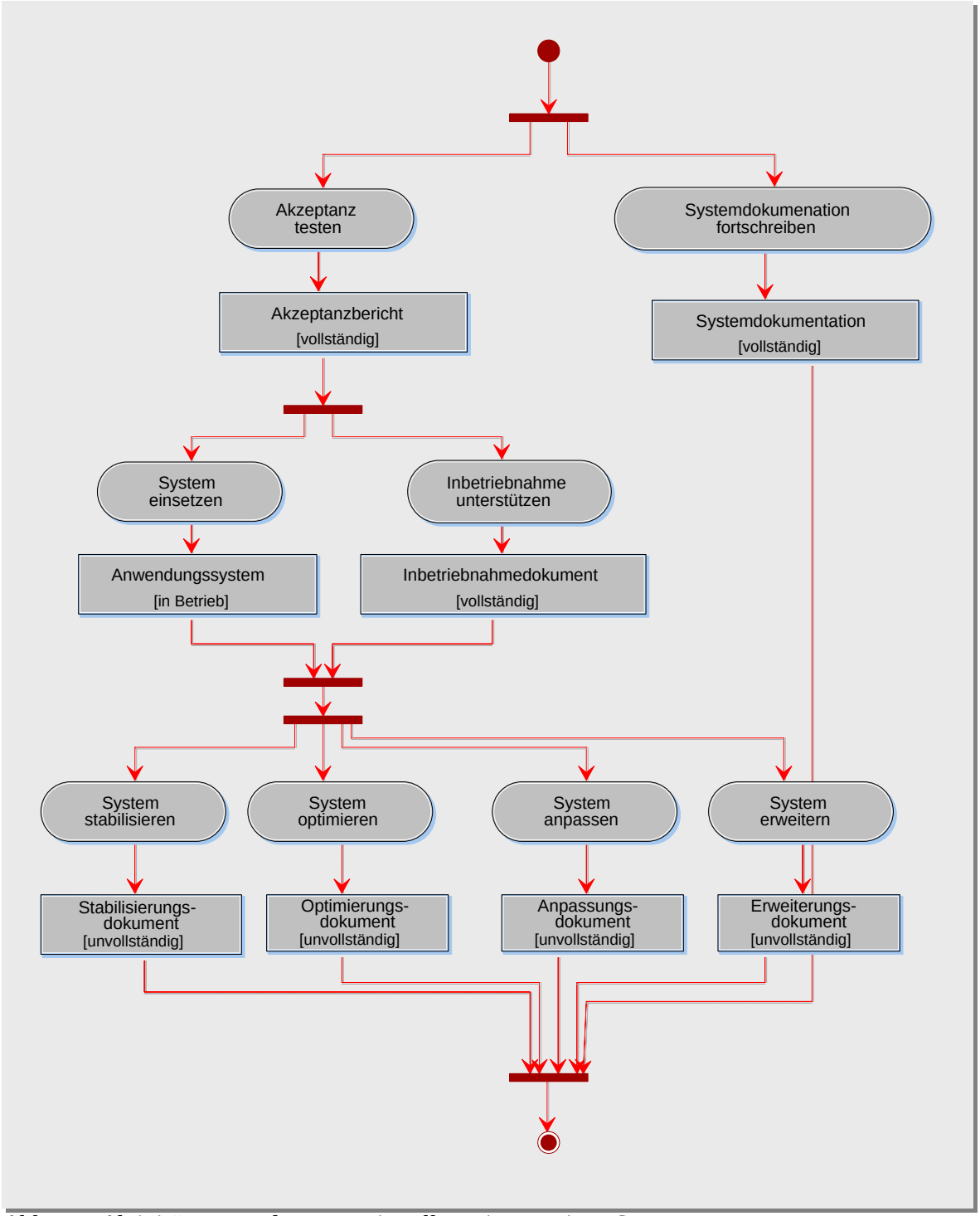


Abb. 22: Aktivitätstypen des operationellen Einsatz eines Systems

2.1.4.1 Akzeptanz testen

Zweck

Der Akzeptanztest ist eine besondere Ausprägung des Systemtests, bei dem das System unter Beobachtung und Mitwirkung des Auftraggebers in der realen Einsatzumgebung getestet wird. Der Auftraggeber ist für die Planung, wie z.B. die Definition von Testfällen, und für die Durchführung verantwortlich.

Beteiligte

Ausführend: Auftraggeber, Anwender
Beratend: Qualitätsprüfer

Vorgehen

Es sind folgende Schritte durchzuführen:

- Akzeptanztest planen,
- Testfälle definieren,
- Akzeptanztest durchführen,
- System abnehmen.

Akzeptanztest planen

Das Abnahmeverfahren für das entwickelte System sollte bereits im Projektauftrag definiert sein. Damit bleiben hier organisatorische Rahmenbedingungen, wie z.B. Umfang des Tests, zu klären.

Testfälle definieren

Bei der Bildung von Testfällen können folgende Möglichkeiten kombiniert werden:

- Die Testfälle des Systemtests werden übernommen und gegebenenfalls modifiziert.
- Eigene Testfälle werden aus den funktionalen und nicht-funktionalen Anforderungen definiert.
- Es werden Testfälle definiert, die die Geschäftsvorfälle in einem typischen Zeitraum widerspiegeln.

Für die Spezifikation dieser Testfälle können die bereits vorgestellten Beschreibungsschemata verwendet werden.

Akzeptanztest durchführen

Der Ablauf und die Ergebnisse des Akzeptanztests sollten ausführlich protokolliert werden, so daß eine Gewichtung der erkannten Fehler vorgenommen werden kann. Bei schwerwiegenden Fehlern müssen gegebenenfalls der Akzeptanztest für die betroffenen Systemteile wiederholt werden.

System abnehmen

Die protokollierten Ergebnisse des Akzeptanztests bilden die Grundlage für den Auftraggeber, das System abzunehmen oder auf weiteren Verbesserungen zu bestehen. In der Regel wird das Anwendungssystem nicht völlig fehlerfrei sein, so daß der Auftraggeber zu entscheiden hat, ob ihre Behebung im nächsten Zyklus vollzogen werden soll.

Ergebnisse

Die Dokumentation des Akzeptanztests erfolgt analog zum Systemtest.

2.1.4.2 Inbetriebnahme unterstützen

Zweck

Dieser Aktivitätstyp dient dazu, den Endanwender bei der Übernahme des Systems in einen laufenden Betrieb zu unterstützen. Eine Übergangsstrategie und ein Stufenplan für die Systemeinführung werden festgelegt.

Beteiligte

Ausführend: Anwendungsbereichsexperte, Softwarearchitekt, Softwareentwickler
Mitwirkend: Anwender/Auftraggeber, Projektleiter

Vorgehen

Es sind folgende zwei Schritte durchzuführen:

- Einführungsstrategie planen,
- Massnahmen zur Inbetriebnahme definieren und durchführen.

Einführungsstrategie planen

Die Übergangsstrategie und ein Stufenplan für die Systemeinführung werden endgültig festgelegt. Eine wichtige Aufgabe bei der Einführung ist die Umstellung bzw. die Migration von Datenbeständen. Manuelle Karteien müssen entsprechend aufbereitet werden, bevor sie im neuen Anwendungssystem erfaßt werden können. Digitale Datenbestände müssen gegebenenfalls in andere Formate überführt werden, damit sie in das neue Anwendungssystem übernommen werden können.

Die eigentliche Inbetriebnahme kann auf folgende drei Arten vorgenommen werden:

- *direkte Umstellung*: Es wird unmittelbar vom alten auf das neue Anwendungsystem übergegangen. Das alte System wird gestoppt, um das neue Anwendungssystem sofort in Betrieb zu nehmen. Die direkte Umstellung birgt viele Risiken und sollte vermieden werden.
- *Parallellauf*: Sowohl das alte System als auch das neue Anwendungssystem laufen parallel, so daß die Ergebnisse verglichen werden können. Der Vorteil des Parallellaufs ist die hohe Sicherheit. Als nachteilig erweisen sich jedoch die hohen Kosten und die Schwierigkeiten, die durch den Parallellauf zweier Systeme entstehen.
- *stufenweise Umstellung:* Das neue Anwendungssystem wird in einzelnen Stufen, in denen verschiedene Funktionsbereiche sukzessiv übernommen werden, eingeführt. Die stufenweise Umstellung bietet hohe Sicherheit. Durch die Planung und Einführung der Stufen entstehen aber zusätzliche Kosten.

Maßnahmen zur Inbetriebnahme definieren und durchführen

Das Vorgehen besteht aus der Definition und Durchführung von Maßnahmen zur Inbetriebnahme. Beispiele für solche Maßnahmen wären:

- Den Auftraggeber bei der Einführungsstrategie unterstützen.
- Den Anwendungssystem an den Einsatzorten installieren und ein betriebsbereiten Zustand herstellen.
- Sicherheitsmechanismen, wie z.B. Benutzer-Registrierung und Vergabe von Benutzer-rechten, vornehmen.
- Die bereits geschulten Anwender zu Beginn betreuen.
- Hotline-Service zur Unterstützung der Anwender bei Fragen und Fehler einrichten.

Ergebnisse

Als Ergebnis sollte ein in Betrieb genommenes Anwendungssystem vorliegen. Außerdem sollte die Inbetriebnahme samt der Einführungsstrategie protokolliert werden.

2.1.4.3 System stabilisieren

Zweck

Das Ziel dieses Aktivitätstyps ist es, auftretende Fehler zu beheben. Dabei kann es sich um Fehler handeln, die bereits bei der Entwicklung in das Anwendungssystem gelangt sind, oder um Fehler, die durch Korrektur-Maßnahmen neu entstanden sind.

Beteiligte

Ausführend: Softwaredesigner, Softwareentwickler
Beratend: Projektleiter, Anwender

Vorgehen

Der aufgetretene Fehler sollte erfaßt und analysiert werden. Hierzu kann das Beschreibungsschema (Abb. 23), in dem auch entsprechende Checklisten enthalten sind, hilfreich sein. Nach der Fehleranalyse sollte entschieden werden, ob es sich bei der Fehlermeldung tatsächlich um einen Fehler handelt, oder ob eine Optimierung, Anpassung oder Erweiterung vorliegt. Der aufgetretene Fehler muß bewertet werden, und es muß eine entsprechende Lösung vorgeschlagen werden. Aufgrund der Priorität der Fehler muß geprüft werden, ob der Fehler sofort korrigiert werden muß oder erst im nächsten Entwicklungszyklus. Die betroffenen Personen sind von dieser Entscheidung zu informieren.

Fehlernummer	⇨ Nennen Sie die Fehlernummer, die das Anwendungssystem angezeigt hat. [**Hier** klicken und Text eingeben]
Fehlermeldung	⇨ Beschreiben Sie hier den Fehler. [**Hier** klicken und Text eingeben]

Gemeldet von	⇨ Nennen Sie die Person, die den Fehler bemerkt hat, so daß gegebenenfalls nachgefragt werden kann. [**Hier** klicken und Text eingeben]
Aufdeckungs-zeitpunkt	⇨ Notieren Sie wann der Fehler aufgedeckt worden ist. [**Hier** klicken und Text eingeben]
Aufdeckungs-methode	⇨ Beschreiben Sie wie der Fehler bemerkt wurde (z.B. zufällig oder durch systematisches Vorgehen). [**Hier** klicken und Text eingeben]
Priorität	⇨ Nicht alle Fehler sind gleich wichtig. Vergeben Sie hier pro Fehler eine Priorität, z. B. zwischen 1 wie sehr hoch und 4 wie gering. [**Hier** klicken und Text eingeben]
Fehlerart	⇨ Analysieren Sie den Fehler und ordnen Sie ihn einer der folgenden Kategorien zu. ☐ Fehlende Fallunterscheidung ☐ Umgebungsproblem ☐ Initialisierungs- oder Adressierungsproblem ☐ Schleifenzähler außerhalb der Bereichsgrenzen ☐ Sonstige
Fehlerursache	⇨ Versuchen Sie die Fehlerursache zu ermitteln und zu kategorisieren. ☐ Informationsfluß ☐ Dokumentationsmangel ☐ Änderungskontrolle ☐ Algorithmisches Verständnis ☐ Flüchtigkeit / Störungen ☐ Sonstige
Betroffene Klassen und Komponenten	⇨ Ermitteln Sie die betroffenen Klassen und Komponenten und beschreiben Sie die entsprechenden Programmänderungen. [**Hier** klicken und Text eingeben]
Zuständigkeit	⇨ Tragen Sie hier die Person ein, die für die Korrektur der Fehler verantwortlich ist. [**Hier** klicken und Text eingeben]
Status	⇨ Tragen Sie hier den Status der Fehlerbebung ein. ☐ behoben ☐ in Bearbeitung ☐ wird im nächsten Zyklus behoben
Hinweise zur Fehlerbehebung	⇨ Sammeln Sie hier alle Hinweise und Bemerkungen, die bei der Ermittlung von Fehlern und bei der Fehlerbehebung von Nutzen sein können. [**Hier** klicken und Text eingeben]

Abb. 23: Beschreibungsschema für Fehler

Ergebnisse

Die aufgetretenen Fehler sollten mit dem obigen Beschreibungsschema dokumentiert werden. Außerdem sollten korrigierte Fehler vorliegen.

2.1.4.4 System optimieren

Zweck

Neu eingesetzte Software ist nicht nur fehlerhaft, sondern verbraucht auch in der Regel mehr Zeit und Speicher als zur Erfüllung ihrer Aufgaben erforderlich. Denn oft hat die Implementierung der Funktionalität eine höhere Priorität als Optimierungsroutinen. Das Ziel dieses Aktivitätstyps ist es, die Leistung des Anwendungssystems zu verbessern.

Beteiligte

Ausführend: Softwaredesigner, Softwareentwickler
Beratend: Projektleiter, Anwender

Vorgehen

Die Optimierungsvorschläge sollten mit dem Beschreibungsschema in der Abbildung 24 dokumentiert und bewertet werden. *Tuning*, *Monitoring* und Reduzierung des Speicherbedarfs sind mögliche durchzuführende Aufgaben. Oft sind aber Restrukturierungen in der Software erforderlich, um die Leistungsverbesserungen zu erreichen. In solchen Fällen sollte die Optimierung im nächsten Entwicklungszyklus vorgenommen werden.

Beschreibung	⇨ Beschreiben Sie hier den Optimierungs-, Anpassungs- oder Erweiterungsvorschlag detailliert. [**Hier** klicken und Text eingeben]
Gemeldet von	⇨ Nennen Sie die Person, die den Vorschlag verkündet hat. [**Hier** klicken und Text eingeben]
Datum	⇨ Notieren Sie wann der Vorschlag gemacht worden ist, damit ersehen werden kann, wie aktuell der Vorschlag ist. [**Hier** klicken und Text eingeben]
Priorität	⇨ Nicht alle Vorschläge sind gleich wichtig. Vergeben Sie hier pro Vorschlag eine Priorität, z. B. zwischen 1 wie sehr hoch und 4 wie gering. [**Hier** klicken und Text eingeben]
Betroffene Klassen und Komponenten	⇨ Ermitteln Sie die betroffenen Klassen und Komponenten und beschreiben Sie die entsprechenden Programmänderungen. [**Hier** klicken und Text eingeben]
Zuständigkeit	⇨ Tragen Sie hier die Person ein, die für die Realisierung des Vorschlags verantwortlich sein könnte. [**Hier** klicken und Text eingeben]
Status	⇨ Tragen Sie hier den Status der Fehlerbebung ein. ☐ erledigt ☐ in Bearbeitung ☐ wird im nächsten Zyklus erledigt
Hinweise	⇨ Sammeln Sie hier alle Hinweise und Bemerkungen, die für die Verarbeitung von anderen Vorschlägen relevant sein können. [**Hier** klicken und Text eingeben]

Abb. 24: Beschreibungsschema für progressive Vorschläge

Ergebnisse

Eine Dokumentation der Optimierungsvorschläge sollte vorliegen.

2.1.4.5 System anpassen

Zweck

Durch Veränderungen im System-Umfeld werden Anpassungen des Anwendungssystems erforderlich. Beispiele für solche Anpassungen sind:

- Ändern der technischen Umgebung, wie z.B. neue Systemsoftware,
- Ändern der Benutzeroberfläche, wie z.B. modifizierte Dialoge,
- Ändern der Funktionen, z.B. bedingt durch Gesetzesänderungen.

Beteiligte

Ausführend: Softwaredesigner, Softwareentwickler
Beratend: Projektleiter, Anwender

Vorgehen

Das Beschreibungsschema in Abbildung 24 sollte genutzt werden, um die Anpassungsvorschläge zu dokumentieren. Je nach Priorität und Umfang der Anpassung muß entschieden werden, ob diese im nächsten Entwicklungszyklus durchgeführt werden sollen. Die betroffenen Personen sollten über diese Entscheidung informiert werden.

Ergebnisse

Eine Dokumentation der Anpassungsvorschläge sollte vorliegen.

2.1.4.6 System erweitern

Zweck

Erweiterungen führen zu einer funktionalen Ergänzung des Anwendungssystems. Das sind Funktionen, die beim ersten Zyklus vorgesehen waren, aber nicht implementiert wurden, oder sich aus den Erfordernissen des Betriebs ergaben.

Beteiligte

Ausführend: Softwaredesigner, Softwareentwickler
Beratend: Projektleiter, Anwender

Vorgehen

Die Erweiterungsvorschläge sollten mit Hilfe des Beschreibungsschemas in Abbildung 24 erfaßt werden. Auch hier muß wieder die Entscheidung getroffen werden, in welchem Entwicklungszyklus die entsprechende Programmierung vorgenommen wird. Oft wird der Aufwand solcher Erweiterungen unterschätzt. Studien haben ergeben, daß dieser Aufwand 40% des Gesamtaufwandes ausmacht.

Ergebnisse

Eine Dokumentation der Erweiterungsvorschläge sollte vorliegen.

Hinweise und Tipps

Die Realisierung von Optimierungen, Anpassungen und Erweiterungen in einem weiteren Entwicklungszyklus bringt folgende Vorteile mit sich:

- Klare Zuordnung von Entwicklungskosten,
- Bessere Planbarkeit durch strikte Trennung zwischen abgeschlossener Entwicklung und Weiterentwicklung.

2.2 Komponenten-Ebene

Im folgenden werden die Aktivitäts- und Produkttypen für einen Komponenten-Zyklus mit den entsprechenden Rollen detailliert beschrieben. Wenn im Rahmen einer Komponente von Anwendungsfällen oder Anwendungsfalldiagrammen die Rede ist, so sind die Anwendungsfälle oder Anwendungsfalldiagramme gemeint, die technisch durch die Komponente realisiert werden. Auf der Komponenten-Ebenen können Klassen-Zyklen angestoßen werden. Ein Klassen-Zyklus ist angestoßen, sobald die entsprechende Klasse identifiziert ist.

2.2.1 Analyse

Es ist nicht realistisch anzunehmen, daß die auf der System-Ebene entwickelte Anforderungsbeschreibung bereits lückenlos und genügend detailliert ist. Daher werden bei der Komponenten-Analyse die funktionalen und nicht-funktionalen Anforderungen, die die jeweilige Komponente technisch realisieren, überarbeitet und verfeinert.

Während der Analyse wird die Bausteinbibliothek zwecks Wiederverwendung von Komponenten durchsucht. Außerdem wird für die zu analysierende Komponente eine Komponentendokumentation angelegt, die stets erweitert und verfeinert werden muß (Abb. 25).

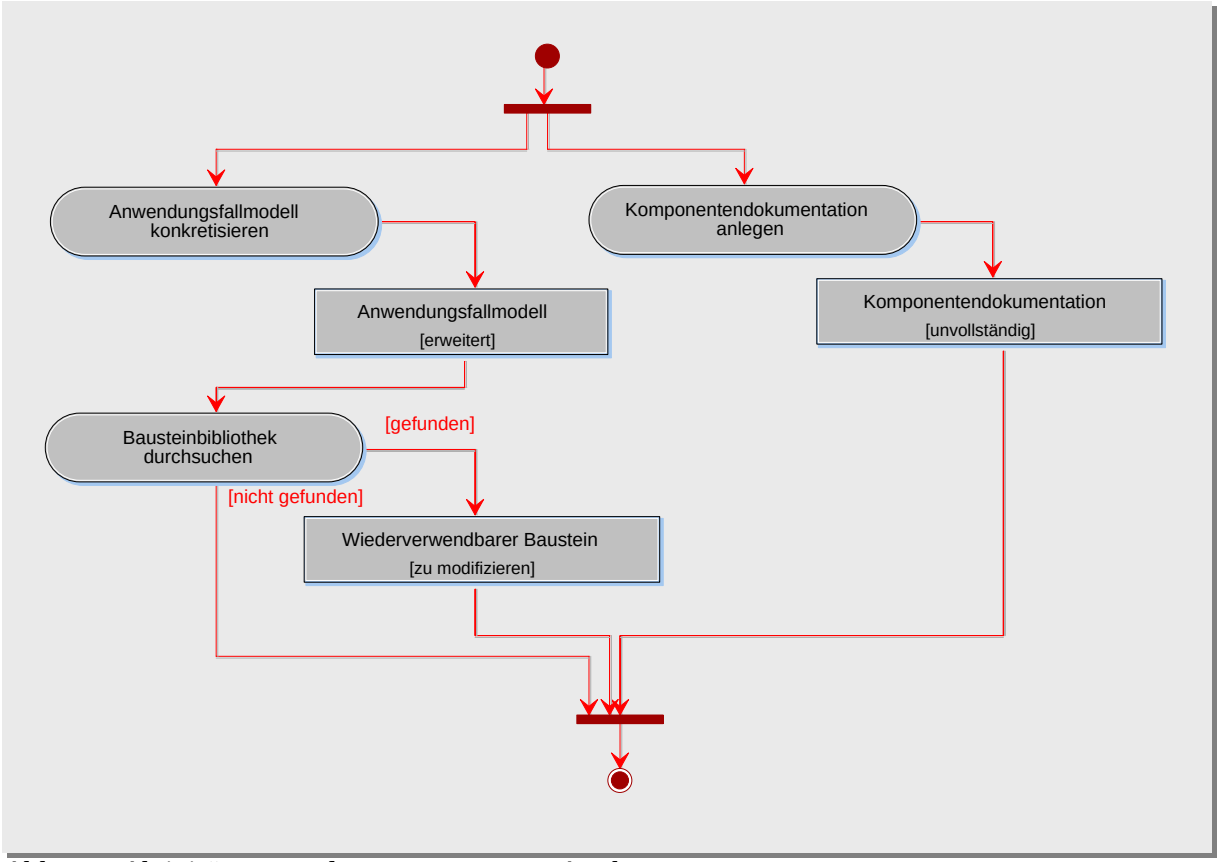


Abb. 25: Aktivitätstypen der Komponenten-Analyse

2.2.1.1 Anwendungsfallmodell konkretisieren

Zweck

Wie bereits erwähnt, können bei der Anforderungsanalyse unterschiedliche Methoden eingesetzt werden. Wenn die Anwendungsfallmodellierung verwendet wurde, so werden die von einer Komponente zu realisierenden Anwendungsfälle vervollständigt, korrigiert und detailliert dokumentiert. Gegebenenfalls werden Anwendungsfälle durch Aktivitäts-diagramme verfeinert oder bereits vorhandene Aktivitätsdiagramme aktualisiert.

Beteiligte

Ausführend: Systemanalytiker
Mitwirkend: Anwender

Vorgehen

Die Hauptszenarien der Anwendungsfälle sind in der Regel schon auf der System-Ebene ausreichend spezifiziert. Problematisch sind aber die möglichen Erweiterungen oder Variationen der Hauptszenarien, die z.B. zur Abwicklung von Ausnahmen benötigt werden. Diese gilt es zu konkretisieren und zu dokumentieren. Dabei kann wie folgt vorgegangen werden:

- Vollständigkeit der Vor- und Nachbedingungen der Anwendungsfälle prüfen,

- Beziehungen der Anwendungsfälle prüfen,
- Änderungen dokumentieren.

Vollständigkeit der Vor- und Nachbedingungen der Anwendungsfälle prüfen

Anhand der Textbeschreibung oder des Aktivitätsdiagramms eines jeden Anwendungsfalls sollte geprüft werden, ob alle möglichen Vor- und Nachbedingungen erkannt worden sind:

- Es sollte nach impliziten Annahmen gesucht werden, da diese meistens eine fachliche Vorbedingung eines Anwendungsfalls darstellen.
- Es sollte geprüft werden, ob die Kombinationen von Vor- bzw. Nachbedingungen fachlich möglich sind.

Sollten neue Vor- bzw. Nachbedingungen gefunden werden, so ist es zu prüfen, ob diese nicht eine Erweiterung oder Variation des Anwendungsfalls erfordern.

Beziehungen der Anwendungsfälle prüfen

Die in der UML definierten Beziehungen zwischen Anwendungsfällen werden in der Praxis oft nicht richtig verwendet. Das liegt unter anderem an der unzureichenden Beschreibung dieser Beziehungen in dem UML-Standard. Daher sind insbesondere die Erweitert- und die Enthälts-Beziehung zwischen den Anwendungsfällen zu überprüfen.

Änderungen dokumentieren

Neu gefundene Anwendungsfälle, Erweiterungen oder Variationen sollten dokumentiert werden, indem die vorhandene Dokumentation modifiziert wird oder gegebenenfalls neue Dokumente angelegt werden.

Ergebnisse

Das Ergebnis dieses Aktivitätstyps sind modifizierte bzw. neu angelegte Anwendungsfall-modelle mit entsprechenden Dokumentationen.

2.2.1.2 Bausteinbibliothek durchsuchen

Zweck

Im Rahmen dieses Aktivitätstyps ist dafür zu sorgen, daß die Baussteinbibliothek nach Komponenten zwecks Wiederverwendung durchsucht wird. Neben der Bausteinbibliothek werden auch andere Quellen, wie z.B. Komponentenanbieter, bei der Suche berücksichtigt.

Beteiligte

Ausführend: Systemanalytiker, Softwaredesigner
Mitwirkend: Projektbibliothekar

Vorgehen

Die Bausteinbibliothek wird auf eine Komponente hin durchsucht werden, die über die gleichen Dienste verfügt wie die zu analysierende Komponente. Ist die Suche erfolgreich, so kann die gefundene Komponente möglicherweise nach Modifikationen wiederverwendet werden. Folgende Schritte werden durchlaufen:

- Die gefundene Komponente wird in die bereits vorliegende Komponentenstruktur eingefügt.
- Die Beziehungen zu den bereits vorhandenen Komponenten müssen angelegt und geprüft werden.
- Die Anforderungen an Modifikationen müssen spezifiziert werden. Diese Anforderungen müssen in den nachfolgenden Phasen realisiert werden.

Ergebnisse

Wenn die Suche in der Bausteinbibliothek erfolgreich war, so liegt eine wiederverwendbare Komponente als Resultat vor.

2.2.2 Entwurf

Nachdem mögliche Subkomponenten einer Komponenten identifiziert worden sind, werden für die einzelnen Subkomponenten die untergeordneten Klassen ermittelt. Für die gefundenen Klassen werden die statischen und dynamischen Aspekte modelliert. Ferner ist der Nachweis zu erbringen, daß die der Komponente zugeordneten, funktionalen und nicht-funktionalen Anforderungen durch die gefundenen Klassen erfüllt werden können. Die nächste Abbildung definiert die entsprechenden Aktivitätstypen.

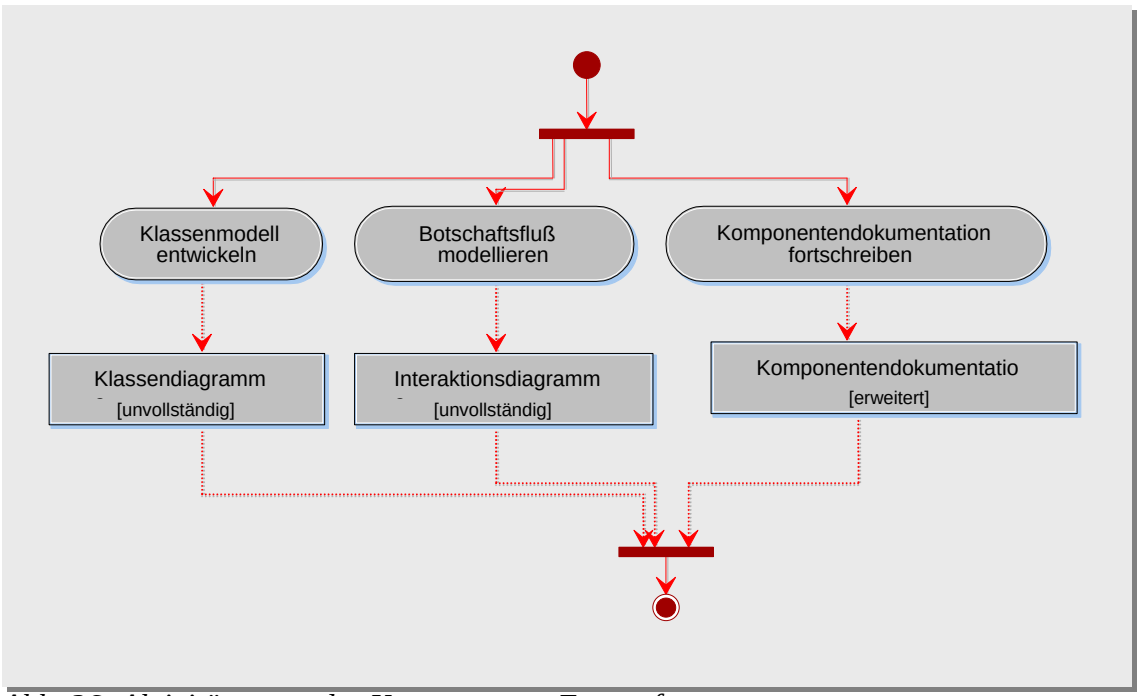


Abb. 26: Aktivitätstypen des Komponenten-Entwurfs

2.2.2.1 Klassenmodell entwickeln

Zweck

Dieser Aktivitätstyp dient zum Auffinden von Klassen einer Komponente. Gefundene Klassen werden graphisch durch Klassendiagramme beschrieben.

Beteiligte

Ausführend: Systemanalytiker, Softwaredesigner
Beratend: Anwender

Vorgehen

- Klassen finden,
- Klassendiagramme definieren,
- Beziehungen definieren,
  - Assoziationen,
  - Aggregationen und Kompositionen,
  - Generalisierungen,
- Klassendiagramme prüfen.

Klassen finden

Grundlage für diesen Schritt sind die Anwendungsfälle der Komponente. Um einen Anwendungsfall abwickeln zu können, werden in der Regel mehrere Klassen benötigt. Die einzelnen Klassen können wiederum an mehreren Anwendungsfällen beteiligt sein. Das bedeutet, das es keine 1:1 Beziehung zwischen Anwendungsfällen und Klassen gibt. Trotzdem macht es beim Identifizieren von Klassen Sinn, von Anwendungsfällen auszugehen.

Bei der Modellierung ist es hilfreich, zwischen Entity-, Control- und Boundary-Klassen zu unterscheiden /Jacobson 1992/. *Entity-Klassen* haben einen passiven Charakter und bilden Gegenstände und Produkte ab, die bei der Abwicklung von Anwendungsfällen benötigt werden. Die Steuerung von Aufgaben, die im Rahmen der Durchführung von Anwendungsfällen anfallen, werden durch die *Control-Klassen* übernommen. Die *Boundary-Klassen* sorgen für die Kommunikation mit den Akteuren innerhalb eines Anwendungsfalls.

1) Boundary-Klassen finden:

Am einfachsten können Boundary-Klassen gefunden werden, wenn eine Benutzeroberfläche für einen Anwendungsfall entwickelt worden ist, z.B. in Form eines Prototypen. Die Benutzeroberfläche kann in der Regel leicht auf die Boundary-Klassen abgebildet werden. Beispielsweise kann aus eine Eingabemaske eine Boundary-Klasse „Eingabedialog“ abgeleitet werden. Liegt keine Benutzeroberfläche vor, so müssen Systemanalytiker bzw. Softwaredesigner die entsprechenden Benutzeroberflächen entwerfen und daraus die Boundary-Klassen ableiten.

2) Entity-Klassen finden:

Das Wissen, das für die Abwicklung eines Anwendungsfalls benötigt wird, wird in Form von Entity-Klassen modelliert. Einfache Techniken zum Finden von Entitiy-Klassen wären: die Reale-Welt-Suche, das Kategorisieren und die Textanalyse. Bei der *Realen-Welt-Suche* werden die Arbeitsabläufe, die durch das künftige Anwendungssystem unterstützt werden sollen, beobachtet und nach Klassen-Kandidaten gesucht. Beim *Kategorisieren* wird der Gegenstandsbereich in Kategorien, wie z.B. Personen, Orte oder Organisationseinheiten, unterteilt. Jede Kategorie ist ein potenzieller Objekt-Kandidat. Die *Textanalyse* ist ebenfalls eine einfache, aber effiziente Methode, um Klassen-Kandidaten zu finden. Hierbei werden alle möglichen Projektdokumente, wie z.B. Pflichtenhefte, Protokolle, Benutzeroberflächen, Anwendungsfall-Beschreibungen, analysiert, um Objekt-Kandidaten zu finden. Inzwischen gibt es auch formale Methoden der Textanalyse /Düwel 2000/. Um Entity-Klassen mit der Technik der Textanalyse zu finden, sollten Textquellen auf Substantive untersucht werden, da Substantive oder substantivierte Verben mögliche Objekt-Kandidaten sind.

3) Control-Klassen finden:

Für jeden Anwendungsfall sollte mindestens eine Control-Klasse entworfen werden, die den Ablauf des Anwendungsfalls steuert. Control-Klassen, wie z.B. „Reservierung“ oder „Steuerberechnung“, sollten natürlich keine willkürliche Zusammenstellung von Funktionalität sein. Sie sollten eine definierte Aufgabe erfüllen, die sich im Namen widerspiegelt.

Der Abschluß dieses Schritts besteht aus einer Überprüfung der gefundenen Klassen. Folgende Aspekte sollten beachtet werden:

- Können alle Anwendungsfälle der Komponente mit den gefundenen Klassen realisiert werden?
- Leisten die Klassen auch wirklich einen Beitrag, um einen Anwendungsfall abzuwickeln?
- Sind mehrere Objekte der jeweiligen Klassen eindeutig voneinander unterscheidbar?
- Sind die Klassen atomar, d.h. nicht mehr in andere Klassen zerlegbar?
- Bieten die Klassen Verhalten in Form von Methoden an?
- Besitzen die Attribute der Objekte sinnvolle Werte, d.h. keine undefinierten Einträge, wie z.B. „NULL“?
- Gibt es Klassen, die in der Bausteinbibliothek gefunden und wiederverwendet werden könnten?

Klassendiagramme definieren

Klassen werden graphisch durch Klassendiagramme beschrieben. Ein Klassendiagramm sollte nur Klassen enthalten, die eine gemeinsame Leistung erbringen, z.B. einen Anwendungsfall realisieren. Nach Bedarf sollten die definierten Klassendiagramme mit dem folgenden Beschreibungsschema dokumentiert werden.

Zweck und Leistung	⇨ Aus welchem Grund wurde das Klassendiagramm angelegt? Welchen semantischen Kontext soll es beschreiben? Speziell: Welche Anwendungsfälle decken seine Klassen ab?
	[**Hier** klicken und Text eingeben]
Annahmen und Restriktionen	⇨ Liegen der Bildung des Klassendiagramms spezielle Annahmen oder Restriktionen zugrunde?
	[**Hier** klicken und Text eingeben]
Erkenntnisse	⇨ Was haben Sie bei der Gestaltung des Klassendiagramms erkannt, welche Konsequenzen

und Konse-quenzen	haben Sie insbesondere hinsichtlich der Gliederung des Systems daraus gezogen? [**Hier** klicken und Text eingeben]
Offene Fragen	⇨ Welche Gestaltungsentscheidungen stehen noch aus? [**Hier** klicken und Text eingeben]
Bearbeitungs-status	⇨ in Bearbeitung, abgenommen, etc. [**Hier** klicken und Text eingeben]

Abb. 27: Beschreibungsschema für Klassendiagramme /objectiF 4.5, microTOOL GmbH/

Auch bei den Klassendiagrammen gilt die Faustregel: nicht mehr als 7 Elemente. Wenn ein Klassendiagramm gegen diese Regel verstößt, so ist es empfehlenswert, eine Zerlegung in Teildiagramme vorzunehmen, damit es überschaubar bleibt.

Beziehungen definieren: Assoziationen

Eine Assoziation stellt einen Zugriffsweg auf Eigenschaften und Verhalten von assoziierten Objekten dar. Ferner bietet sie die Möglichkeit der Delegation von Aufgaben, was eine interessante Alternative zur Vererbung sein kann. Die nächste Abbildung veranschaulicht die Alternativen an einem einfachen Beispiel.

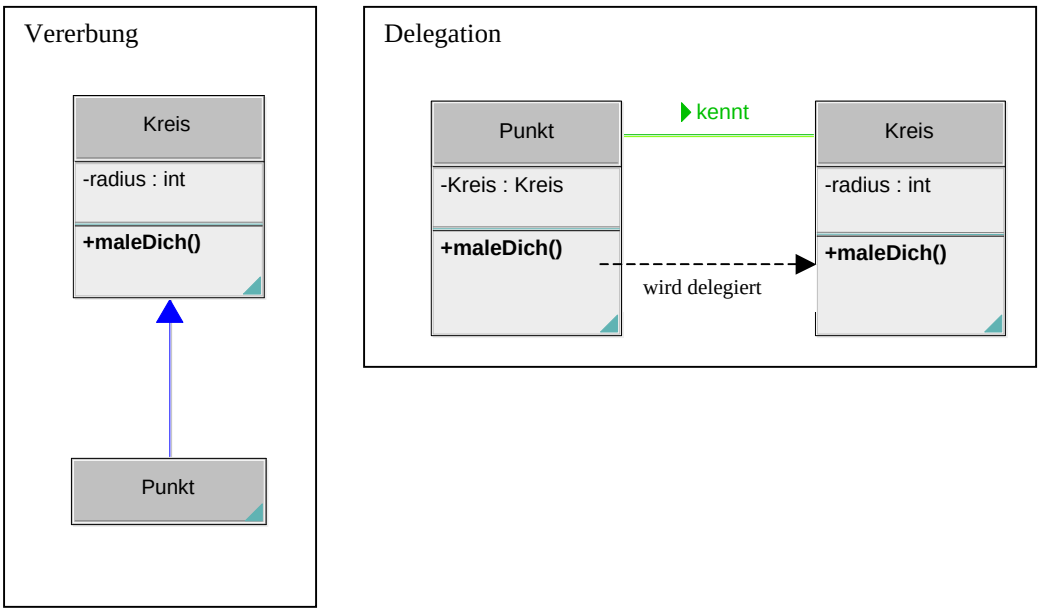


Abb. 28: Unterschied zwischen Vererbung und Delegation

Wenn man annimmt, daß ein Punkt bloß ein Kreis mit einem Radius > 0 ist, so würde man einen Punkt als eine Spezialisierung eines Kreises modellieren (Abb. 28, links). In diesem Fall bräuchte man in der Klasse „Punkt“ die Methode „maleDich“ gar nicht zu implementieren. Der Sachverhalt könnte aber auch durch die Assoziation modelliert werden (Abb. 28, rechts). In diesem Fall verfügt die Klasse „Punkt“ über ein Beziehungsattribut vom Typ „Kreis“. Die Methode „maleDich“ der Klasse „Punkt“ wird durch einen Aufruf der entsprechenden Methode der Klasse „Kreis“ realisiert.

Die Delegation sollte an Stelle der Generalisierung angewendet werden, sofern Zweifel bestehen, daß die abgeleitete Klasse ihre Basisklasse vollständig ersetzen kann. Folgende Fragen helfen bei der Entwicklung von Assoziationen:

- Gibt es Klassen, deren Objekte ein dauerhaftes Objekt einer anderen Klasse besitzen? Wenn ja, so liegt eine Assoziation vor.
- Welche Klassen müssen sich unbedingt kennen?

Die Textanalyse bietet ebenfalls eine Möglichkeit, Assoziationen zu identifizieren. Dazu muß nach Verben gesucht werden, denn diese sind Kandidaten für die Assoziationen. Das Arbeiten mit Mustern kann die Suche nach Assoziationen erleichtern. Im folgenden listen wir einige Muster auf, die stark mit assoziierten Klassen arbeiten. Details können in der referenzierten Literatur nachgelesen werden.

- Sammler/Zulieferer-Muster (*Collection/Worker-Pattern*) /Coad 1997/,
- Koordinator-Muster /Balzert 1995/,
- Plan/Planausführungs-Muster (*Plan/Plan-Execution-Pattern*) /Coad 1997/.

Bevor die gefundenen Assoziationen im Klassendiagramm eingetragen werden, sollte noch überprüft werden, ob die Assoziationen auch aus fachlicher Sicht einen Sinn machen. Denn zwischen Klassen lassen sich oft unzählige, fachlich unnötige, Assoziationen definieren. Zu einer Assoziation kann noch Folgendes festgelegt werden: eine Bezeichnung, ein Stereotyp, Rollen, Multiplizität und Constraints. Wenn nötig, sollte die Assoziation auch als Text dokumentiert werden. Ein entsprechendes Beschreibungsschema wird in der nächsten Abbildung dargestellt.

Zweck und Bedeutung	⇨ Beschreiben Sie hier die Bedeutung der Assoziation und ihre Aufgabe in der modellierten Lösung. Begründen Sie Ihre Gestaltungsentscheidung. [**Hier** klicken und Text eingeben]
Rollenbeschreibung der assoziierten Klassen	⇨ Spezifizieren Sie hier die Bedeutung genauer, die den assoziierten Klassen in der Beziehung jeweils zukommt. [**Hier** klicken und Text eingeben]
Benutzt von/für	⇨ Durch diese Angabe begründen Sie die Notwendigkeit der modellierten Assoziation. [**Hier** klicken und Text eingeben]
Restriktionen/ Bedingungen	⇨ Halten Sie hier alle Einschränkungen im Zusammenhang mit der Multiplizität fest. Dazu gehört z.B. eine erforderliche Sortierung. [**Hier** klicken und Text eingeben]

Abb. 29: Beschreibungsschema für Assoziationen /objectiF 4.5, microTOOL GmbH/

Beziehungen definieren: Aggregationen und Kompositionen

Eine Aggregation ist ein Spezialfall einer Assoziation. Die beteiligten Klassen sind jedoch nicht gleichrangig. Es gibt eine Aggregat-Klasse, die die anderen umfaßt. Ein Aggregat „kennt“ alle seine Teile und reagiert stellvertretend für seine Teile auf Botschaften, d.h. Operationen werden übernommen und an die Teile weitergeleitet. Dieser Vorgang wird als *Propagation* bzw. *Fortpflanzung von Operationen* bezeichnet. Eine Aggregation setzt nicht unbedingt eine Existenzabhängigkeit der Teile vom Ganzen voraus, d.h.

- Teile können unabhängig existieren,
- Teile können Teile anderer Aggregate sein,
- Teile können selbst Aggregate sein.

Eine Komposition ist eine Aggregation, bei der die Lebensdauer der Teile an das Aggregat gekoppelt ist. Um Aggregationen und Kompositionen zu finden, können die folgenden vier Strategien kombiniert werden:

- Top-Down-Vorgehen
  Alle gefundenen Klassen werden darauf untersucht, ob sie sich in Teile zerlegen lassen.

- Bottom-Up-Vorgehen
  Die gefundenen Klassen werden darauf untersucht, ob sie sich nicht zu einem Aggregat verdichten lassen.

- Kategorisieren/Textanalyse
  Die gefundenen Klassen werden in Aggregations-Kategorien eingeordnet. Beispiele für Aggregations-Kategorien wären /Coad 1997/: Ganzes/Teile, Container/Inhalt, Verpackung/Exemplare, Gruppe/Mitglieder, Sammlung/Stücke. Hat man solche Kategorien gefunden, so liegt eine Aggregation vor. Auch die Textanalyse kann eingesetzt werden, um nach solchen Kategorien zu suchen.

- Muster verwenden
  Hier folgt eine Liste von Mustern, die sich die Aggregations- und Kompositions-Mechanismen zu Nutze machen. Das Erläutern würde den Rahmen springen, so daß wir auf die entsprechende Literatur hinweisen.

  - Transaktion/Positionen /Coad 1995/,
  - Spezifikation/Exemplare /Balzert 1995/,
  - Gegenstand/Ereignisse /Balzert 1995/,
  - Statische/Variable Hierarchie.

Zuletzt sollten die identifizierten Aggregationen und Kompositionen überprüft werden. Wenn man nicht sicher ist, ob eine Komposition vorliegt, so sollte besser eine Aggregation modelliert werden. Die Faustregel besagt: Im Zweifelsfall immer eine Assoziation modellieren. Aggregationen und Kompositionen können mit Rollen, Mulitiplizität, Constraits näher spezifiziert werden, gegebenenfalls können sie auch als Text dokumentiert werden (Abb. 30).

Zweck und Bedeutung	⇨ Beschreiben Sie hier die Bedeutung der Aggregation/Komposition und ihre Aufgabe in der modellierten Lösung. Begründen Sie Ihre Gestaltungsentscheidung. [**Hier** klicken und Text eingeben]
Rollenbeschreibung der assoziierten Klassen	⇨ Spezifizieren Sie hier die Bedeutung genauer, die der Aggregat-Klasse und den Teile-Klassen in der Beziehung zukommt. [**Hier** klicken und Text eingeben]
Benutzt von/für	⇨ Durch diese Angabe begründen Sie die Notwendigkeit der modellierten Aggregation/Komposition. [**Hier** klicken und Text eingeben]
Restriktionen/Bedingungen	⇨ Halten Sie hier alle Einschränkungen im Zusammenhang mit der Multiplizität fest. Dazu gehört z.B. eine erforderliche Sortierung. [**Hier** klicken und Text eingeben]

Abb. 30: Beschreibungsschema für Aggr./Kompositionen /objectiF 4.5, microTOOL GmbH/

Beziehungen definieren: Generalisierungen

Als einen weiteren Abstraktionsmechanismus bietet die Objektorientierung die Generalisierung an. Hierbei werden gemeinsame Attribute und Methoden von Klassen zwecks Wiederverwendung in eine *Basisklasse* gekapselt. Eine Klasse, die durch Erweiterung bzw. Spezialisierung aus der Basisklasse gewonnen wird, wird als eine *abgeleitete Klasse* bezeichnet. Zur Entwicklung der Generalisierung stehen folgende Techniken zur Verfügung:

- Textanalyse
  Kandidaten für die Generalisierung sind zusammengesetzte Substantive und Substantive, die durch Adjektive näher beschrieben sind.

- Top-Down-Vorgehen
  Beim Anlegen neuer Klassen, sollte geprüft werden, ob bereits eine Klasse existiert, die einen Teil der benötigten Eigenschaften oder des gewünschten Verhaltens besitzt. Wenn ja, so handelt sich bei der neu angelegten Klasse um eine abgeleitete Klasse. Die bereits gefundenen Klassen sollten hinsichtlich einer Basisklasse geprüft werden.

- Bottom-Up-Vorgehen
  Wenn zur Abwicklung der definierten Anwendungsfälle allgemein gültige Eigenschaften oder Verhalten benötigt werden, so sollten diese in Basisklassen gekapselt werden.

- Muster verwenden
  Eine Reihe von Mustern arbeiten ebenfalls stark mit dem Mechanismus der Generalisierung. Ein Beispiel hierzu wäre: Strategie-Muster (*Strategy-Pattern*) /Gamma 1995/.

Die gefundenen Generalisierungen sollten zuletzt nach folgenden Gesichtspunkten geprüft werden:

- Handelt es sich bei den abgeleiteten Klassen lediglich um Synonyme der Basisklassen oder besitzen die abgeleiteten Klassen wirkliche Erweiterungen?
- Werden die Attribute und Methoden der Basisklasse auch von allen abgeleiteten Klassen benötigt?
- Ist die gefundene Generalisierung/Spezialisierung für das Anwendungssystem überhaupt relevant?
- Unterscheiden sich die abgeleiteten Klassen einer Basisklasse bezüglich ihrer Attribute und Methoden voneinander?
- Handelt es sich bei der abgeleiteten Klasse wirklich um eine Spezialisierung oder könnte sie lediglich ein Attribut der Basisklasse sein? Hierbei ist der semantische Gesichtspunkt maßgebend.

Falls nötig können die Generalisierungen mit dem folgenden Beschreibungsschema dokumentiert werden.

Zweck und Bedeutung	⇨ Beschreiben Sie hier die Bedeutung der Generalisierung. Geben Sie hier außerdem Auskunft über die von Ihnen verfolgte Gestaltungsabsicht. [**Hier** klicken und Text eingeben]	
Diskriminator	⇨ Geben Sie in Form eines Substantivs das Kriterium an, nach dem Sie spezialisiert bzw. Subtypen gebildet haben. [**Hier** klicken und Text eingeben]	
Eigenschaften	☑ disjunkt ☐ überlappend	☑ vollständig ☐ unvollständig

Abb. 31: Beschreibungsschema für eine Generalisierung /objectiF 4.5, microTOOL GmbH/

In dem obigen Beschreibungsschema ist mit „Diskriminator“ das Kriterium der Generalisierung/Spezialisierung gemeint. Zum Beispiel kann eine Klasse „Hotel“ nach Kategorien, wie z.B. Luxushotel, Mittelklassehotel, spezialisiert werden. Eine anderes Kriterium wäre die Lage, z.B. Cityhotel, Strandhotel. Die Eigenschaften „disjunkt/ überlappend“ sind nur bei der Mehrfachvererbung relevant. Denn bei der Mehrfachvererbung kann eine abgeleitete Klasse mehrere Vererbungspfade von einer Basisklasse erben. Die Eigenschaften „vollständig/unvollständig“ sollen andeuten, ob schon alle notwendigen Spezialisierungen einer Basisklasse gefunden worden sind.

Klassendiagramme prüfen

Im letzten Schritt wird das entwickelte Klassendiagramm noch nach formalen Gesichtspunkten untersucht:

- Wurden nicht benötigte Schnittstellen definiert?
- Sind die gefundenen Klassen vollständig?
- Reichen die gefundenen Attribute und Methoden aus, damit die Klassen die spezifizierten Dienste leisten können?

Ergebnisse

Klassendiagramme mit entsprechender Dokumentation sollte als Ergebnis vorliegen.

2.2.2.2 Botschaftsfluß modellieren

Zweck

Zur Beschreibung von Botschaftsflüssen zwischen Objekten bietet UML die Sequenz- und Kollaborationsdiagramme an. Diese sind sehr nützlich, um Methoden zu finden und die modellierten Klassen zu stabilisieren. Da die Sequenz- und Kollaborationsdiagramme äquivalent sind (d.h. sie können ineinander transformiert werden) haben wir uns auf die in der Praxis häufiger benutzten Sequenzdiagramme beschränkt. Prinzipiell können Sequenzdiagramme in zwei Ausprägungen verwendet werden: *generische* und *instanzbezogene*. In der generischen Form stellt ein Sequenzdiagramm einen Algorithmus dar. Hierbei werden die Sequenzdiagramme leicht unübersichtlich, daher ist es empfehlenswert, sie in der instanzbezogenen Ausprägung anzuwenden. Das heißt, daß der Botschaftsfluß eines speziellen Szenarien dargestellt wird.

Beteiligte

Ausführend: Systemanalytiker
Beratend: Softwaredesigner

Vorgehen

- Hauptszenario modellieren,
- Nebenszenarien modellieren,
- Botschaftsmodell prüfen und beschreiben.

Hauptszenario modellieren

Zuerst sollten die Objekte, die an der Durchführung der Szenarien beteiligt sind, angelegt werden und mit ihren Rollen im Szenario gekennzeichnet werden (zur Definition von Szenario siehe UML-Spezifikation). Die beteiligten Akteure werden indirekt durch die Systemgrenze repräsentiert. Als nächstes werden die Botschaftsflüsse zwischen den Objekten gestaltet. Spätestens jetzt wird man auf fehlende Methoden stoßen, die in den entsprechenden Klassen definiert werden müssen. Falls nötig, können die Botschaftsflüsse mit dem folgenden Beschreibungsschema dokumentiert werden.

Zweck und Leistung	⇨ Aus welchem Grund wurde die Botschaft modelliert? Was wird mit ihr erreicht? [**Hier** klicken und Text eingeben]
Art	⇨ Legen Sie hier die Art der Botschaft fest: ☐ einfach ☑ synchron ☐ asynchron ☐ synchron mit sofortiger Antwort
Annahmen und Restriktionen	⇨ Liegen der gestalteten Botschaft spezielle Annahmen oder Restriktionen zugrunde? Welche der gegebenenfalls bereits für die aufgerufene Methode spezifizierten Aufrufbeschränkungen sind zu beachten? Gibt es hinsichtlich des Rückgabewerts Annahmen oder Einschränkungen zu beachten? [**Hier** klicken und Text eingeben]
Zusammenhänge/ Abhängigkeiten	⇨ Stellen Sie hier eventuell vorhandene Abhängigkeiten oder Zusammenhänge mit anderen Botschaften dar. [**Hier** klicken und Text eingeben]

Abb. 32: Beschreibungsschema für Botschaften /objectiF 4.5, microTOOL GmbH/

Nebenszenarien modellieren

Nachdem das Hauptszenario modelliert ist, werden die fachlich anspruchvollsten Erweiterungen oder Variationen in eigenen Sequenzdiagrammen behandelt. Zur Gestaltung von Botschaftsflüssen in Sequenzdiagrammen gibt es Spielraum zwischen zwei Extremen /Jacobson 1992/:

- Zentralisierte Kontrollstruktur:
  Hierbei gibt es ein Objekt, das die Koordination aller anderen am Szenario beteiligten Objekte übernimmt. Das entsprechende Sequenzdiagramm wird in diesem Fall wie eine Gabel aussehen (Abb. 33, links).
- Dezentralisierte Kontrollstruktur:

Hierbei kennt jedes Objekt lediglich einen Teil der Beteiligten, so daß die Teilaufgaben von Objekt zu Objekt delegiert werden. Dadurch entsteht eine treppenförmige Diagrammstruktur (Abb. 33, rechts).

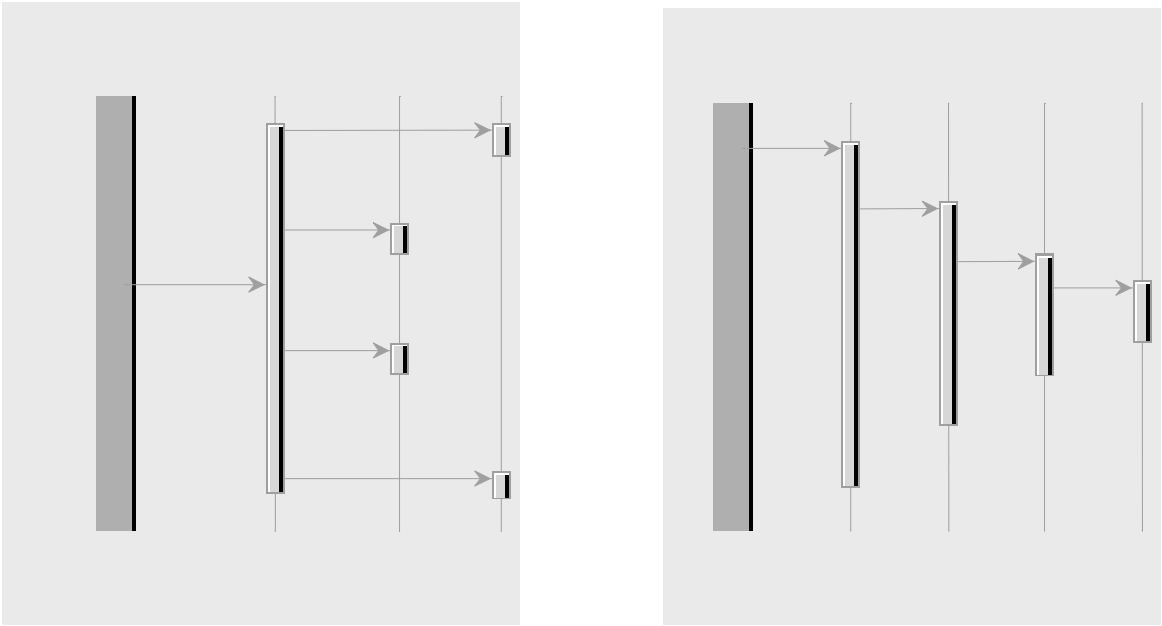


Abb. 33: Gabel- und treppenförmige Sequenzdiagramme

Botschaftsmodell prüfen und beschreiben

Beim Überprüfen des Sequenzdiagramms sollte auf folgende Fragen geachtet werden:

- Modelliert jedes Sequenzdiagramm wirklich ein fachliches Szenario?
- Gibt es Klassen, die kaum verwendet werden, so daß ihre Existenz fraglich ist?
- Gibt es Methoden, die nicht verwendet werden und auf die verzichtet werden könnte?

Für eine textuelle Beschreibung der Sequenzdiagramm wird das folgende Schema vorgeschlagen:

Szenario	⇨ Zu welchem Anwendungsfall gehört das Sequenzdiagramm und welches spezielle Szenario beschreibt es? [**Hier** klicken und Text eingeben]
Annahmen und Restriktionen	⇨ Liegen dem gestalteten Sequenzdiagramm spezielle Annahmen oder Restriktionen zugrunde? [**Hier** klicken und Text eingeben]
Erkenntnisse und Konsequenzen	⇨ Was haben Sie bei der Gestaltung des Sequenzdiagramms erkannt, welche Konsequenzen haben Sie speziell in Hinsicht auf die Gestaltung von Klassen und die Zuordnung von Methoden daraus gezogen? [**Hier** klicken und Text eingeben]
Offene Fragen	⇨ Welche Gestaltungsentscheidungen stehen noch aus? [**Hier** klicken und Text eingeben]
Bearbeitungsstatus	⇨ in Bearbeitung, fertig, abgenommen, etc. [**Hier** klicken und Text eingeben]

Abb. 34: Beschreibungsschema für Sequenzdiagramme /objectiF 4.5, microTOOL GmbH/

Ergebnisse

Die fachlich relevanten Szenarien sollten mit Sequenzdiagrammen beschrieben und entsprechend dokumentiert sein.

2.2.3 Implementierung

Beim Implementieren von Komponenten werden die entwickelten und bereits einzeln getesteten Klassen integriert. Wie auf der System-Ebene müssen auch für die Komponenten-Integration Subsysteme definiert werden. Nach der Integration muß im Rahmen des Komponententests der Nachweis erbracht werden, daß die integrierten Klassen fehlerfrei arbeiten und die von der Komponente erwarteten Dienste erbringen. Die nächste Abbildung faßt die durchzuführenden Aktivitätstypen zusammen.

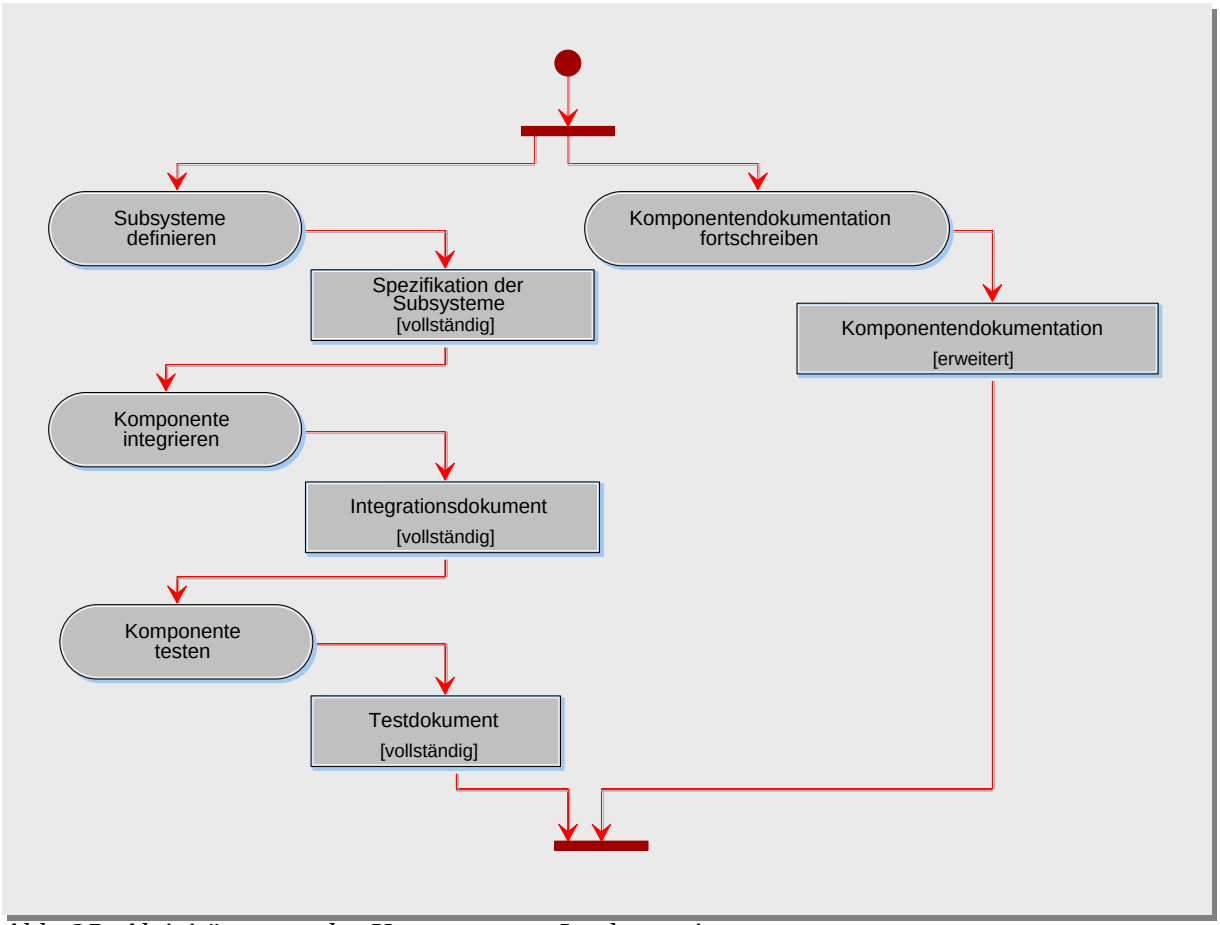


Abb. 35: Aktivitätstypen der Komponenten-Implementierung

Die Komponentendokumentation muß natürlich wieder weitergepflegt werden. Außerdem müssen erneut Subsysteme definiert werden. Im folgenden konzentrieren wir uns auf die Beschreibung der neu hinzugekommenen Aktivitätstypen.

2.2.3.1 Komponente integrieren

Zweck

Das Ziel dieses Aktivitätstyps ist es, die zu einer Komponente gehörenden Klassen zu integrieren und deren Zusammenwirken zu prüfen.

Beteiligte

Ausführend: Softwareentwickler, Qualitätsprüfer
Mitwirkend: Softwaredesigner

Vorgehen

Für die Integration muß zunächst eine Strategie festgelegt werden. Grundsätzlich werden vier Typen von Integrationsstrategien unterschieden /Pagel, Six 1994/:

- *nicht-inkrementell:* Bei dieser Integrationsstrategie werden alle oder eine größere Anzahl von Klassen gleichzeitig integriert. Der Vorteil dieser Strategie ist, daß keine zusätzlichen Testtreiber und Platzhalter benötigt werden. Als nachteilig erweist sich jedoch die schwere Lokalisierbarkeit von Fehlern.
- *inkrementell:* Hier werden die Klassen in einzelne oder sehr kleine Gruppen schrittweise integriert. Das hat den Vorteil, daß Klassen integriert werden können sobald sie fertiggestellt und geprüft sind. Außerdem sind Fehler leichter zu lokalisieren. Nachteilig ist, daß viele Testtreiber und Platzhalter benötigt werden.
- *testzielorientiert:* Bei dieser Strategie werden, ausgehend von den Testzielen, Testfälle erstellt. Die Testfälle werden abgearbeitet, indem die jeweils benötigten Klassen integriert werden.
- *vorgehensorientiert:* Die Integrationsreihenfolge wird bei dieser Strategie aus der Systemarchitektur und vom zugrundeliegenden Vorgehensmodell abgeleitet.

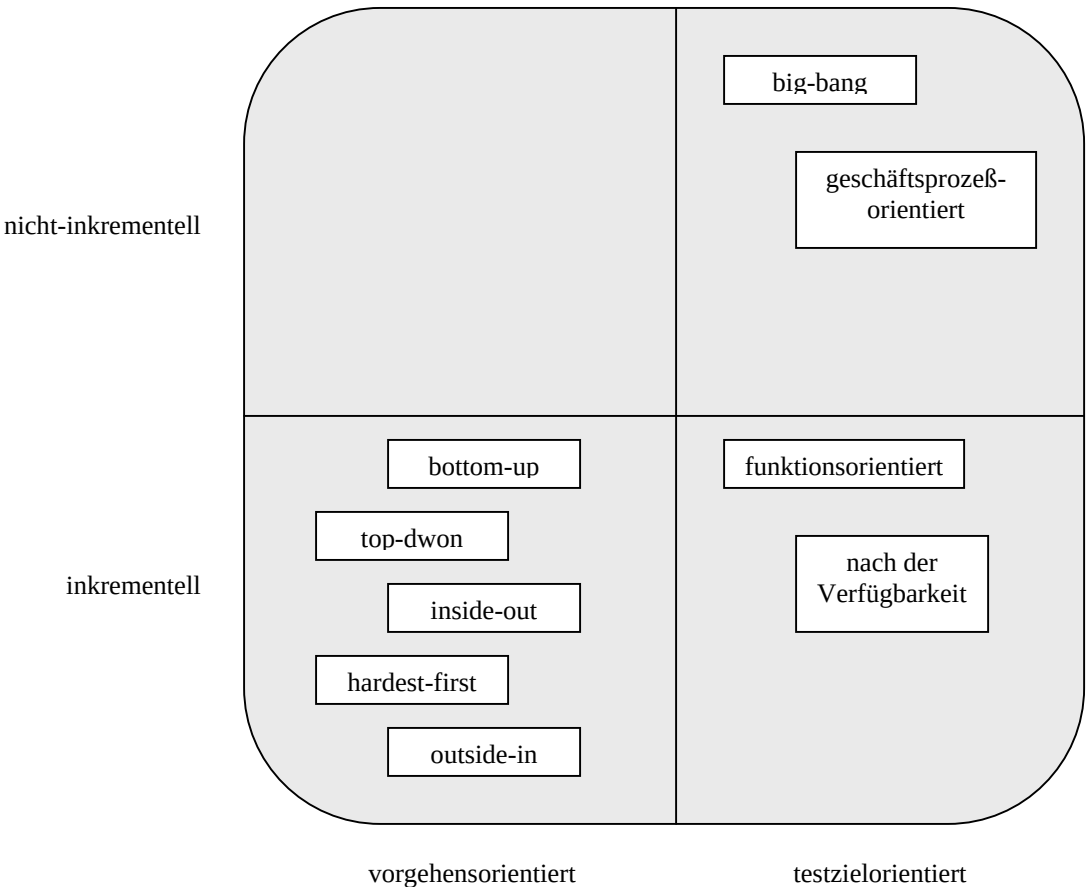


Abb. 36: Überblick über Integrationsverfahren /Pagel, Six 1994/

Abbildung 36 faßt bekannte Integrationsverfahren zusammen.

Bei der *big-bang Integration* werden alle Klassen einer Komponente gleichzeitig integriert. Der Integrationstest verläuft somit unstrukturiert ab, so daß Fehler schwer zu lokalisieren sind. Daher eignet sich diese Integrationsstrategie nur für Komponenten, die aus wenigen Klassen bestehen. Bei der *geschäftsprozessorientierten Integration* werden gleichzeitig diejenigen Klassen integriert, die einen Geschäftsprozeß abbilden.

Bei der *funktionsorientierten Integration* werden funktionale Testfälle definiert, anhand derer die Klassen integriert werden. Bei der *Integration nach Verfügbarkeit* werden Klassen sofort nachdem sie erfolgreich getestet worden sind, integriert. Die Integration endet, wenn alle Klassen implementiert sind. Pagel klassifiziert die *Integration nach Verfügbarkeit* als testzielorientiert. Eine vorgehensorientierte Klassifikation wäre aber auch denkbar.

Es gibt noch weitere Integrationsverfahren, die wir zur Vollständigkeit kurz erläutern wollen. Bei der *top-down Integration* werden die in der Systemarchitektur am weitesten oben stehenden Komponenten überprüft und danach schrittweise integriert. Bei der *bottom-up Integration* wird mit der Integration von den Komponenten begonnen, die sich am weitesten unten in der Systemarchitektur befinden. Bei der *outside-in Strategie* wird gleichzeitig auf der höchsten und der niedrigsten Schicht mit der Integration begonnen und in beiden Richtungen aufeinander zugearbeitet. Ausgehend von der mittleren Architekturschicht, werden bei der *inside-out Integration* nach und nach Komponenten aus höheren und tieferen Schichten hinzugefügt. Bei der *hardest-first Strategie* werden die kritischen und komplizierten Komponenten zuerst integriert.

Empfehlenswert ist es, von den definierten Anwendungsfällen auszugehen, die technisch durch die Komponente abgebildet werden und daraus Testfälle abzuleiten. Die Integration sollte sich an diesen Testfällen orientieren.

Ergebnisse

Die gewählte Integrationsstrategie sollte dokumentiert und begründet werden.

Hinweise und Tipps

Vertiefende Informationen zur Integration von Komponenten können in /Pressman 1997/ nachgelesen werden.

2.2.3.2 Komponente testen

Zweck

Das Ziel dieses Aktivitätstyps ist es, die Komponentenintegration zu testen. Dazu müssen eine Testumgebung und entsprechende Testfälle definiert werden. Auftretende Fehler müssen analysiert und behoben werden.

Beteiligte

Ausführend: Softwareentwickler, Qualitätsprüfer
Mitwirkend: Softwaredesigner

Vorgehen

Es sind folgende Schritte zu durchlaufen:

- Testfälle für den Integrationstest definieren,
- Integrationsumgebung erstellen,
- Integrationstest der Klassen durchführen,
- Fehler analysieren und beheben.

Testfälle für den Integrationstest definieren

Für den Integrationstest sollten jeweils solche Klassen in ihrem Zusammenwirken getestet werden, die zur Abwicklung eines Anwendungsfalls benötigt werden. Hierbei sind zwei Testformen zu berücksichtigen (Abb. 37):

- *Strukturtest:* Die „interne“ Sicht der an einem Anwendungsfall beteiligten Klassen ist zu überprüfen. Vom Typ her handelt es sich hierbei um einen White-Box-Test, da der Quellcode der Klassen vorliegen muß.
- *Funktionstest:* Das Verhalten der beteiligten Klassen ist hinsichtlich der spezifizierten Interaktion mit den Akteuren zu testen.

Die nächste Abbildung faßt die bekannten Testverfahren zusammen.

Für Struktur- und Funktionstests müssen aus einem Anwendungsfall zunächst konkrete Testfälle für das Hauptszenario (positiver Durchlauf) und für die Nebenszenarien (bedingungsabhängige Abweichungen und Erweiterungen) abgeleitet werden. Weitere Testfälle ergeben sich aus den nicht-funktionalen Anforderungen. Für das Ermitteln von Testfällen sind auch die Sequenzdiagramme der Anwendungsfälle hilfreich.

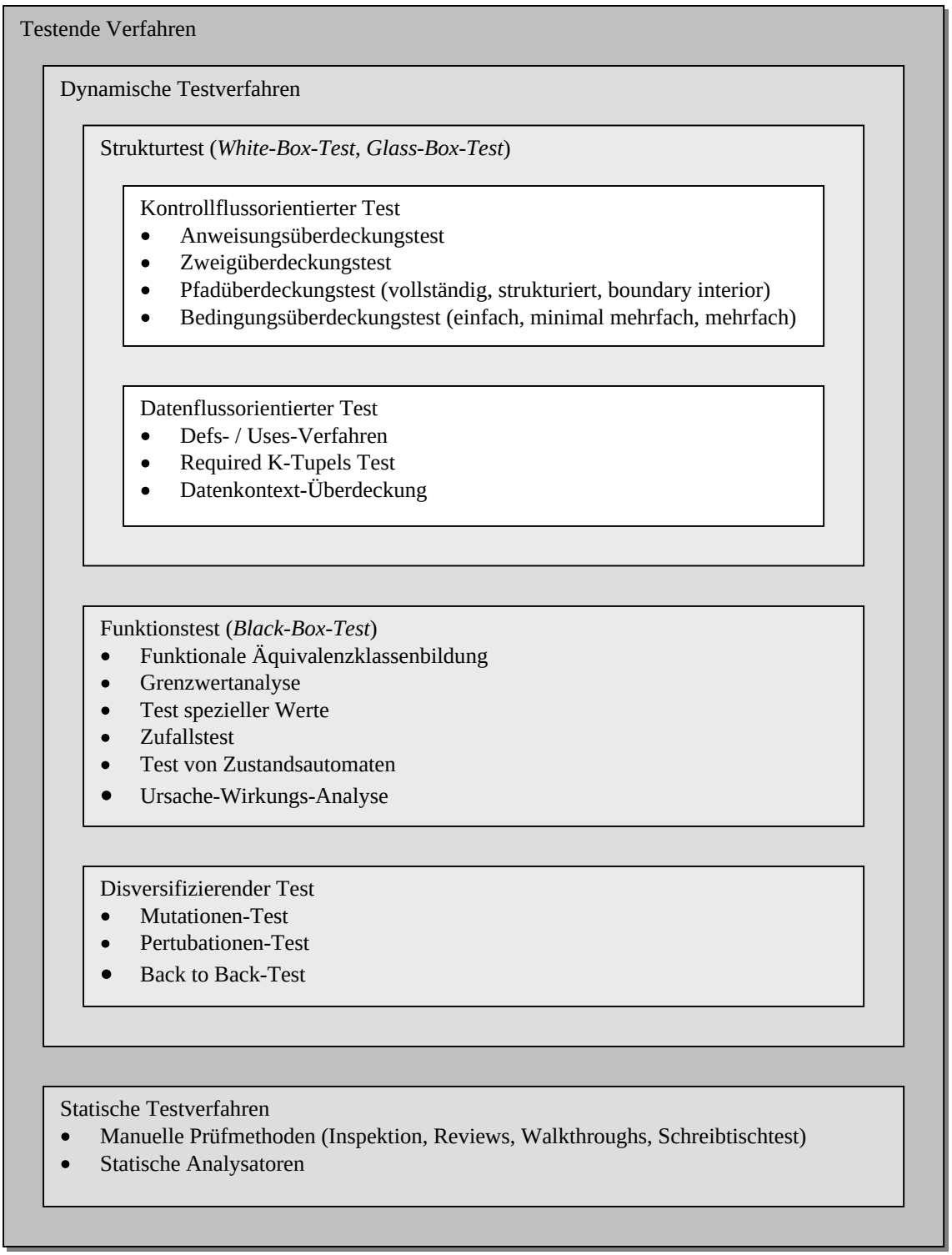


Abb. 37: Überblick der testenden Verfahren /Balzert 1998/

Damit Regressionstests ermöglicht werden können, sollten die Testfälle und die Ergebnisse der Tests mit dem folgenden Beschreibungsschema dokumentiert werden.

Testnummer	⇨ Da ein Testfall in der Regel mehrfach getestet wird, empfiehlt sich die Vergabe eines identifizierenden Merkmals pro Test. [**Hier** klicken und Text eingeben]		
Testdatum	⇨ Geben sie hier an, wann der Test stattgefunden hat. [**Hier** klicken und Text eingeben]		
Testteam	⇨ Führen Sie hier die Namen der am Test beteiligten Personen auf. [**Hier** klicken und Text eingeben]		
Anwendungsfall	⇨ Geben Sie hier den für die Komponente relevanten Anwendungsfall an, aus dem der Testfall abgeleitet wurde. [**Hier** klicken und Text eingeben]		
Szenario	⇨ Beschreiben Sie hier, auf welches Szenario sich der Testfall bezieht. [**Hier** klicken und Text eingeben]		
ID	⇨ Vergeben Sie hier ein eindeutiges Merkmal zur Identifikation des Testfalls, denn pro Szenario kann es mehrere Testfälle geben. [**Hier** klicken und Text eingeben]		
Zweck	⇨ Was ist der Zweck des Testfalls? Was soll konkret nachgewiesen werden? [**Hier** klicken und Text eingeben]		
Hinweis	⇨ Sammeln Sie hier alle Hinweise und Bemerkungen, die notwendig sind, um den Test anhand des beschriebenen Testfalls zu verstehen. [**Hier** klicken und Text eingeben]		
Lfd. Nummer	**Externe Bedingungen und Einflüsse** ⇨ Geben Sie alle äußeren Bedingungen an, die auf die Testeinheit wirken sollen. [**Hier** klicken und Text eingeben]		
Lfd. Nummer	**Testanweisungen** ⇨ Beschreiben Sie hier die vom Tester durchzuführenden Testschritte. [**Hier** klicken und Text eingeben]		
Lfd. Nummer	**Ausgangszustand**	**Erwartetes Ergebnis**	**Tatsächliches Ergebnis**
	⇨ Listen Sie hier alle Merkmale auf, die den Zustand beschreiben, indem sich das System zu Beginn des Tests befindet.	⇨ Geben Sie hier die Merkmale des Zustands an, in dem Sie das System nach dem Test erwarten.	⇨ Dokumentieren Sie den tatsächlich erreichten Zustand.
	[**Hier** klicken]	**Hier** klicken]	[**Hier** klicken]
Lfd. Nummer	**Eingabedaten**	**Erwartetes Ergebnis**	**Tatsächliches Ergebnis**
	⇨ Listen Sie hier alle Eingabedaten auf.	⇨ Führen Sie in dieser Liste die erwarteten Ausgaben auf.	⇨ Dokumentieren Sie die tatsächlichen Ausgaben des Testfalls.
	[**Hier** klicken]	**Hier** klicken]	[**Hier** klicken]
Zusammen-fassung	⇨ Geben Sie hier, sofern erforderlich, eine Zusammenfassung von Testverlauf und Ergebnissen. [**Hier** klicken und Text eingeben]		
Bewertung/ Empfehlung	☐ fehlerfreier Ablauf ☐ leichte Fehler	☐ Ergebnis akzeptieren, Test abschließen	

	☐ schwere Fehler ☐ fatale Fehler	☐ Korrektur ohne Testwiederholung ☐ Korrektur und Testwiederholung
Hinweise zur Fehlerbehebung	⇨ Sammeln Sie hier alle Hinweise und Bemerkungen, die bei der Ermittlung der Fehlersuche und bei der Fehlerbehebung von Nutzen sein können. [**Hier** klicken und Text eingeben]	

Abb. 38: Beschreibungsschema für Testfälle, Testanweisungen und Testprotokolle /Pressman 1997/

Integrationsumgebung erstellen

Die Integration sollte in einem speziell zu diesem Zweck für das Projektteam definierten Arbeitsbereich stattfinden. Die passenden Versionen der zu integrierenden Bausteine sollten bereitgestellt werden. Sollten Klassen mit anderen Komponenten interagieren, so sind entsprechende Platzhalter zur Simulation dieser Komponente bereitzustellen.

Integrationstest der Klassen durchführen

Pro Testfall ist die aktuelle Testumgebung und der Testverlauf zu protokollieren. Erkannte Fehler sollten dokumentiert und bewertet werden. Zur Bewertung können folgende Fragen hilfreich sein: Wie schwerwiegend ist der Fehler? Wie dringlich ist die Behebung des Fehlers?

Fehler analysieren und beheben

Die erkannten Fehler sollten einer Analyse unterzogen werden. Hierbei ist nach Ursache und Korrekturmaßnahmen zu suchen. Nachdem die Fehlerursache gefunden ist, sollte seine Behebung veranlaßt werden.

Ergebnisse

Die Testfälle und der Verlauf sollten mit dem entsprechenden Beschreibungsschema dokumentiert werden.

Hinweise und Tipps

Es sollte eine statische Auswertung der ermittelten Fehler unternommen werden, damit prinzipielle Schwächen aufgedeckt werden können und entsprechende Gegenmaßnahmen unternommen werden können.

2.2.4 Operationeller Einsatz

Mit der Inbetriebnahme des Systems beginnt sowohl für das System als auch für die enthaltenen Komponenten der operationelle Einsatz, bei dem die Aspekte der Erprobung, Nutzung und Revision betrachtet werden.

Die nächste Abbildung zeigt eine Übersicht über die durchzuführenden Aktivitätstypen.

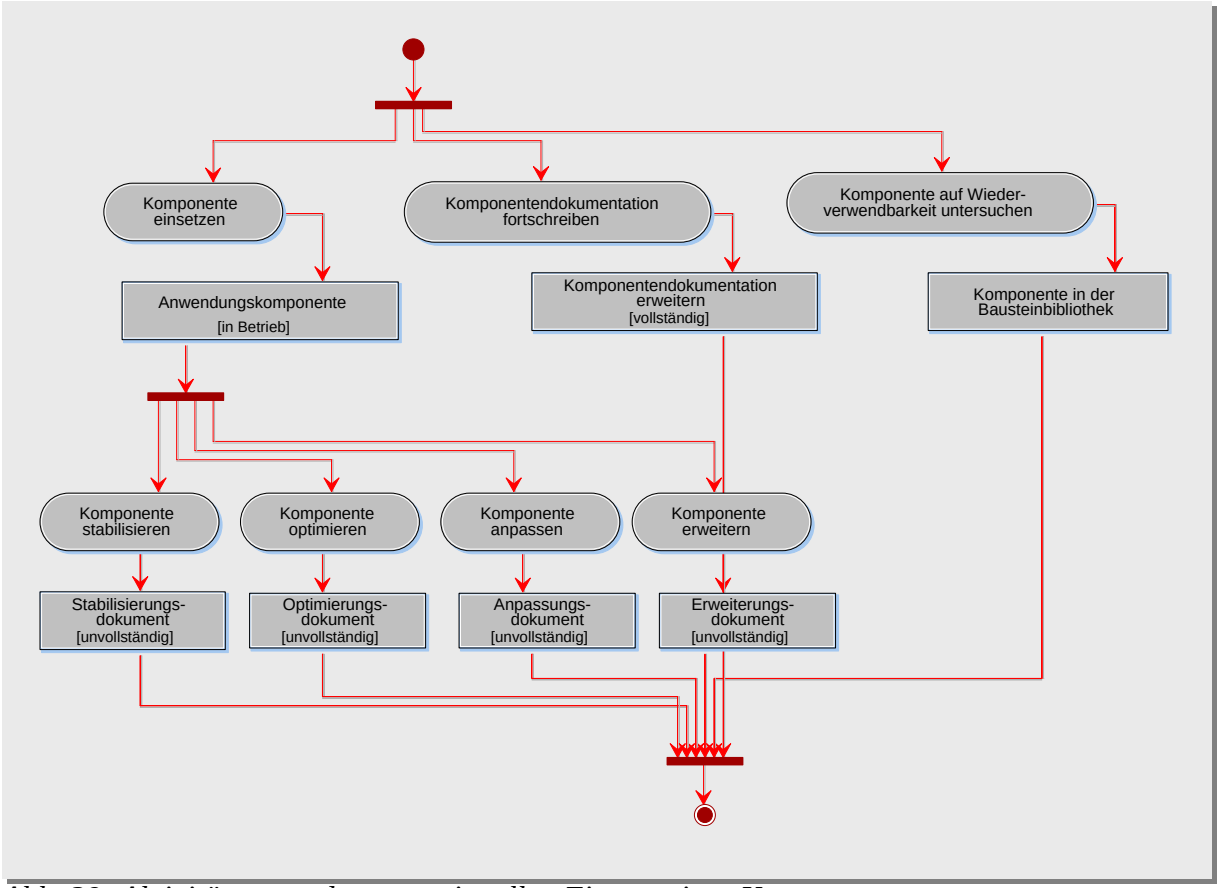


Abb. 39: Aktivitätstypen des operationellen Einsatz einer Komponente

Zu den bekannten Aktivitätstypen „Komponentendokumentation fortschreiben“ kommt der Aktivitätstyp „Komponente auf Wiederverwendbarkeit untersuchen“ hinzu. Hierbei wird untersucht, ob die Komponente für die Wiederverwendung geeignet ist.

Das System wird in der Regel Fehler aufweisen. Die Anwender werden Optimierungs-, Anpassungs- und Erweiterungswünsche äußern. Die Betriebsfehler und die geäußerten Wünsche beziehen sich auf eine oder mehrere Komponenten des Systems. Damit setzen sich die auf der System-Ebene genannten Aktivitätstypen auf die entsprechenden Komponenten fort. Es sind folgende Aktivitätstypen durchzuführen, um die geforderten Korrekturen und Verbesserungen an einer Komponente vornehmen zu können:

- Komponente stabilisieren,
- Komponente optimieren,
- Komponente anpassen,
- Komponente erweitern.

Eine Komponente zu stabilisieren, heißt die aufgetretenen Fehler zu dokumentieren und zu beheben. Beim Optimieren wird die Systemleistung der Komponente verbessert. Die Anpassung des Systems an ein verändertes Umfeld und die Erweiterung der Systemfunktionalität erfolgt ebenfalls durch die Modifikation der entsprechenden Komponenten. Für jeden der genannten Aktivitätstypen ist zu entscheiden, ob die Änderungen zu einem neuen Komponentenzyklus führen. Beschreibungsschemas für die Dokumentation von Fehlern und Verbesserungen wurden bereits auf der System-Ebene (Abschnitt: Operationeller Einsatz) vorgestellt. Der Ablauf der genannten Aktivitätstypen ist analog zu

den entsprechenden Aktivitätstypen auf der System-Ebene, so daß auf eine detaillierte Beschreibung verzichtet wird.

2.3 Klassen-Ebene

Wenn in diesem Abschnitt im Rahmen einer Komponente von Klassen gesprochen wird, so sind damit die der Komponente zugeordneten Klassen gemeint.

2.3.1 Analyse

Ein Klassen-Zyklus wird in der Regel auf der Komponenten-Ebene angestoßen. Die Analyse einer Klasse beginnt mit ihrer Identifikation während des Komponenten-Entwurfs. Im Rahmen der Analyse wird die Funktionalität der identifizierten Klasse beschrieben. Außerdem wird die Bausteinbibliothek nach wiederverwendbaren Klassen durchsucht. Es wird eine Klassendokumentation angelegt, die alle im Rahmen der Klasse entstehenden Ergebnisse beinhaltet .

Die nächste Abbildung zeigt eine Übersicht über die durchzuführenden Aktivitätstypen.

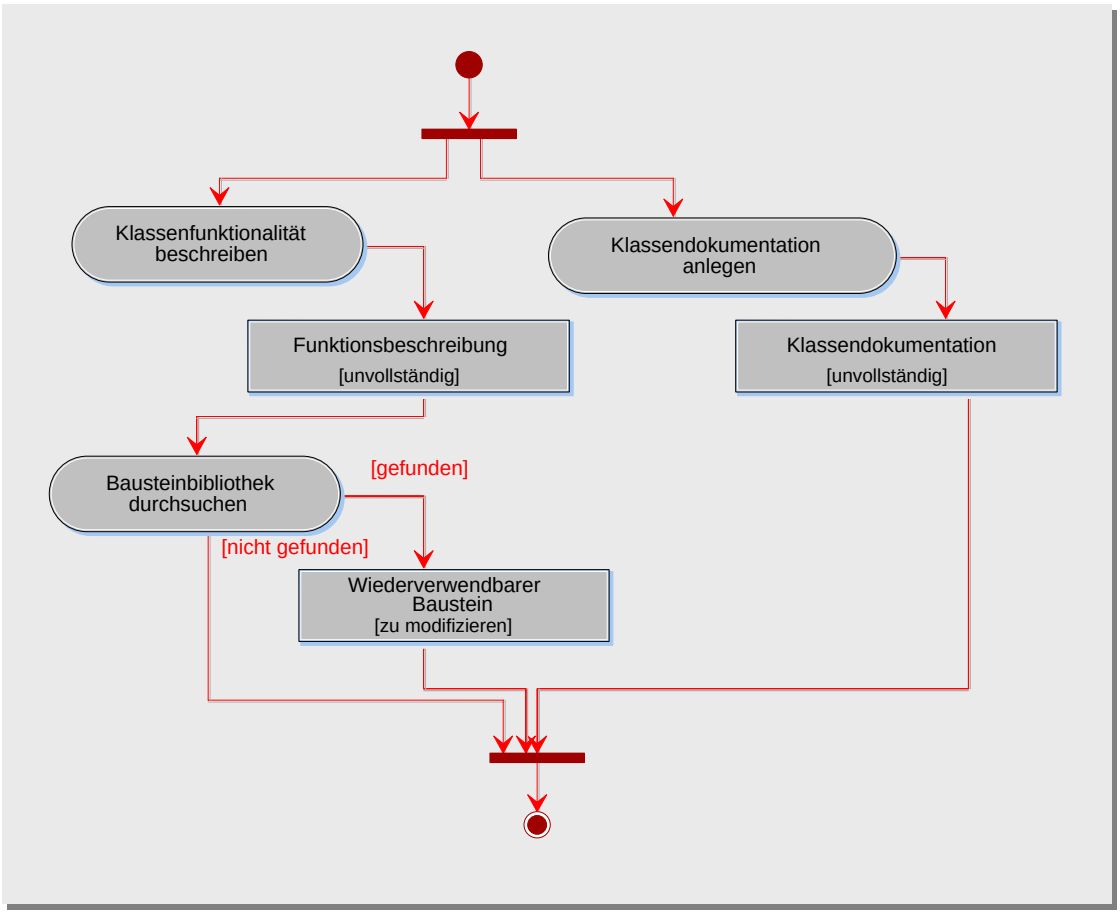


Abb. 40: Aktivitätstypen der Klassen-Analyse

2.3.1.1 Klassenfunktionalität beschreiben

Zweck

Der Zweck und die Leistung der Klasse werden beschrieben.

Beteiligte

Ausführend: Softwaredesigner
Mitwirkend: Systemanalytiker

Vorgehen

Die Klassen sollte gegebenenfalls mit entsprechenden Stereotypen versehen werden und anhand des folgenden Beschreibungsschemas dokumentiert werden.

Zweck und Leistung	⇨ Machen Sie hier eine Aussage darüber, wofür die Klasse verantwortlich ist bzw. welches Wissen sie bereitstellt. Stellen Sie kurz dar, an welchen Anwendungsfällen die Klasse beteiligt ist und welche Aufgaben ihr dabei zukommen. Gab es alternative Gestaltungsmöglichkeiten? Begründen Sie Ihre Entwurfsentscheidung. [**Hier** klicken und Text eingeben]
Art	⇨ Wenn Sie keine Stereotypen für Analyseklassen definiert haben, so sollten Sie hier dokumentieren, zu welchem Typ – den Entity-, Boundary- und Control-Klassen – die gefundene Klasse gehört. [**Hier** klicken und Text eingeben]
Muster	⇨ Möglicherweise haben Sie Analyse- oder Designmuster angewandt. Dann nennen Sie hier die Muster, auf die diese Klasse zurückgeht, und nach Bedarf auch die Quelle der Muster. Alternativ können Sie diesen Zusammenhang auch mit benutzerdefinierten Eigenschaften dokumentieren. [**Hier** klicken und Text eingeben]
Musterklassen	⇨ Nennen Sie hier – um die Modellierung nachvollziehbar zu machen – die Rollennamen, die die Klasse innerhalb der verwendeten Muster besitzt (Beispiel: *Component*). Auch dies können Sie alternativ natürlich mit benutzerdefinierten Eigenschaften tun. [**Hier** klicken und Text eingeben]
Schnittstelle	⇨ Beschreiben Sie die Schnittstelle der Klasse bestehend aus öffentlichen Attributen und Methoden, und begründen Sie Ihre Entwurfsentscheidungen. [**Hier** klicken und Text eingeben]

Abb. 41: Beschreibungsschema für Klassen /objectiF 4.5, microTOOL GmbH/

Ergebnisse

Das Beschreibungsschema in der Abbildung 41 sollte ausgefüllt werden.

2.3.1.2 Bausteinbibliothek durchsuchen

Zweck

Durch die Wiederverwendung von Bausteinen können Kosten gespart und die Produktqualität erhöht werden. Daher sollte die Bausteinbibliothek nach Klassen durchsucht werden, die über

die gleichen Dienste wie die zu analysierenden Klasse verfügen, so daß eine Wiederverwendung ermöglicht wird.

Beteiligte

Ausführend: Softwaredesigner
Mitwirkend: Systemanalytiker

Vorgehen

Wenn eine Klasse wiederverwendet werden kann, so muß sie dem Klassenmodell hinzugefügt werden. Die notwendigen Modifikationen sollten durchgeführt werden und die Beziehungen zu den bereits vorhandenen Klassen revidiert werden.

Ergebnisse

Bei einer erfolgreichen Suche liegt eine wiederverwendbare Klasse mit möglichen Modifikationen vor.

2.3.2 Entwurf

Beim Klassen-Entwurf wird die innere Struktur (Attribute und Methoden) einer Klasse definiert. Für Klassen mit komplexen Lebenszyklen, wo die Objekte fachlich interessante Zustandsübergänge haben, sollte ein Zustandsmodell entwickelt werden. Zustandsmodelle erleichtern die Analyse des dynamischen Verhaltens von Klassen. Natürlich muß die Klassendokumentation wieder erweitert und verfeinert werden. Die nächste Abbildung zeigt die durchzuführenden Aktivitätstypen.

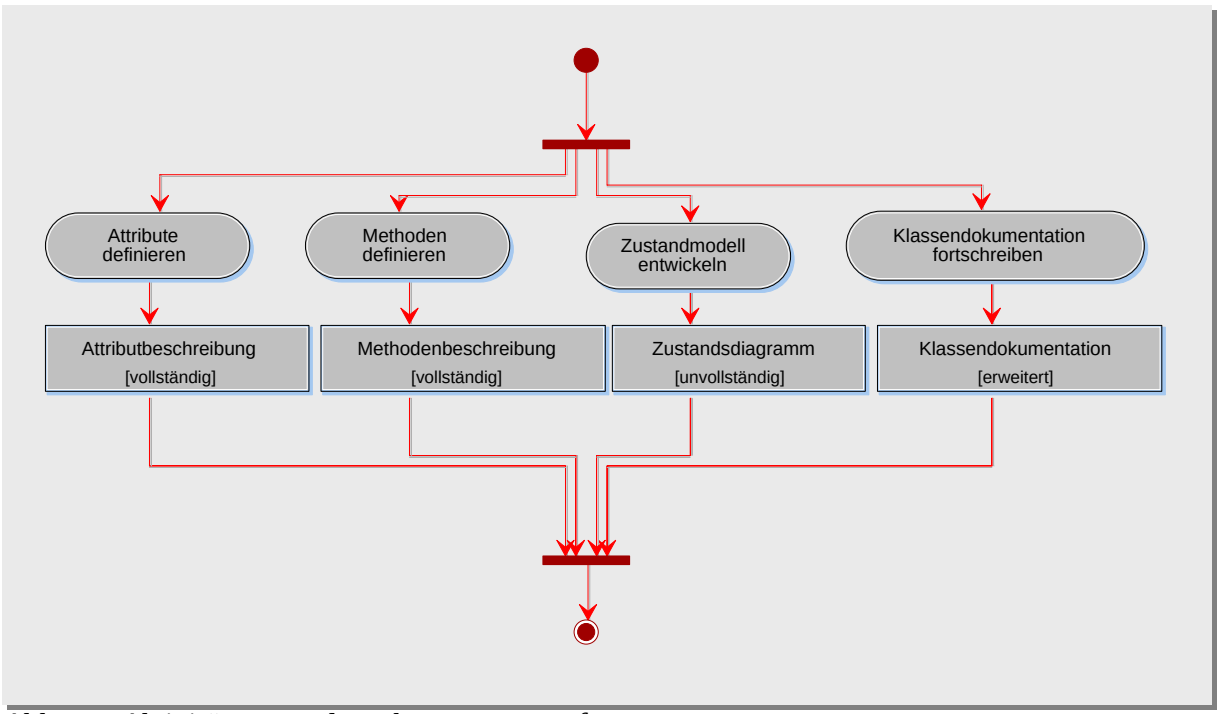


Abb. 42: Aktivitätstypen des Klassen-Entwurfs

2.3.2.1 Attribute definieren

Zweck

Die passiven Merkmale der zu entwerfenden Klasse werden bei diesem Aktivitätstyp identifiziert und dokumentiert.

Beteiligte

Ausführend: Softwaredesigner
Mitwirkend: Systemanalytiker

Vorgehen

Attribute beschreiben die Zustände einer Klasse. Um Kandidaten für Attribute zu finden, können folgende Punkte hilfreich sein:

- Welche Eigenschaften benötigt eine Klasse, um ihre Aufgaben realisieren zu können?
- Welche Zustände kann eine Klasse annehmen?

Zum Beispiel kann die Textanalyse wieder eingesetzt werden, um Attribute zu identifizieren. Hilfreich ist hierbei, wenn Benutzeroberflächen (z.B. in Form eines Prototypen) entworfen worden sind. Denn alle Eingabefelder oder Kontrollfelder sind Kandidaten für Attribute.

Schwierig ist zu klären, ob ein Attributkandidat wirklich eine Eigenschaft darstellt, oder ob es sich dabei eher um eine neue Klasse handelt. Um das zu Entscheiden, hilft es zu klären, ob das Attribut selbst durch Eigenschaften beschrieben werden muß. Ist das der Fall, so handelt es sich bei dem gefundenen Attribut um einen Klassen-Kandidaten. Bevor ein Attribut angelegt wird, sollten folgende Aspekte geprüft werden:

- Ist das Attribut der richtigen Klasse zugeordnet?
- Wird durch das Attribut ein Zustand beschrieben, der für das Anwendungssystem relevant ist?
- Läßt sich der Wert des Attributs aus anderen Attributwerten ermitteln? Wenn ja, kann auf das Attribut verzichtet werden?
- Nimmt das Attribut auch sinnvolle Werte an?

Als letzter Schritt sollte dem Attribut gegebenenfalls ein Stereotyp und eine entsprechende Sichtbarkeit, wie z.B. öffentlich oder privat, zugeordnet werden. Falls nötig, kann ein Attribut auch mit dem folgenden Beschreibungsschema dokumentiert werden.

Zweck und Leistung	⇨ Was wird mit dem Attribut gemacht? Beschreiben Sie, welchen primären Verwendungszweck das Attribut erfüllt. [**Hier** klicken und Text eingeben]
Art	⇨ Klassifizieren Sie hier das Attribut. Beschreibt das Attribut einen Zustand oder wird es benötigt, um die Dienste einer anderen Klasse benutzen zu können. [**Hier** klicken und Text eingeben]

Maßeinheit	⇨ In welcher Maßeinheit werden die Werte des Attributs gemessen?
	[**Hier** klicken und Text eingeben]
Wertebereich	⇨ Spezifizieren Sie hier den Wertebereich, aus dem die Attributwerte stammen können.
	[**Hier** klicken und Text eingeben]
Genauigkeit	⇨ Machen Sie hier Angaben zur Genauigkeit der Attributwerte, zulässige Runden etc.
	[**Hier** klicken und Text eingeben]
Default-Wert	⇨ Gibt es einen festen Vorgabewert für das Attribut? Dokumentieren Sie ihn hier.
	[**Hier** klicken und Text eingeben]
Restriktionen	⇨ Beschreiben Sie hier Einschränkungen und Randbedingungen, die beim Anlegen und Zugreifen auf das Attribut zu berücksichtigen sind.
	[**Hier** klicken und Text eingeben]
Zusammenhänge /Abhängigkeiten	⇨ Stellen Sie hier eventuell vorhandene Abhängigkeiten oder Zusammenhänge mit anderen Attributen dar.
	[**Hier** klicken und Text eingeben]

*Abb. 43: Beschreibungsschema für Attribute /objectiF 4.5, microTOOL Gmb*H/

Ergebnisse

Als Resultat sollte eine Beschreibung der gefundenen Attribute vorliegen.

2.3.2.3 Methoden definieren

Zweck

Die aktiven Merkmale der zu entwerfenden Klasse werden bei diesem Aktivitätstyp identifiziert und dokumentiert.

Beteiligte

Ausführend: Softwaredesigner
Mitwirkend: Systemanalytiker

Vorgehen

Methoden stellen das Verhalten einer Klasse dar. Das Finden von Methoden geschieht parallel zur Suche von Attributen. Folgende Aspekte helfen beim Finden von Methoden:

- Welches Verhalten muß eine an einem Anwendungsfall beteiligte Klasse aufweisen, damit der Anwendungsfall abgewickelt werden kann?
- Welches Verhalten benötigt eine Klasse zur Beschaffung, Erzeugung und Auswertung von Daten?
- Welches Verhalten wird benötigt, um auf Ereignisse reagieren zu können?

Bei der Identifikation von Methoden können Sequenz- und Zustandsdiagramme, auf die wir weiter unten noch eingehen werden, sehr hilfreich sein. Aber auch die Textanalyse kann wieder eingesetzt werden. Hierbei ist nach Verben zu suchen, denn diese sind häufig

Kandidaten für Methoden. Das Kategorisieren kann ebenfalls nützlich sein. Im folgenden sind mögliche Methoden-Kategorien aufgelistet /Yordon 1994/:

- einfache Methoden,
- Methoden in Verbindung mit Attributen,
- Methoden in Verbindung mit Aggregationen und Assoziationen,
- Methoden in Verbindung mit temporären Botschaftsbeziehungen,
- Methoden in Verbindung mit Zuständen.

Es sollte noch überprüft werden, ob die Methoden tatsächlich benötigt werden, und ob sie den richtigen Klassen zugeordnet sind. Zuletzt kann der Methode ein Stereotyp und eine entsprechende Sichtbarkeit zugeordnet werden. Ergänzend kann auch eine verbale Beschreibung der Methode vorgenommen werden.

Zweck und Leistung	⇨ Beschreiben Sie, welche primäre Leistung die Methode bezogen auf einen Anwendungsfall erbringt. [**Hier** klicken und Text eingeben]
Art	⇨ Klassifizieren Sie hier die Methode. Handelt es sich zum Beispiel um eine reine Zugriffsmethode? Oder ist die Methode für Verarbeitung oder Steuerung zuständig? [**Hier** klicken und Text eingeben]
Parameter	⇨ Listen Sie hier die Methodenparameter auf und beschreiben Sie sie. [**Hier** klicken und Text eingeben]
Rückgabewert	⇨ Spezifizieren Sie hier den Rückgabewert, den die Methode liefert. [**Hier** klicken und Text eingeben]
Restriktionen	⇨ Welche Aufrufbeschränkungen sind zu beachten? [**Hier** klicken und Text eingeben]
Zusammenhänge/ Abhängigkeiten	⇨ Stellen Sie hier eventuell vorhandene Abhängigkeiten oder Zusammenhänge mit anderen Methoden dar. [**Hier** klicken und Text eingeben]

Abb. 44: Beschreibungsschema für Methoden /objectiF 4.5, microTOOL GmbH/

Ergebnisse

Als Resultat sollte eine Beschreibung der gefundenen Methoden vorliegen.

Hinweise und Tipps

Die entworfene Klasse sollte einer Überprüfung unterzogen werden. Im folgenden stellen wir einige Prüfschritte vor, die auf /Love 1991/, /Rumbaugh 1993/ und /Yourdon 1994/ beruhen:

- Kapselung prüfen:
  Ziel sollte es sein, daß die interne Struktur einer Klasse anderen verborgen bleiben sollte. Insbesondere sollten andere Klassen keinen direkten Zugriff auf die Attribute einer Klasse besitzen. Es sei denn, daß die Klasse öffentliche Attribute hat, was im Allgemeinen vermieden werden sollte.

- Kopplung prüfen:
  Ziel sollte es sein, daß eine Klasse hohe Kohäsion und niedrige Kopplung besitzt. Eine Klasse sollte nicht mit mehr als ca. 7 anderen unmittelbar zusammenarbeiten müssen, damit die Wartung erleichtert wird.

- Sichtbarkeit prüfen:
  Ziel sollte es sein, daß eine Klasse möglichst schmale Schnittstellen besitzt. Dazu sollten die Sichtbarkeitseigenschaften privat, öffentlich und geschützt geprüft werden und auf öffentliche Methoden soweit wie möglich verzichtet werden. Denn je weniger öffentliche Methoden eine Klasse hat, desto schmaler ist ihre Schnittstelle nach außen.

- Wiederverwendbarkeit prüfen:
  Je kleiner eine Methode ist, desto besser kann sie wieder verwendet werden. Daher sollten umfangreiche Methoden möglichst aufgeteilt werden. Denn kleine Methoden sind leicht zu überschauen und lösen Teilprobleme komplizierterer Methoden.

- Verständlichkeit prüfen:
  Optimal ist es, wenn eine Klasse nicht mehr als sechs bis sieben öffentliche Methoden hat. Jede Methode sollte nicht mehr als zwei bis drei Attribute behandeln. Daher sollten umfangreiche Klassen möglichst aufgeteilt werden, damit die Wartung erleichtert werden kann.

2.3.2.3 Zustandsmodell entwickeln

Zweck

Um das dynamische Verhalten einer Klasse zu beschreiben, wird von UML das Zustandsdiagramm angeboten. Ein Zustandsdiagramm beschreibt die Zustände, die die Objekte einer Klasse im Verlauf ihrer Existenz annehmen können. Es zeigt, welche Ereignisse (z.B. empfangene Botschaften) zu einem Zustandswechsel führen. Außerdem werden die Aktionen, d.h. gesendete Botschaften, die mit einem Zustandswechsel verbunden sind, dargestellt. Zustandsdiagramme sind auch bei der Identifikation von Methoden sehr hilfreich.

Beteiligte

Ausführend: Systemanalytiker
Beratend: Softwaredesigner

Vorgehen

- Zustandsmodell erstellen,
- Zustandsmodell überprüfen und beschreiben.

Zustandsmodell erstellen

Zustandsdiagramme haben einen exemplarischen Charakter, d.h. Zustandsdiagramme nur dann, wenn es sich aufgrund der Komplexität lohnt. Für eine Klasse sollte ein Zustandsdiagramm entwickelt werden, wenn:

- ein Objekt der Klasse auf identische Botschaften abhängig vom aktuellen Zustand unterschiedlich reagiert,

- eine Methode nur dann ausgeführt werden kann, wenn das Objekt in einem besonderen Zustand vorliegt.

Bei der Gestaltung des Zustandsdiagramms sollte von dem Lebenszyklus eines Objekts der betrachteten Klasse ausgegangen werden und „durchlebte“ Zustände gesammelt werden. Ausgehend von den gefundenen Zuständen, sollte nun nach Ereignissen und Aktionen gesucht werden, die zu einem Zustandswechsel beitragen. Hierbei werden womöglich neue Methoden gefunden. Zustände die eine textuelle Beschreibung erfordern, können mit dem folgenden Beschreibungsschema dokumentiert werden.

Semantik	⇨ Beschreiben Sie hier die fachliche Bedeutung des Zustands. [**Hier** klicken und Text eingeben]
Zustandsattribut	⇨ Listen Sie hier alle die Attribute mit ihren Werten auf, die den Zustand beschreiben. Wählen Sie dazu die Form: Attributname = Attributwert [**Hier** klicken und Text eingeben]
Aktivitäten	⇨ Sind mit dem Eintritt (entry), dem Verweilen (do) und dem Verlassen (exit) des Zustands spezielle Aktionen des Objekts, d.h. die Durchführung von Methoden, verbunden? Beschreiben Sie sie in der Form: Entry / Aktion des Objekts Do / Aktion des Objekts Exit / Aktion des Objekts Ereignisabhängige Aktionen können Sie so festhalten: On / Ereignis: Aktion des Objekts [**Hier** klicken und Text eingeben]
Häufigkeit	⇨ Handelt es sich um einen eher seltenen Zustand oder geht jedes Objekt ein- oder mehrfach in diesen Zustand über? [**Hier** klicken und Text eingeben]
Verweildauer	⇨ Wie lange bleibt ein Objekt durchschnittlich in diesem Zustand? [**Hier** klicken und Text eingeben]

Abb. 45: Beschreibungsschema für Zustände /objectiF 4.5, microTOOL GmbH/

Zustandsmodell überprüfen und beschreiben

Zur Überprüfung des Zustandsmodells helfen folgende Aspekte:

- Sind für alle Klassen, die keinen trivialen Lebenszyklus besitzen, Zustandsdiagramme angelegt worden?
- Sind die entwickelten Zustandsdiagramme vollständig und korrekt?
- Haben die Zustandsdiagramme einen Anfangs- und Endzustand?

Wo es nützlich erscheint, können die Zustandsdiagramme textuell beschrieben werden. Die nächste Abbildung zeigt ein entsprechendes Beschreibungsschema.

Zweck und Leistung	⇨ Aus welchem Grund wurde das Diagramm entwickelt, welche Fragen soll es beantworten? [**Hier** klicken und Text eingeben]
Art	⇨ Klassifizieren Sie den Lebenszyklus: Ist er eher linear oder zyklisch? [**Hier** klicken und Text eingeben]
Annahmen und	⇨ Liegen dem gestalteten Objektlebenszyklus spezielle Annahmen oder Restriktionen

Restriktionen	zugrunde? [**Hier** klicken und Text eingeben]
Erkenntnisse und Konsequenzen	⇨ Was haben Sie bei der Gestaltung des Objektlebenszyklus erkannt, welche Konsequenzen haben Sie speziell in Hinsicht auf die Gestaltung und Zuordnung von Methoden daraus gezogen? [**Hier** klicken und Text eingeben]
Offene Fragen	⇨ Welche Gestaltungsentscheidungen stehen noch aus? [**Hier** klicken und Text eingeben]
Bearbeitungsstatus	⇨ in Bearbeitung, fertig, abgenommen, etc. [**Hier** klicken und Text eingeben]

Abb. 46: Beschreibungsschema für Zustandsdiagramme /objectiF 4.5, microTOOL GmbH/

Ergebnisse

Falls entschieden wurde, daß für die Klasse ein Zustandsdiagramm notwendig ist, so sollte als Ergebnis ein Zustandsdiagramm mit der entsprechenden Dokumentation vorliegen.

Hinweise und Tipps

Hier folgen einige Entwurfsprinzipien für Klassen, deren konsequente Einhaltung zu hohe Wiederverwendbarkeit und gute Wartbarkeit führen. Für genaue Erläuterungen verweisen wir auf die entsprechende Literatur.

- Das Offen-geschlossen-Prinzip (*open-closed principle*) /Meyer 1990/:
  Eine Klasse sollte offen für Erweiterungen, aber geschlossen gegenüber Änderungen sein. Offen bedeutet, daß das Verhalten einer Klasse so ergänzt werden kann, daß sie neuen Anforderungen gerecht werden kann. Geschlossen bedeutet, daß der vorhandene Quell-Code einer Klasse nicht modifiziert zu werden braucht, wenn neue Anforderungen abgewickelt werden müssen. Leider ist das Prinzip in der Praxis nicht so einfach einzuhalten.

- Das Prinzip der Liskov-Substitution (*Liskov substitution principle*) /Martin March 1996/ :
  Überall, wo ein Objekt einer Basisklasse erwartet wird, kann auch ein Objekt der entsprechenden abgeleiteten Klassen verwendet werden. Dieses Prinzip ermöglicht den mächtigen Mechanismus der Polymorphie.

- Das Prinzip der Trennung von Schnittstellen (*interface segregation principle*) /Martin August 1996/:
  Klassen sollten niemals gezwungen werden, von Schnittstellen abzuhängen, die sie selbst nicht benutzen. Sollte das nicht der Fall sein, so ist eine gute Wartbarkeit nicht gewährleistet.

2.3.3 Implementierung

Gegenstand der Implementierung ist es, eine spezifizierte Klasse zu kodieren, zu kompilieren und zu testen. Die Klassendokumentation muß natürlich fortgeschrieben werden. Die nächste Abbildung faßt die entsprechenden Schritte zusammen.

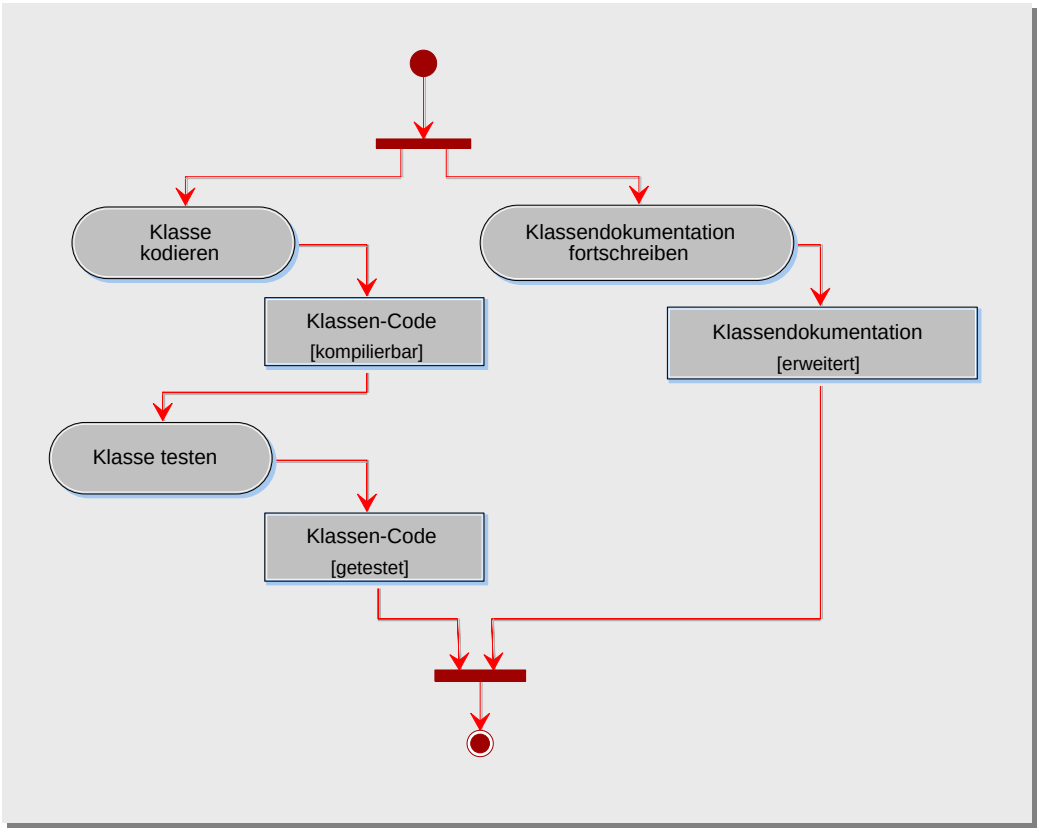


Abb. 47: Aktivitätstypen der Klassen-Implementierung

2.3.3.1 Klasse kodieren

Zweck

Die Aktivität dieses Typs umfaßt die Entwicklung des Quellcodes einer Klasse sowie das Kompilieren und Debuggen.

Beteiligte

Ausführend: Softwareentwickler
Mitwirkend: Softwaredesigner

Vorgehen

Grundlage für diesen Aktivitätstyp ist das Klassenmodell und die entsprechende Beschreibung der zu implementierenden Klasse. In der Regel wird im Verlauf der Implementierung das Klassenmodell im Detail modifiziert, da z.B. neue Modellelemente hinzu kommen werden.

Bei der Programmierung sollten folgende Prinzipien eingehalten werden /Balzert 1996/:

- *Prinzip der Verbalisierung:* Verbalisierung soll dazu dienen, die Ideen und Konzepte des Programmierers im Programm möglichst gut sichtbar zu machen und zu dokumentieren. Eine gute Verbalisierung kann z.B. durch aussagekräftige Namensgebungen und geeignete

Kommentare erreicht werden. Damit werden Code-Reviews und Modifikationen erleichtert.

- *Prinzip der problemadäquaten Datentypen:* Die für ein Problem benötigten Daten- und Kontrollstrukturen sollten in der programmiersprachlichen Lösung möglichst unverfälscht widergespiegelt werden. Damit wird die Verständlichkeit und Lesbarkeit des Programms erleichtert.
- *Prinzip der Verfeinerung:* Dieses Prinzip dient zur Strukturierung des Programms durch Abstraktionsebenen, die das zu erreichende Ziel beschreiben. Dabei wird zunächst eine Lösung mit abstrakten Daten und abstrakten Anweisungen formuliert, die schrittweise konkretisiert wird. Mit diesem Prinzip können die Entwicklungsentscheidungen besser nachvollzogen werden, so daß die Einarbeitung erleichtert wird.
- *Prinzip der integrierten Dokumentation:* Das entwickelte Programm sollte in geeigneter Form dokumentiert werden. Es sollten Verwaltungsinformationen, wie z.B. Name und Version des Programms, angegeben werden. Außerdem sollten Entwicklungsentscheidungen und Algorithmen ausreichend beschrieben werden.

Nachdem eine Klasse teilweise oder vollständig programmiert ist, wird sie kompiliert. Im Allgemeinen wird der Quellcode Fehler enthalten, die durch eine Fehleranalyse, unterstützt durch Debugger, behoben werden müssen. Viele dieser Fehler sind auf die menschliche Wahrnehmung und Denkweise zurückzuführen. Die nächste Abbildung klassifiziert typische Programmierfehler:

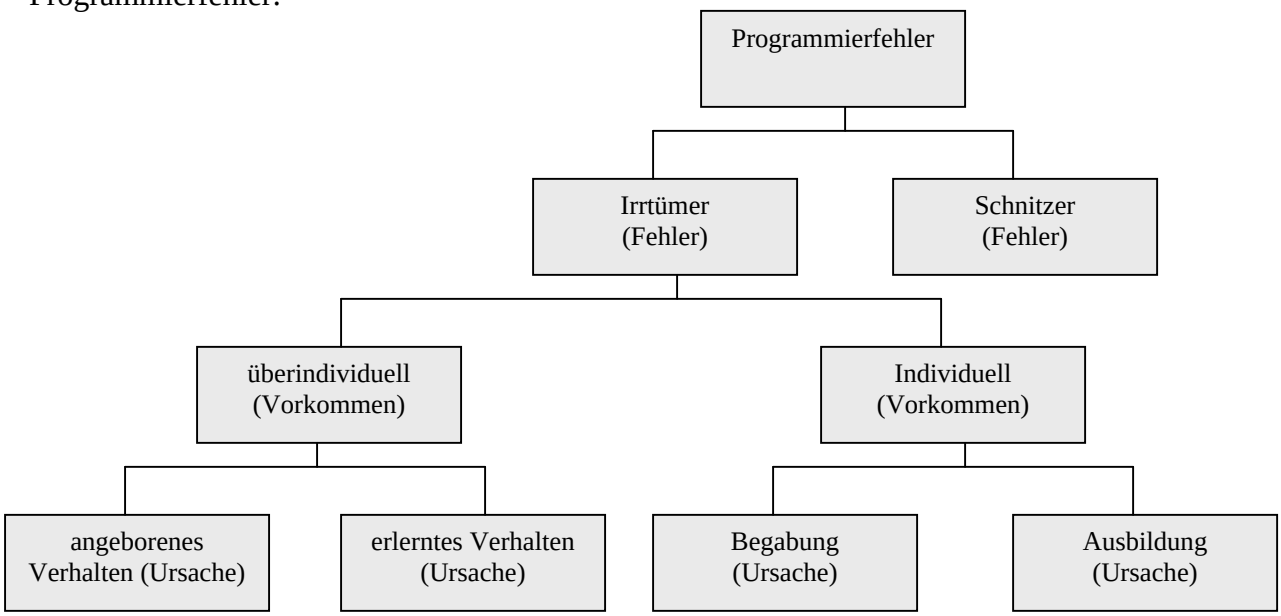


Abb.48: Klassifizierung von Programmierfehlern /Gramms 1990/

Schnitzer sind z.B. Tippfehler, die in der Regel von dem Kompiler entdeckt werden. Fehler, die auf *Irrtümer* beruhen, werden von dem Programmierer auch beim wiederholten Durchlesen seines Programms nicht gesehen. *Überindividuelle* Irrtümer ergeben sich, wenn das Hintergrundwissen der Aufgabenstellung nicht ausreichend verstanden wurde. *Individuelle* Fehler entstehen, weil die *Begabung* oder *Ausbildung* des Programmierers nicht angemessen ist. Denkfallen entstehen aufgrund von bestimmten Denkstrukturen und Prinzipien. Diese führen zu einem Verhaltens- und Denkmodell, welches das Verhalten des Programmierers beeinflußt. Solche Verhaltensweisen sind im Allgemeinen *angeboren* oder *erlernt*.

Nachdem die Programmierung abgeschlossen ist, sollten ein Testplan und entsprechende Testfälle entwickelt werden.

Ergebnisse

Als Ergebnis dieses Schritts sollte der Quellcode der spezifizierten Klasse vorliegen.

2.3.3.2 Klasse testen

Zweck

Das Ziel ist es, alle implementierten Klassen einzeln zu testen. Das Testen von Klassen entspricht im Kern dem Testen von Modulen bei konventioneller Entwicklungstechnik. Das bedeutet, daß klassische Testverfahren, wie Strukturtests (White-Box-Test) oder Funktionstests (Black-Box-Test), auf Klassen angewendet werden können. Einen Überblick über testende Verfahren haben wir bereits weiter oben gegeben (Abb. 37). Zusätzlich zu den testenden Verfahren können die weitaus anspruchsvolleren und aufwendigeren Verfahren der Verifikation angewendet werden.

Der Klassentest bildet die Grundlage für die schrittweise Integration von Klassen zu einer Komponente. Der Entwickler eines Testfalls spezifiziert das erwartete Verhalten und den Zustand eines Objekts. Während und nach der Durchführung des Tests sind diese mit dem tatsächlich beobachtbaren Verhalten und dem eingetretenen Zustand zu vergleichen. Abweichungen weisen auf Fehler hin, die analysiert und behoben werden müssen.

Das objektorientierte Paradigma erhöht durch eine Reihe von Faktoren die Komplexität von Klassentests:

- *Zustandsabhängigkeit des Verhaltens:* Für den Nachweis der korrekten Implementierung einer Klasse muß das Verhalten ihrer Objekte in verschiedenen Zuständen überprüft werden. Der Zustand eines Objekts ist durch seine Attributwerte definiert, die wiederum auf andere Objekte verweisen können. Damit entstehen eine sehr große Anzahl von Zuständen, die nicht mehr alle getestet werden können.
- *Kapselung und Information Hiding:* Das Testen von Klassen erfordert das Nachvollziehen der Zustände und den Zustandswechseln von Objekten. Durch Kapselung und Information Hiding wird das Beobachten von Zuständen aufwendiger.
- *Vererbung:* Abgeleitete Klassen definieren einen eigenen Zusammenhang, in dem eine geerbte Methode erneut getestet werden muß. Außerdem können geerbte Methoden redefiniert werden, was ebenfalls die Komplexität erhöht.
- *Polymorphismus:* Durch Polymorphie wird die Anzahl der Fallunterscheidungen im Code reduziert. Damit werden zwar die Kontrollstrukturen vereinfacht, eine Überdeckungsanalyse jedoch erschwert. Hinzu kommt, daß ein Test der möglichen dynamischen Bindung erforderlich wird.

Beteiligte

Ausführend: Softwareentwickler
Mitwirkend: Softwaredesigner

Vorgehen

Der Klassentest muß geplant, vorbereitet und durchgeführt werden. Hierzu sind folgende Schritte zu durchlaufen:

- Klassentest planen,
- Testumgebung für den Klassentest erstellen,
- Testfälle für den Klassentest definieren,
- Klassentest durchführen,
- Fehler analysieren und beheben.

Klassentest planen

Es sind die Art und der Umfang des Klassentests festzulegen. Die in der Abbildung 49 dargestellten Testkategorien können als Orientierungsgrundlage verwendet werden.

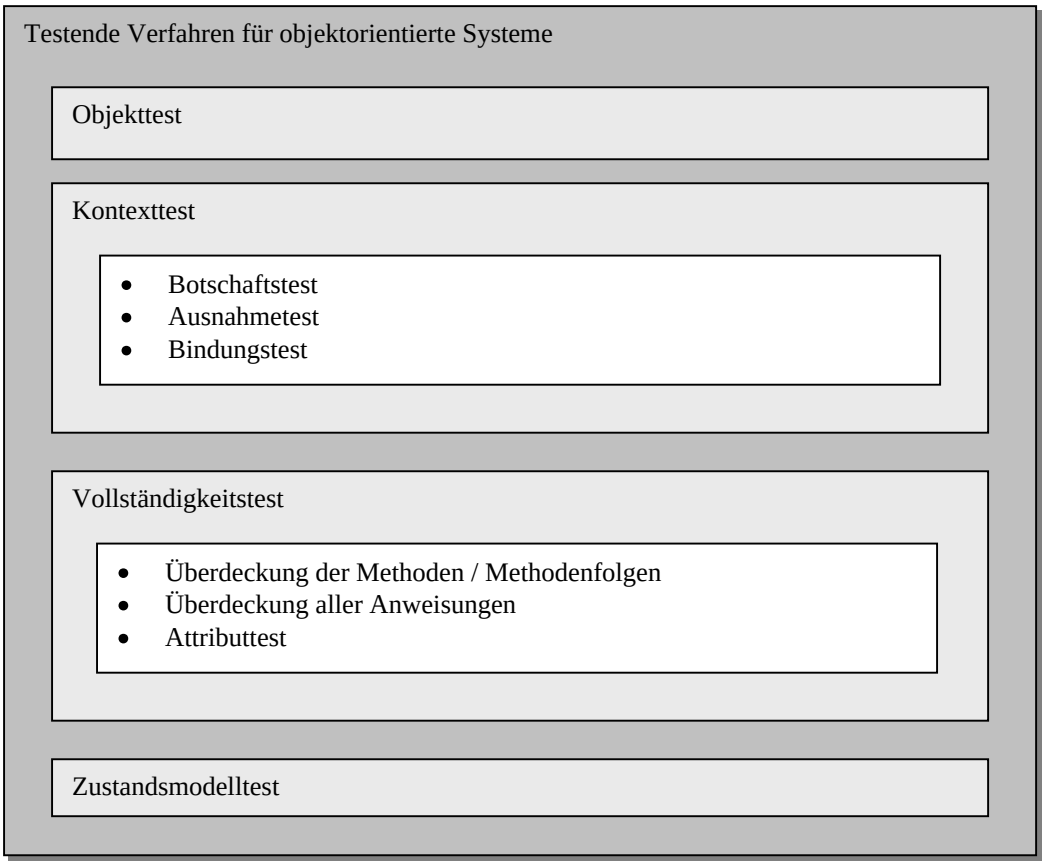


Abb. 49: Kategorien für den Test objektorientierte Systeme

Beim *Objekttest* einer Klasse werden für eine Klasse charakteristische Objekte ausgewählt, die erzeugt werden müssen. Im *Kontexttest* werden die Objekte einer Klasse in alle praktisch relevanten Situationen, wie beim Empfangen von Botschaften (*Botschaftstest*), beim Auslösen von Ausnahmebedingungen (*Ausnahmetest)* und dem dynamischen Binden (*Bindungstest*), getestet. Die Überdeckung aller *Methoden*, *Methodenfolgen* sowie aller *Anweisungen* pro Methode werden beim *Vollständigkeitstest* überprüft. Außerdem sind die *Attribute* hinsichtlich ihrer Zustandsänderung zu testen. Beim *Zustandsmodelltest* wird überprüft, ob sich die Objekte einer Klasse so verhalten, wie es in dem entsprechenden Zustandsdiagramm

beschrieben ist. Hierzu müssen alle Zustände erzeugt werden und die jeweiligen Übergänge angestoßen werden.

Sind einer oder mehrere der genannten Tests ausgewählt, so sind die benötigte Umgebung und Ressourcen zu bestimmen. Danach ist der Ablauf der Klassentests, die Festlegung der Erfolgskriterien und die zeitliche Durchführung festzulegen.

Testumgebung für den Klassentest erstellen

Die für den Klassentest benötigte Testumgebung sollte die in der nächsten Abbildung dargestellten Elemente enthalten:

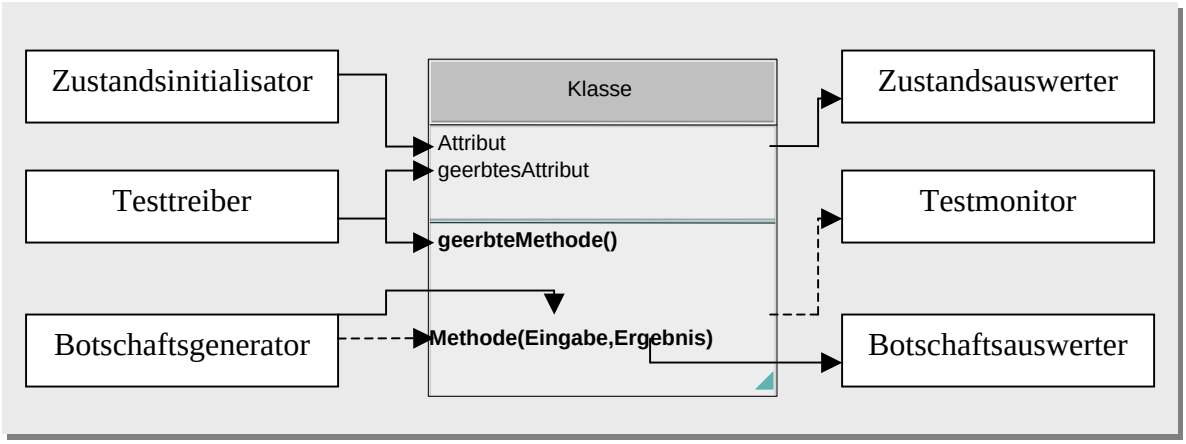


Abb. 50: Elemente der Testumgebung für den Test einer Klasse /Binder 1999/

Der *Testtreiber* sorgt dafür, daß die geerbten Attribute und Methoden simuliert werden und stößt die zu testende Methode an. Da Methoden in der Regel Eingabeparameter benötigen, erzeugt der *Botschaftsgenerator* die entsprechenden Parameter. Der *Zustandsinitialisator* sorgt dafür, daß sich das zu testende Objekt im gewünschten Ausgangszustand befindet. Der *Testmonitor* protokolliert den Ablauf des Tests, indem er die Folge der Methodenaufrufe festhält. Der erreichte Zustand nach dem Test wird von dem *Zustandsauswerter* protokolliert. Ein Vergleich der spezifizierten Ergebnisse mit den tatsächlich erreichten Ergebnissen wird vom *Botschaftsauswerter* vorgenommen.

Testfälle für den Klassentest definieren

Ein erfolgreicher Test setzt die vollständige und einheitliche Spezifikation der Testfälle voraus. Hierzu sind entsprechende einheitliche Beschreibungsschemata nötig, denn nur so ist es möglich:

- Reviews der Testfälle durchzuführen,
- Tests zu automatisieren und zu versionieren,
- Tests vergleichbar und wiederholbar zu machen.

Die Vergleichbarkeit und Wiederholbarkeit von Tests spielen im Rahmen von Regressionstests eine wichtige Rolle, um z.B. nachzuweisen, daß das Verhalten einer Klasse nach Modifikation anderer Klassen unverändert bleibt. Regressionstests sind bei einem evolutionären Entwicklungsprozeß besonders wichtig. Denn bei jedem neuen Zyklus wird der größte Teil der Tests der vergangenen Zyklen wiederholt werden. Dieser Zusammenhang wird durch die nächste Abbildung veranschaulicht.

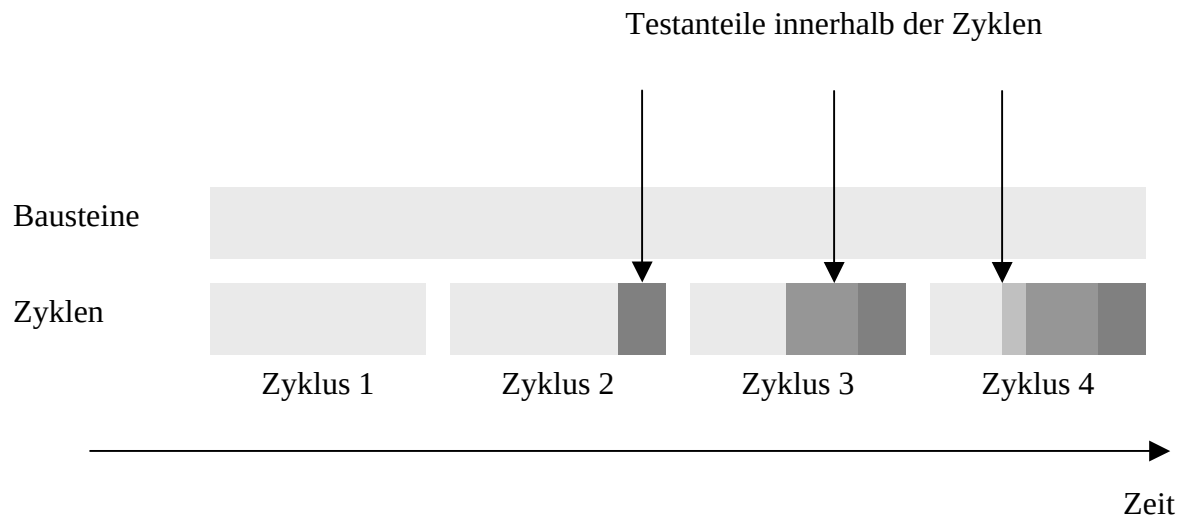


Abb. 51: Regressionstest bei der evolutionären Software-Entwicklung

Zur Spezifikation von Testfällen kann das nachfolgende Beschreibungsschema, das auf /Berard 1993/ zurückgeht, verwendet werden. Es kann gleichzeitig auch als Testanweisung und Testprotokoll verwendet werden. Das Beschreibungsschema besteht aus einem Kopfteil mit einer Beschreibung des Testfalls, einem Hauptteil mit der Spezifikation von Teilschritten und einem Fußteil mit Bewertung der Testergebnisse.

Testnummer	⇨ Da ein Testfall in der Regel mehrfach getestet wird, empfiehlt sich die Vergabe eines identifizierenden Merkmals pro Test. [**Hier** klicken und Text eingeben]
Testdatum	⇨ Geben sie hier an, wann der Test stattgefunden hat. [**Hier** klicken und Text eingeben]
Testteam	⇨ Führen Sie hier die Namen der am Test beteiligten Personen auf. [**Hier** klicken und Text eingeben]
Klasse	⇨ Führen Sie hier die zu testende Klasse an. Sorgen Sie für die eindeutige Identifizierbarkeit der Klassen, indem Sie gegebenenfalls die Komponente benennen, der die Klasse zugeordnet ist. [**Hier** klicken und Text eingeben]
ID	⇨ Vergeben Sie hier ein eindeutiges Merkmal zur Identifikation des Testfalls, denn pro Klasse gibt es i.a. mehrere Testfälle. [**Hier** klicken und Text eingeben]
Testkategorie	☐ Objekttest ☐ Kontexttest ☐ Vollständigkeitstest ☐ Zustandsmodelltest
Zweck	⇨ Begründen Sie hier, warum Sie diesen Testfall gebildet haben. Was ist sein Zweck, was soll damit nachgewiesen werden? [**Hier** klicken und Text eingeben]
Hinweis	⇨ Sammeln Sie hier alle Hinweise und Bemerkungen, die notwendig sind, um den Test anhand des beschriebenen Testfalls zu verstehen. [**Hier** klicken und Text eingeben]

Abb. 52: Beschreibungsschema für Testfälle, Testanweisungen und Testprotokolle: Kopfteil für den Klassentest

Lfd. Nummer	**Externe Bedingungen und Einflüsse**			
	⇨ Objekte können durch äußere Bedingungen (externe Ereignisse, Interrupts etc.) beeinflußt werden. Spezifizieren Sie hier diese Bedingungen in Form einer Liste.			
	[**Hier** klicken und Text eingeben]			
Lfd. Nummer	**Testanweisungen**			
	⇨ Beschreiben Sie hier, was der Tester im Rahmen dieses Testschritts zu tun hat.			
	[**Hier** klicken und Text eingeben]			
Lfd. Nummer	**Erwarteter Zustand**		**Tatsächlicher Zustand**	
	⇨ Führen Sie in dieser Liste alle Zustände auf, die im Rahmen dieses Testschrittes auftreten.		⇨ Dokumentieren Sie hier nach Durchführung des Testschrittes den tatsächlich erreichten Zustand.	
	[**Hier** klicken]		[**Hier** klicken]	
Lfd. Nummer	**Botschaften/ Methoden**	**Eingabeparameter**	**Erwartete Ergebnisparameter**	**Tatsächlicher Wert Ergebnisparameter**
	⇨Stellen Sie in dieser Liste alle Botschaften/ Methoden zusammen, die an der Durchführung des Testschrittes beteiligt sind.	⇨ Führen Sie hier pro Botschaft/Methode die erwarteten Ergebnis - parameter (Name, Wert, Position, Typ) auf.	⇨Führen Sie hier pro Botschaft / Methode die erwarteten Ergebnis- parameter (Name, Wert, Position, Typ) auf.	⇨ Dokumentieren Sie hier pro Botschaft / Methode die tatsächlichen Werte der Ergebnis- parameter.
	[**Hier** klicken]	**Hier** klicken]	[**Hier** klicken]	[**Hier** klicken]
Lfd. Nummer	**Erwartete Ausnahme**		**Tatsächliche Ausnahme**	
	⇨ Erstellen Sie hier eine Liste aller Ausnahmen, die im Rahmen des Testschrittes erwartet werden können.		⇨ Stellen Sie den erwarteten die tatsächlich aufgetretenen Ausnahmen gegenüber.	
	[**Hier** klicken]		[**Hier** klicken]	
Lfd. Nummer	**Erwartete Interrupts**		**Tatsächliche Interrupts**	
	⇨ Hardware-nahe Objekte erzeugen gegebenenfalls Interrupts. Der Zweck eines Testfalls kann also darin bestehen, ein Objekt dazu zu bewegen, einen Interrupt auszulösen. Dokumentieren Sie hier den erwarteten Interrupt.		⇨ Stellen Sie hier dem erwarteten den tatsächlichen Interrupt gegenüber.	
	[**Hier** klicken]		[**Hier** klicken]	

Abb. 53: Spezifikation jedes Teilschritts

Zusammen-fassung	⇨ Geben Sie hier, sofern erforderlich, eine Zusammenfassung von Testverlauf und Ergebnissen. [**Hier** klicken und Text eingeben]	
Bewertung/ Empfehlung	☐ fehlerfreier Ablauf ☐ leichte Fehler ☐ schwere Fehler ☐ fatale Fehler	☐ Ergebnis akzeptieren, Test abschließen ☐ Korrektur ohne Testwiederholung ☐ Korrektur und Testwiederholung
Hinweise zur Fehler-behebung	⇨ Sammeln Sie hier alle Hinweise und Bemerkungen, die bei der Ermittlung der Fehlersuche und bei der Fehlerbehebung von Nutzen sein können. [**Hier** klicken und Text eingeben]	

Abb. 54: Beschreibungsschema für Testfälle, Testanweisungen und Testprotokolle: Fußteil für den Klassentest

Klassentest durchführen

Wie beim Einzeltest von Klassen vorgegangen wird, hängt sehr stark davon ab, ob eine *gewöhnliche*, *abgeleitete*, *abstrakte* oder *parametrisierte* Klasse (Template-Klasse) zu überprüfen ist.

- *Gewöhnliche Klasse testen:* Hierbei wird zunächst ein Objekt der zu testenden Klasse in der Testumgebung erstellt. Dann sind alle Methoden einzeln zu testen, die keine anderen Methoden benutzen. Anschließend sind die Methoden getestet, die von anderen Methoden derselben Klasse abhängen. Zuletzt sind alle Methoden zu testen, die Dienste fremder Objekte nutzen.
- *Abgeleitete Klasse testen:* Erweitert die abgeleitete Klasse die Basisklasse lediglich um neue Methoden, so sind alle Testfälle der Basisklasse auch auf die abgeleiteten Klassen anzuwenden, um die geerbten Methoden im veränderten Kontext zu testen. Für neue und redefinierte Methoden sind eigene Testfälle zu entwickeln und auszuführen.
- *Abstrakte Klasse testen:* Da abstrakte Klassen nicht selbst instantiiert werden können, sind Objekte der abgeleiteten Klassen zu erstellen und zu testen.
- *Parametrisierte Klassen testen:* Bevor eine parametrisierte Klasse getestet werden kann, muß sie „konkretisiert“ werden. Dabei sollte zunächst eine möglichst einfache Klasse mit unkomplizierten Parametern erzeugt und getestet werden. Danach sollte zu komplizierteren, aber praktisch relevanten Parametern übergegangen werden.

Der Testverlauf und die Ergebnisse sollten protokolliert werden.

Fehler analysieren und beheben

Die aufgetretenen Fehler sollten analysiert und behoben werden. Zu beachten ist, daß neben der getesteten Klasse auch die Testumgebung, die Spezifikation der Testfälle und die Testdurchführung als mögliche Fehlerquellen in Frage kommen. Nachdem die Fehler korrigiert worden sind, sollte die entsprechende Klasse erneut getestet werden.

Ergebnisse

Zur Spezifikation der Testfälle, Testanweisungen und Testergebnisse sollte das vorgestellte Beschreibungsschema verwendet werden. Für detaillierte und problemspezifische Testdokumentation sind die folgenden ANSI/IEEE-Normen zu empfehlen:

ANSI/IEEE STD 829-1983:	Software Test Documentation
ANSI/IEEE STD 1008-1987:	Standard for Software Unit Testing
ANSI/IEEE STD 1012-1986:	Standard Verification and Validation Plans

Hinweise und Tipps

Als vertiefende Darstellung der Aspekte des Testens von objektorientierten Systemen ist /Binder 1999/ zu empfehlen. Weitere hilfreiche Quellen sind /Balzert 1998/ und /Goldberg, Rubin 1995/.

2.3.4 Operationeller Einsatz

Beim operationellen Einsatz muß die Klassendokumentation weiter gepflegt werden und die Klasse auf Wiederverwendbarkeit untersucht werden. Die nächste Abbildung faßt diese Aktivitätstypen zusammen.

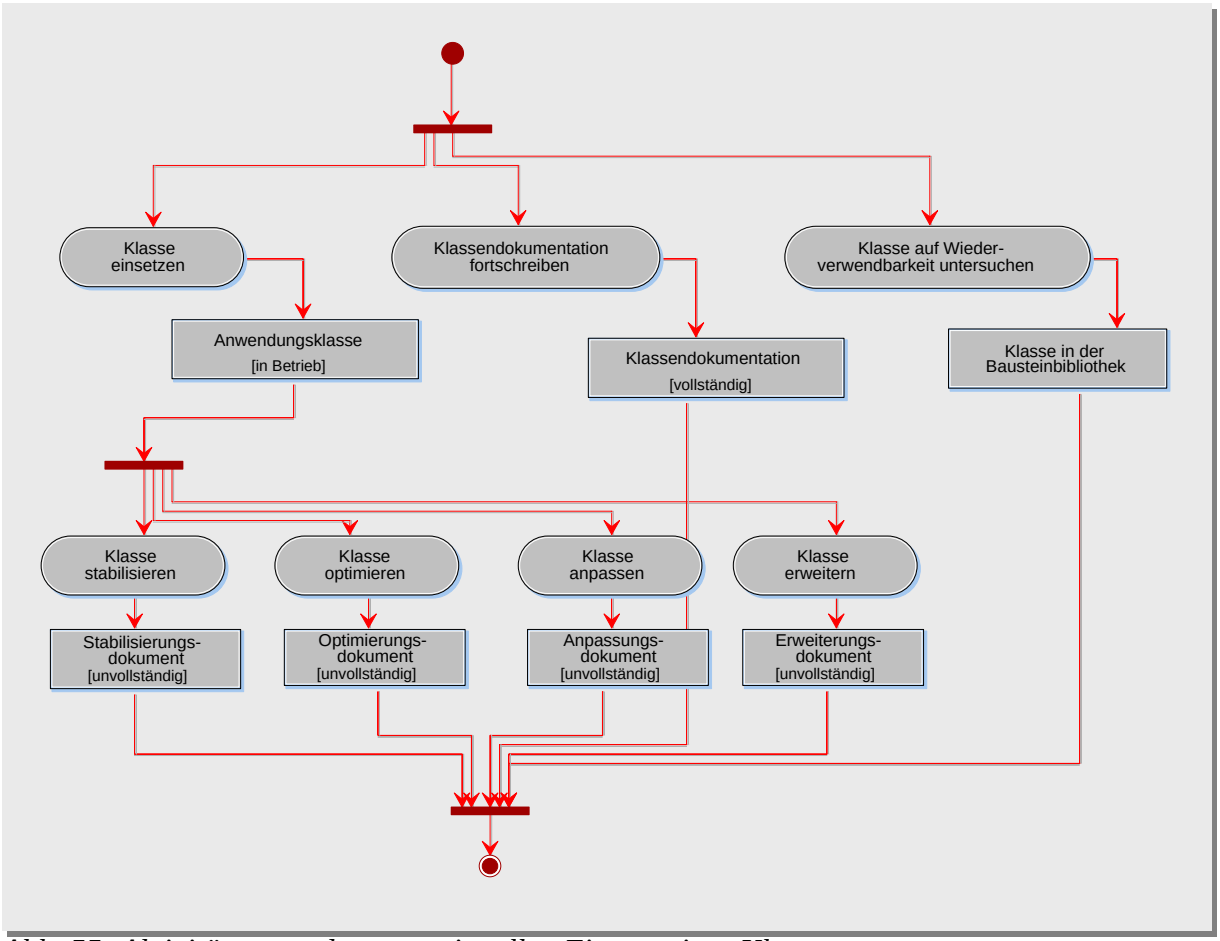


Abb. 55: Aktivitätstypen des operationellen Einsatz einer Klasse

Die im Rahmen des operationellen Einsatzes notwendigen Optimierungen, Anpassungen, Erweiterungen und Fehlerkorrekturen einer Komponente betreffen im Detail ihre zugeordneten Klassen. Daher müssen während des operationellen Einsatzes einer Klasse folgende Schritte unternommen werden, die unter Umständen weitere Klassenzyklen veranlassen können:

- Klasse stabilisieren,
- Klasse optimieren,
- Klasse anpassen,
- Klasse erweitern.

Die genannten Aktivitätstypen verlaufen analog zu den entsprechenden Aktivitätstypen auf der Komponenten-Ebene und werden daher nicht weiter ausgeführt.

In diesem Kapitel haben wir eine Methode zur Software-Entwicklung für das EOS-Modell vorgestellt, die wir mit den nächsten Abbildungen zusammenfassen. Die ersten beiden Tabellen beschreiben die entsprechenden Aktivitätstypen, Beteiligte und Ergebnisse für die drei Entwicklungsebenen vom EOS-Modell. In der Grafik in Abbildung 58 wird der Einsatz von UML für den Software-Entwicklungsprozeß im EOS-Modell dargestellt.

Baustein	Aktivitätstypen	Beteiligte	Ergebnisse
System			
Analyse	funktionale Anforderungen analysieren	Systemanalytiker, Anwender	Beschreibung der fkt. Anfroderungen
	nicht-funktionale Anforderungen definieren	Systemanalytiker, Anwender	Beschreibung der nicht-fkt. Anforderungen
	Anforderungen prüfen	Projektmanager, Anwender, Projektleiter, Systemanalytiker	Anforderungsbewertung
	Vorgaben definieren	Projektleiter, Systemanalytiker, Softwareentwickler	Projekt-Richtlinien
	Systemdokumentation anlegen	Systemanalytiker, Projektleiter	Systemdokumentation
Entwurf	Anwendungsbereich in Komponenten strukturieren	Systemanalytiker, Anwendungsbereich-experte	Komponentenstruktur
	Zielumgebung spezifizieren	Projektleiter, Systemarchitekt	Spezifikation der Zielumgebung
	Integrationsstrategie festlegen	Softwaredesigner, Systemarchitekt, Softwareentwickler, Qualitätsprüfer	Integrationsstrategie
	Programmiervorgaben erstellen	Projektleiter, Softwareentwickler	Programmier-Richtlinien
	Systemarchitektur entwerfen	Systemarchitekt, Systemdesigner, Projektleiter	Systemarchitektur
	Systemdokumentation fortschreiben	Systemanalytiker, Projektleiter	Systemdokumentation

Implementierung	Subsysteme definieren	Softwaredesigner, Systemarchitekt, Softwareentwickler, Qualitätsprüfer	Spezifikation der Subsysteme
	System integrieren	Softwareentwickler, Qualitätsprüfer, Softwaredesigner, Systemarchitekt	Integrationsdokument
	System testen	Qualitätsprüfer, Softwareentwickler	Testdokument
	Anwendungsdokumentation erstellen	Technischer Autor, Anwendungsbereich-experte, Systemanalytiker, Softwaredesigner, Softwareentwickler	Handbücher
	Systemdokumentation fortschreiben	Systemanalytiker, Projektleiter	Systemdokumentation
Operationeller Einsatz	Akzeptanz testen	Auftraggeber, Anwender, Qualitätsprüfer	Akzeptanzbericht
	System einsetzen	Softwareentwickler, Projektleiter, Anwender	Anwendungssystem
	Inbetriebnahme unterstützen	Anwendungsbereich-experte, Softwarearchitekt, Softwareentwickler, Auftraggeber, Projektleiter	Inbetriebnahmedokument
	Systemdokumentation fortschreiben	Systemanalytiker, Projektleiter	Systemdokumentation
	System stabilisieren	Softwaredesigner, Softwareentwickler, Projektleiter, Anwender	Stabilisierungsdokument
	System optimieren	Softwaredesigner, Softwareentwickler, Projektleiter, Anwender	Optimierungsdokument
	System anpassen	Softwaredesigner, Softwareentwickler, Projektleiter, Anwender	Anpassungsdokument
	System erweitern	Softwaredesigner, Softwareentwickler, Projektleiter, Anwender	Erweiterungsdokument

Abb. 56: Überblick der Aktivitätstypen, der Beteiligten und der Ergebnisse

Baustein	Aktivitätstypen	Beteiligte	Ergebnisse
Komponente			
Analyse	Anwendungsfall konkretisieren	Systemanalytiker, Anwender	Anwendungsfallmodell
	Bausteinbibliothek durchsuchen	Systemanalytiker, Softwaredesigner, Projektbibliothekar	wiederverwendbarer Baustein
	Komponentendokumentation anlegen	Systemanalytiker, Softwaredesigner	Komponenten-dokumentation
Entwurf	Klassenmodell entwickeln	Systemanalytiker, Anwender	Klassendiagramme
	Botschaftsfluß modellieren	Systemanalytiker, Softwaredesigner	Interaktionsdiagramme
	Komponentendokumentation fortschreiben	Systemanalytiker, Softwaredesigner	Komponenten-dokumentation
Implementierung	Subsysteme definieren	Softwaredesigner, Systemarchitekt, Softwareentwickler, Qualitätsprüfer	Spezifikation der Subsysteme
	Komponente integrieren	Softwareentwickler, Qualitätsprüfer, Softwaredesigner	Integrationsdokument
	Komponente testen	Softwareentwickler, Qualitätsprüfer, Softwaredesigner	Testdokument
	Komponentendokumentation fortschreiben	Systemanalytiker, Softwaredesigner, Softwareentwickler	Komponenten-dokumentation
Operationeller Einsatz	Komponente einsetzen	Softwareentwickler, Anwender	Anwendungskomponente
	Komponente auf Wieder-verwendung untersuchen	Softwaredesigner, Softwareentwickler, Projektbibliothekar	Komponente in der Bausteinbibliothek
	Komponentendokumentation fortschreiben	Systemanalytiker, Softwaredesigner, Softwareentwickler	Komponenten-dokumentation
	Komponente stabilisieren	Softwaredesigner, Softwareentwickler, Projektleiter, Anwender	Stabilisierungsdokument
	Komponente optimieren	Softwaredesigner, Softwareentwickler, Projektleiter, Anwender	Optimierungsdokument
	Komponente anpassen	Softwaredesigner, Softwareentwickler, Projektleiter, Anwender	Anpassungsdokument
	Komponente erweitern	Softwaredesigner, Softwareentwickler, Projektleiter, Anwender	Erweiterungsdokument
Klasse			
Analyse	Klassenfunktionalität beschreiben	Softwaredesigner, Systemanalytiker	Klassenbeschreibung
	Bausteinbibliothek durchsuchen	Systemanalytiker, Softwaredesigner, Projektbibliothekar	wiederverwendbarer Baustein
	Klassendokumentation anlegen	Systemanalytiker, Softwaredesigner	Klassendokumentation
Entwurf	Attribute definieren	Softwaredesigner, Systemanalytiker	Attributbeschreibung

	Methoden definieren	Softwaredesigner, Systemanalytiker	Methodenbeschreibung
	Zustandsmodell entwickeln	Systemanalytiker, Softwaredesigner	Zustandsdiagramm
	Klassendokumentation fortschreiben	Systemanalytiker, Softwaredesigner	Klassendokumentation
Implementierung	Klasse kodieren	Softwareentwickler, Softwaredesigner	Klassen-Code
	Klasse testen	Softwareentwickler, Softwaredesigner	Testfälle, Testanweisungen und Testprotokoll
	Klassendokumentation fortschreiben	Systemanalytiker, Softwaredesigner, Softwareentwickler	Klassendokumentation
Operationeller Einsatz	Klasse einsetzen	Softwareentwickler, Anwender	Anwendungsklasse
	Klasse auf Wiederverwendung untersuchen	Softwaredesigner, Softwareentwickler, Projektbibliothekar	Klasse in der Bausteinbibliothek
	Klassendokumentation fortschreiben	Systemanalytiker, Softwaredesigner, Softwareentwickler	Klassendokumentation
	Klasse stabilisieren	Softwaredesigner, Softwareentwickler, Anwender	Stabilisierungsdokument
	Klasse optimieren	Softwaredesigner, Softwareentwickler, Anwender	Optimierungsdokument
	Klasse anpassen	Softwaredesigner, Softwareentwickler, Anwender	Anpassungsdokument
	Klasse erweitern	Softwaredesigner, Softwareentwickler, Anwender	Erweiterungsdokument

Abb. 57: Überblick der Aktivitätstypen, der Beteiligten und der Ergebnisse (Fortsetzung)

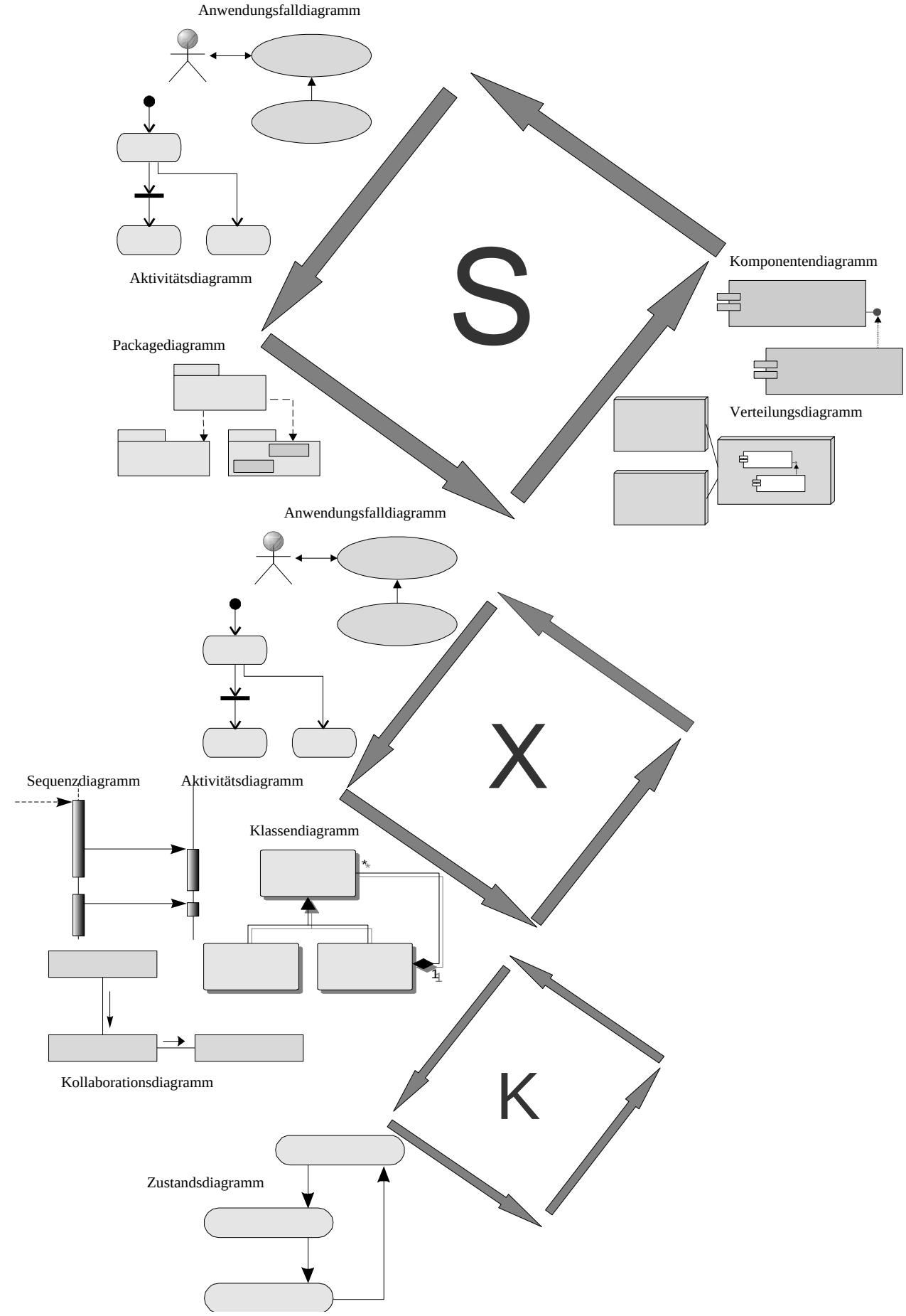


Abb. 58: Einsatz von UML im EOS-Modell

3 Qualitätssicherung

„Fehler, die nicht gemacht werden können, brauchen auch nicht behoben zu werden“. Daher sollte stets durch vorausblickende konstruktive Maßnahmen der Umfang von analytischen Qualitätssicherungs-Maßnahmen reduziert werden. In konventionellen Vorgehensmodellen werden analytische Qualitätssicherungs-Maßnahmen im allgemeinen erst am Ende des Entwicklungsprozesses eingesetzt. Das Beheben von Fehlern ist aber teuer, vor allem wenn sie im späteren Verlauf des Projekts entdeckt werden. Um die frühzeitige Fehlererkennung zu ermöglichen, ist im EOS-Modell die Qualitätssicherung in dem Entwicklungsprozeß integriert und begleitet die Software-Entwicklung. In dem Prozeßmodell EOS ist der primäre Gegenstand der Qualitätssicherung die entwickelten Bausteine. Sobald Analyse, Entwurf, Implementierung oder operationeller Einsatz an einem Baustein abgeschlossen sind, sollte eine Prüfung der produzierten Ergebnisse vorgenommen werden (*statische Verfahren*). Der Qualitätsstand des Systems, der Komponenten und Klassen wird so sichtbar und eine realistische Beurteilung des Entwicklungsfortschritts möglich.

Da Softwareentwickler in der Regel einen hohen Qualitätsanspruch haben, prüfen sie alle Produkte, die während der Entwicklungsphasen eines Bausteins entwickelt wurden, selbst kritisch. Für viele Produkte wird aber vom Unternehmen eine Qualitätssicherung vorgeschrieben, die nicht von den Softwareentwicklern selbst ausgeführt wird. Das gilt ebenfalls für die Prüfung von Aktivitäten, die zur Verbesserung der Prozeßqualität dient. Es gibt auch Maßnahmen zur Qualitätssicherung, die unverzichtbar sind, wie z.B. Integrationstest oder Klassentest. Daher haben wir solche unverzichtbaren Qualitätssicherungs-Maßnahmen von vornherein als Aktivitätstypen in den Subprozessen, wie z.B. in den Subprozeß „Software-Entwicklung“, aufgenommen (*dynamische Verfahren*). Die Qualitätssicherungs-Maßnahmen aller anderen Produkt- und Aktivitätstypen werden im Rahmen des Subprozesses „Projektmanagement“ geplant. Dort wurden die zu prüfenden Aktivitäts- und Produkttypen ausgewählt, die zugehörigen konstruktiven und analytischen Qualitätssicherungs-Maßnahmen festgelegt und zur Durchführung der Maßnahmen eine personelle Zuordnung getroffen.

Zur Qualitätssicherung eines Produkts oder einer Aktivität müssen folgende allgemeine Schritte durchlaufen werden:

- Prüfung planen, d.h. Prüfgegenstand und -verfahren festlegen,
- Prüfung vorbereiten, d.h. Prüffälle definieren und Prüfumgebung einrichten,
- Prüfung durchführen, d.h. das Produkt oder die Aktivität wird formal und inhaltlich geprüft.

Die genannten Schritte werden in den nächsten Abschnitten ausführlich beschrieben.

3.1 Prüfung planen

Zweck

Das Ziel dieses Aktivitätstyps ist es, den Prüfvorgang für ein Produkt oder eine Aktivität im Detail zu planen. Das Ergebnis wird in einem QS-Handbuch verbindlich festgehalten, das alle Ergebnisse des Teilschrittes „Prüfung planen“ beinhaltet. Das QS-Handbuch dokumentiert die Entscheidungen, in dem es folgende Fragen klärt:

- Was ist der Prüfgegenstand?
- Welche Qualitätsanforderungen müssen berücksichtigt werden?
- Wie ist der Verlauf des Prüfverfahrens?
- Welche Prüfkriterien sind bei dem gewählten Prüfverfahren zu beachten?
- Wie sind Termine und Ressourcen zu planen?

Wann und unter welchem Einsatz von Ressourcen die in der Regel statischen Prüfungen durchgeführt werden, definiert der QS-Plan (siehe Kapitel „Projektmanagement“, Abschnitt „Qualitätssicherung planen“).

Beteiligte

Ausführend: Qualitätsprüfer, Projektleiter
Beratend: Projektmanager

Vorgehen

Im Detail sind folgende Schritte durchzuführen:

- Prüfgegenstand festlegen,
- Prüfverfahren bestimmen,
- Prüfkriterien definieren,
- Termine und Ressourcen planen.

Prüfgegenstand festlegen

Im QS-Plan sind die im Rahmen eines Bausteines entwickelten Produkttypen und auszuführenden Aktivitätstypen definiert, die geprüft werden müssen. Es ist nun zu entscheiden, ob wirklich alle Aktivitäten bzw. Produkte des festgelegten Typs geprüft werden. In der Regel ist es sinnvoll, Ausnahmen zu machen, z.B. werden nur die Produkte von kritischen Bausteinen geprüft.

Außerdem ist zu entscheiden, was genau zum Prüfgegenstand gehört. Wenn z.B. Komponenten im Entwurf überprüft werden, gehört dann die Prüfung von Sequenzdiagrammen dazu, oder sollen nur Klassendiagramme geprüft werden?

Prüfverfahren bestimmen

In Abhängigkeit von der Zielsetzung lassen sich drei Typen von Qualitätssicherungs-Maßnahmen unterscheiden /Liggesmeyer 1990/:

- Testende Verfahren,
  - Dynamische Testverfahren,
  - Statische Testverfahren,
- Verifizierende Verfahren,
  - Verifikation,
  - Symbolische Ausführung,
- Analytische Verfahren,
  - Metriken,
  - Anomalienanalyse.

Testende Verfahren dienen zur Erkennung von Fehlern. *Verifizierende Verfahren* beweisen die Korrektheit eines Bausteins. *Analysierende Verfahren* vermessen Eigenschaften eines Bausteins.

Unter die *dynamischen Tests* fallen eine Reihe von Testverfahren, die das ausführbare Programm mit konkreten Eingabewerten versehen und in einer realen Umgebung ausführen lassen. Der *statische Test* umfaßt Testverfahren, bei denen Produkte, wie z.B. Dokumente oder Quellcode, analysiert werden, um Fehler zu finden. Häufig benutzte Verfahren dieser Kategorie sind Inspektionen, Reviews und Walkthroughs. Die *Verifikation* beweist mit mathematischen Mitteln die Korrektheit der Implementierung anhand der Spezifikation. Bei der *symbolischen Ausführung* werden symbolische Eingabewerte erzeugt und über einen Interpreter durch den Quellcode verarbeitet. Mit Hilfe von *Metriken* lassen sich Eigenschaften von Bausteinen, wie z.B. Komplexität, Umfang oder interne Bindung, vermessen. Die ermittelten Werte helfen, Projekte zu vergleichen und gegebenenfalls Eigenschaften vorherzusagen. Durch *Tabellen und Grafiken* lassen sich Programme unter speziellen Aspekten analysieren und die Ergebnisse in gut lesbarer und interpretierbarer Form darstellen, wie z.B. Attribut-Verwendungslisten. Anomalien, wie z.B. Updateanomalien, lassen sich gezielt durch *Anomalieanalysen* aufdecken und korrigieren.

Die Abbildung 1 klassifiziert analytische Qualitätssicherungs-Maßnahmen und faßt bekannte Verfahren zusammen.

Dynamische Testverfahren, vor allem Verfahren für objektorientierte Software, wurden bereits im Kapitel „Software-Entwicklung“ ausführlich diskutiert. Im folgenden werden wir auf die in der Praxis häufig eingesetzten manuellen Prüfmethoden eingehen. Diese sind oft die einzige Möglichkeit, Semantik zu überprüfen. Das Beschreiben der verifizierenden und analysierende Verfahren würde den Rahmen springen, so daß wir auf /Apt, Olderog 1994/, /Francez 1992/, /Henderson-Sellers 1996/, /Chidamber, Kemerer 1994/, /Lorenz, Kidd 1994/ und /Zuse 1998/ verweisen müssen. Im folgenden gehen wir kurz auf die manuellen Prüfmethoden näher ein.

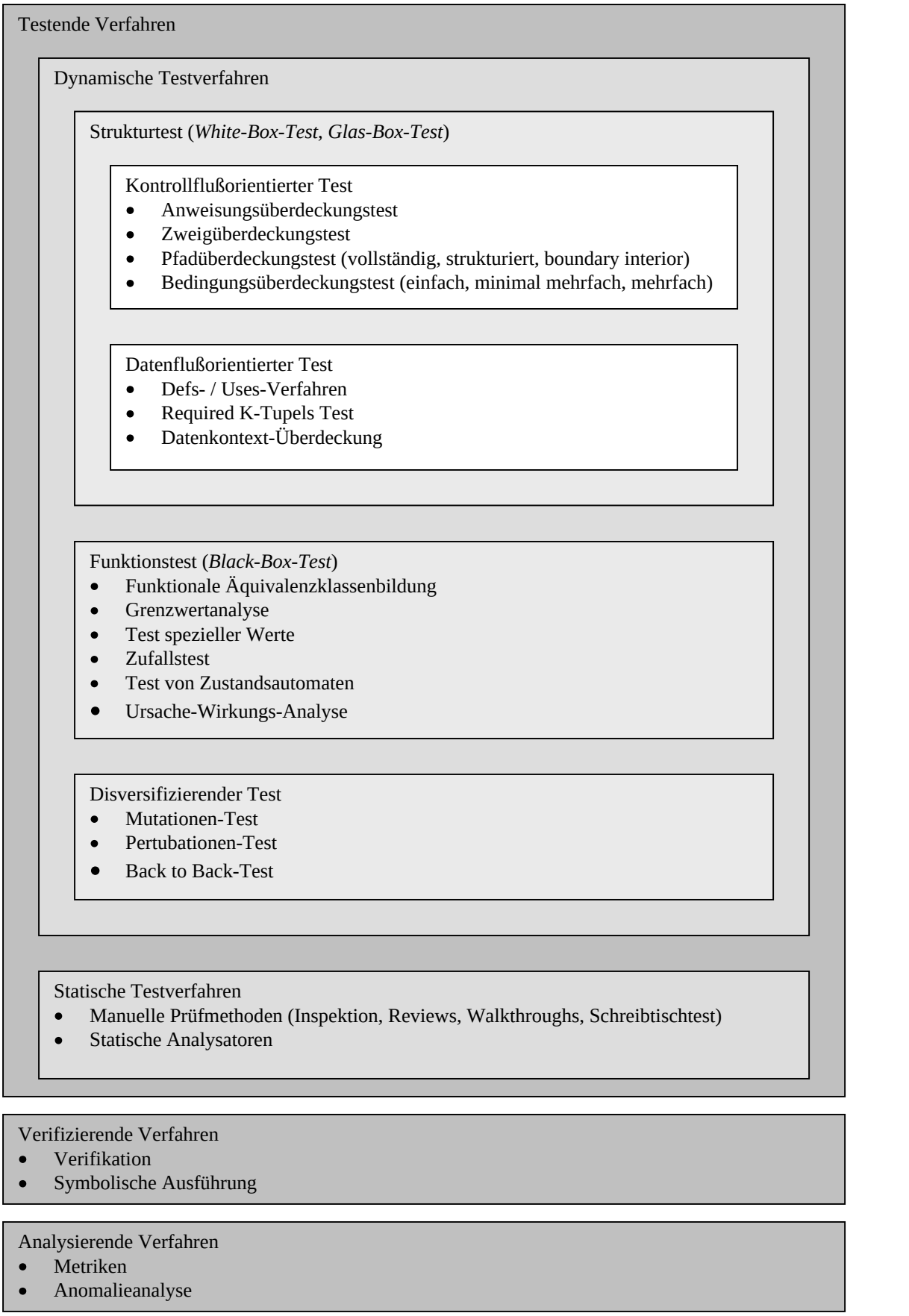


Abb. 1: Klassifikation analytischer Qualitätssicherungs-Maßnahmen /Balzert 1998/

Es lassen sich im wesentlich drei manuelle Prüfmethoden unterscheiden:

- Inspektionen,
- Reviews,
- Walkthroughs.

Die Abgrenzung zwischen den drei Prüfmethoden ist in der Literatur sehr verschwommen. Das gilt auch für die Begriffsbildung (einige Unternehmen sprechen z.B. von Umlaufreview und Sitzungsreview). Es gibt eine Reihe von Gemeinsamkeiten, aber auch gravierende Unterschiede zwischen den Methoden. Das Ziel der drei Prüfmethoden ist es, Fehler, Defekte und Inkonsistenzen durch manuelle Analysen aufzudecken. Die Überprüfung findet in einer Gruppensitzung durch ein möglichst kleines Team mit klar definierten Rollen statt. Die Ergebnisse und das Vorgehen beim Prüfen werden schriftlich festgehalten. Vorgesetzte nehmen an den Prüfungen nicht teil. Insbesondere dürfen die Prüfungsergebnisse nicht zur Beurteilung von Mitarbeitern herangezogen werden.

Die Hauptunterschiede zwischen den drei Prüfmethoden sind in der nächsten Tabelle zusammengefaßt.

	Inspektion	**Review**	**Walkthrough**
Ziele	• Schwere Defekte im Prüfobjekt identifizieren • Entwicklungsprozeß verbessern • Metriken ermitteln	• Stärken und Schwächen des Prüfobjekts identifizieren • Entwicklungsprozeß verbessern	• Defekte und Probleme des Prüfobjekts identifizieren • Ausbildung von Benutzern und Mitarbeitern
Teilnehmer	• Moderator • Autor • Gutacher • Protokollführer • (Vorleser)	• Moderator • Autor • Gutacher • Protokollführer	• Autor (= Moderator) • Gutacher
Durchführung	• Eingangsprüfung • Planung • (Einführungssitzung) • indiv. Vorbereitung und Prüfung • Gruppensitzung • Überarbeitung • Nachprüfung • Freigabe	• (Eingangsprüfung) • Planung • (Einführungssitzung) • indiv. Vorbereitung und Prüfung • Gruppensitzung • Überarbeitung • Nachprüfung	• (indiv. Vorbereitung und Prüfung) • Gruppensitzung
Referenzunterlagen	• Ursprungsprodukt • Erstellungsregeln • Checklisten • Inspektionsregeln • Inspektionsplan	• Ursprungsprodukt • Erstellungsregeln • Fragenkataloge	• Ursprungsprodukt • Fragenkataloge
Charakteristika	• Ausgebildeter Moderator • Prüfobjekt wird vom Vorleser Absatz für Absatz vorgetragen • Moderator gibt Freigabe	• Ausgebildeter Moderator • Prüfteam gibt Empfehlung an Manager	• Prüfobjekt wird vom Autor ablauforientiert vorgetragen • Autor entscheidet

Abb. 2: Hauptunterschiede zwischen manuellen Prüfmethoden /Balzert 1998/

Graham beschreibt die ***Inspektion*** wie folgt /Gilb, Graham 1993/:

- *Definition:* Manuelle, formalisierte Prüfmethode, um schwere Defekte in schriftlichen Dokumenten anhand von Referenzunterlagen zu identifizieren und durch den Autor beheben zu lassen.
- *Ziel der Prüfung:* Identifikation von Defekten im Prüfobjekt unter Berücksichtigung des Ursprungsprodukts, aus dem das Prüfobjekt entsprechend den Entwicklungsregeln erstellt wurde. Die Verbesserung der Entwicklungsregeln und des Entwicklungsprozesses ist ebenfalls Ziel der Prüfung.
- *Objekte der Prüfung:* Produkte und Teilprodukte (Dokumente) einschließlich des Prozesses ihrer Erstellung (Erstellungsregeln).
- *Referenzunterlagen für die Prüfung (Bezugsobjekte):* Ursprungsprodukt, aus dem das Prüfobjekt entsteht; Erstellungsregeln für das Prüfobjekt, Checklisten für die Erstellung.
- *Beschreibungsform der Prüf- und Bezugsobjekte:* Prüfobjekte: informal (z.B. Pflichtenheft), semiformal (z.B. Pseudocode) und formal (z.B. Quellcode); Bezugsobjekte: informal (z.B. Checklisten), semiformal (z.B. Pseudocode) und formal (z.B. Quellcode).
- *Ergebnisse:* Formalisiertes Inspektionsprotokoll und Fehlerklassifizierung (schwer, leicht), Fragen an den Autor und Prozeßverbesserungs-Vorschläge; Inspektionsmetriken, überarbeitetes Prüfobjekt.
- *Vorgehensweise:* Menschliche Begutachtung.
- *Ablauf der Prüfung:* Statische Prüfung, d.h. in der Reihenfolge der Ausschreibung des Prüfobjekts.
- *Vollständigkeit der Prüfung:* Stichprobenartig.
- *Teilnehmer:* Moderator, Autor, (Vorleser), Protokollführer, Inspektoren; insgesamt 3-7 Teilnehmer (wenn 3: Moderator/Protokollführer, Inspektor, Autor).
- *Durchführung:* Eingangsprüfung, Inspektionsplanung, optionale Einführungssitzung (Vorstellung von Prüfobjekt und Umfeld), individuelle Vorbereitung und Prüfung (jeder Inspektor prüft das Prüfobjekt nach den ihm zugeteilten Aspekten), Inspektionssitzung (jeder Inspektor nennt seine Prüfergebnisse, gemeinsam werden weitere Defekte identifiziert), Autor überarbeitet Prüfobjekt, Moderator nimmt eine Nachprüfung vor und gibt das Prüfobjekt anhand definierter Kriterien frei oder weist es zurück.
- *Aufwand:* Individuelle Vorbereitung: ca. 1 Seite/Stunde pro Inspektor. Inspektionssitzung: max. 2 Stunden (ca. 1 Seite/Stunde).
- *Nutzen:* Individuelle Prüfung: 80% der Gesamtdefekte identifiziert. Inspektionssitzung: 20% der Gesamtdefekte identifiziert.

/Frühauf, Ludewig, 1995/ und /Wallmüller 1990/ beschreiben ***Reviews*** wie folgt:

- *Definition:* Manuelle, semiformale Prüfmethode, um Stärken und Schwächen eines schriftlichen Dokuments anhand von Referenzunterlagen zu identifizieren und durch den Autor beheben zu lassen.
- *Ziel der Prüfung:* Feststellung von Mängeln, Fehler, Inkonsistenzen, Unvollständigkeiten, Verstößen gegen Vorgaben, Richtlinien, Standards, Pläne; formale Planung und Strukturierung der Bewertungsprozesse und formale Abnahme des Prüfobjekts.
- *Objekte der Prüfung:* Jeder in sich abgeschlossene, für Menschen lesbare, Teil von Software, z.B. ein einzelnes Dokument, Quellcode, ein Klassendiagramm.

- *Referenzunterlagen für die Prüfung (Bezugsobjekte):* Vorgaben für die Erstellung des Prüfobjekts; relevante Richtlinien und Standards; Fragenkataloge mit Listen von Fragen, die im Review beantwortet werden sollen.
- *Beschreibungsform der Prüf- und Bezugsobjekte:* Prüfobjekte: informal, semiformal und formal; Bezugsobjekte: informal, semiformal und formal.
- *Ergebnisse:* Review-Protokolle, Empfehlung über Freigabe an den Manager, überarbeitetes Prüfobjekt.
- *Vorgehensweise:* Menschliche Begutachtung.
- *Ablauf der Prüfung:* Statische Prüfung, d.h. in der Reihenfolge der Aufschreibung des Prüfobjekts.
- *Vollständigkeit der Prüfung:* Stichprobenartig.
- *Teilnehmer:* Review-Team bestehend aus 4-7 Personen: Moderator, Autor, Protokollführer, 2-5 Gutacher.
- *Durchführung:* Eingangsprüfung, optionale Einführungssitzung (Vorstellung von Prüfobjekt und Umfeld), individuelle Vorbereitung (jeder Gutacher prüft das Prüfobjekt nach den ihm zugeteilten Aspekten), Review-Sitzung (Bewertung durch die Gutacher, max. 2 Stunden Dauer, keine Probleme lösen oder beheben; Autor ist passiv), Überarbeitung des Prüfobjekts (durch den Autor); bei gravierenden Mängeln erneute Review-Sitzung.
- *Aufwand:* 15-20% des Erstellungsaufwands des Prüfobjekts.
- *Nutzen:* 60%-70% der Fehler in einem Dokument werden gefunden. Reduktion der Fehlerkosten in der Entwicklung von 75% und mehr. Nettoeinsparungen für die Entwicklung ca. 20%, für die Wartung ca. 30%.

Balzert beschreibt ***Walkthroughs*** wie folgt /Balzert 1998/:

- *Definition:* Manuelle, informale Prüfmethode, um Fehler, Defekte, Unklarheiten und Probleme in schriftlichen Dokument zu identifizieren. Der Autor präsentiert das einem Dokument in einer Sitzung den Gutachtern.
- *Ziel der Prüfung:* Identifikation von Fehlern, Defekten, Unklarheiten und Problemen. Ausbildung von Benutzern und Mitarbeitern. Die Überarbeitung des Prüfobjekts ist nicht Ziel der Prüfung.
- *Objekte der Prüfung:* Produkte und Teilprodukte (z.B. Dokumente).
- *Referenzunterlagen für die Prüfung (Bezugsobjekte):* Prüfobjekte: informal (z.B. Pflichtenheft), semiformal (z.B. Pseudocode) und formal (z.B. Quellcode).
- *Ergebnisse:* Walkthrough-Protokoll
- *Vorgehensweise:* Menschliche Begutachtung.
- *Ablauf der Prüfung:* Dynamische Prüfung, d.h. in der Reihenfolge der Ausführung der Prüfobjekte.
- *Vollständigkeit der Prüfung:* Stichprobenartig.
- *Teilnehmer:* Autor (gleichzeitig Moderator), Gutachter.
- *Durchführung:* Vorbereitung (optional), Gruppensitzung (Autor präsentiert Prüfobjekt Schritt für Schritt, Gutachter stellen vorbereitete oder spontane Fragen, Autor antwortet).
- *Aufwand:* Geringer Aufwand (Sitzungszeit, u. U. Vorbereitungszeit).
- *Nutzen:* Gering, verglichen mit Inspektionen und Reviews, aber höher als bei fehlender Überprüfung.

Prüfkriterien definieren

In diesem Schritt werden Kriterien zur Prüfung eines Prüfgegenstands festgelegt. Zum Beispiel wäre eine ausreichende Kommentierung ein Kriterium für die Inspektion von Quellcodes. Dieser Schritt basiert auf dem QS-Plan. Für manuelle Prüfmethoden, wie Inspektionen, Reviews oder Walkthroughs, empfiehlt es sich, entsprechende Checklisten zu definieren.

Die nächste Abbildung zeigt eine Checkliste zur Inspektion von Dokumenten.

Lfd. Nr.	**Anforderung**	**Erfüllt**	**Bemerkung**
1	Titel	☐ ja ☐ nein	
2	Version	☐ ja ☐ nein	
3	Ersteller	☐ ja ☐ nein	
4	Erstelldatum	☐ ja ☐ nein	
5	Aktualisierungsdatum	☐ ja ☐ nein	
6	Änderungshistorie	☐ ja ☐ nein	
7	Prüf- und Freigabevermerke	☐ ja ☐ nein	
8	Verteiler	☐ ja ☐ nein	
9	Ablage / Verzeichnis /Pfad	☐ ja ☐ nein	
10	Copy-Right	☐ ja ☐ nein	
11	Firmen-Logo	☐ ja ☐ nein	
12	Dokumentenidentfikation	☐ ja ☐ nein	
13	Inhaltsübersicht	☐ ja ☐ nein	
14	...	...	...

Abb. 3: Checkliste für die Inspektion von Dokumenten

Termine und Ressourcen planen

Für die vorgesehenen Prüfungen sind im Rahmen dieses Schritts die Termine und der Ressourceneinsatz zu planen. Das Ergebnis der Planung muß mit dem Projektplan abgeglichen werden. Die Planung wird nach und nach verfeinert, indem zuerst für das System, dann für die Komponenten und zuletzt für die Klassen geplant wird. Bei der Planung von Terminen und dem Einsatz von Mitarbeitern kann wieder das Aufwandsschätz-Verfahren CEOS genutzt werden. Mit CEOS läßt sich der Aufwand für Analyse, Entwurf, Implementierung und operationeller Einsatz schätzen. Diese Schätzungen berücksichtigen den Aufwand für Prüfungen, die in der Regel nach einer Phase wie Analyse, Entwurf, Implementierung und operationeller Einsatz stattfinden.

Bei der personellen Auswahl der Prüfenden ist zu beachten, daß der Entwickler eines Produkts am schlechtesten zur Prüfung des Produkts geeignet ist. Andererseits muß sichergestellt werden, daß ein Entwickler nicht eigene Aufgaben an die Qualitätssicherung abschiebt.

Bei der personellen Besetzung der Qualitätssicherung gibt es prinzipiell drei Alternativen:

- *Nur QS:* Hierbei sind Mitarbeiter nur an der Qualitätssicherung beteiligt und nicht an der Entwicklung. Das sorgt für einen hohen Spezialisierungsgrad der Mitarbeiter, trägt aber zur Entfremdung von der Anwendungsentwicklung bei.
- *Rotation:* In festgelegten Abständen rotieren die Mitarbeiter zwischen Entwicklung und Qualitätssicherung. Dieses Vorgehen stellt einen systematischen Wissenstransfer sicher, kann aber zur Überforderung der Mitarbeiter führen.
- *QS und Entwicklung parallel:* Bei diesem Vorgehen ist jeder Mitarbeiter sowohl an einer Entwicklung als auch an der Qualitätssicherung von einem oder mehreren Bausteinen beteiligt. Damit ist eine flexibler Personaleinsatz möglich. Die Vermischung der Entwicklung und Qualitätssicherung kann aber dazu führen, daß keine Arbeit mehr richtig gemacht wird.

Ergebnisse

Das Ergebnis der ersten drei Schritte wird in dem QS-Handbuch festgehalten, für das die nächste Abbildung ein Gliederungsschema vorgibt. Angaben bezüglich Termine und Ressourcen werden im QS-Plan (vgl. Abschnitt 1 „Projektmanagement“, „Qualitätssicherung planen“) gepflegt.

1 Prüfgegenstände

⇨ Grenzen Sie hier die zu überprüfenden Produkte und Aktivitäten ab.

[**Hier** klicken]

2 Prüfverfahren

⇨ Beschreiben Sie hier, welche Prüfverfahren (z.B. Reviews oder Funktionstests) für die Prüfgegenstände verwendet werden.

[**Hier** klicken]

3 Prüfkriterien

⇨ Beschreiben Sie die Aspekte eines Prüfgegenstands, der schwerpunktmäßig geprüft werden sollen

Abb. 4: Gliederungsschema für das QS-Handbuch

Hinweise und Tipps

Neben den genannten manuellen Prüfmethoden gibt es noch einige Varianten. Bei der *Stellungsnahme* gibt der Autor Kopien des Prüfobjekts an Kollegen weiter und bittet sie um Kommentare. Diese Methode ist sehr einfach anzuwenden und führt zu guten Ergebnissen. Beim *Round Robin Review* werden Argumente für die Güte des Prüfobjekts gesammelt. Finden sich nicht genügend überzeugende Argumente, so wurde ein Problem identifiziert. Beim *Peer Review* wird das Review-Team in einem Raum „eingesperrt“ und mit der Erstellung eines Gutachtens beauftragt. Die Organisation, d.h. Verteilung der Aufgaben und Vorgehensweise, wird von den Gutachtern selbst bestimmt.

3.2 Prüfung vorbereiten

Zweck

Zur Durchführung von Qualitätssicherungs-Maßnahmen sind in der Regel Vorbereitungen nötig. Dazu gehört die Einrichtung der Prüfumgebung, die Definition von Prüffällen und die Entwicklung von Prüfprozeduren.

Beteiligte

Ausführend: Qualitätsprüfer
Beratend: Projektleiter

Vorgehen

Im Detail sind folgende Schritte zu durchlaufen:

- Prüfumgebung einrichten,
- Prüffälle festlegen,
- Prüfprozeduren entwickeln.

Da wir beim Testen von Klassen, Komponenten und des Systems ausführlich auf Testumgebung, Testfälle und Testprozeduren eingegangen sind, beschreiben wir im folgenden die oben genannten Vorbereitungsschritte auf einem allgemein gültigen Niveau.

Prüfumgebung einrichten

Zur Prüfung eines Produkts, außerhalb der Entwicklungsumgebung, muß in der Regel zunächst die software- und hardwaretechnische Umgebung eingerichtet werden. Das Bereitstellen des Arbeitsbereichs für die Qualitätsprüfer gehört zur Prüfumgebung. Zum Beispiel müssen gegebenenfalls Werkzeuge installiert und eingerichtet werden oder benötigte Hardware-Komponenten beschafft werden.

Prüffälle festlegen

Beim Entwickeln von Prüffällen können folgende Grundsätze behilflich sein:

- Prüffälle sollten stets in einer nachvollziehbaren Weise auf Anforderungen zurück verfolgbar sein.
- Prüfungen sollten kontinuierlich von der Analyse bis zum operationellen Einsatz von Bausteinen erfolgen. Damit wird vermieden, daß Fehler sich fortpflanzen und ihre Beseitigung erschwert wird.
- Sowohl bei der Spezifikation als auch bei der Durchführung von Prüfungen sollte stets mit Prüfgegenständen von geringem Umfang begonnen werden. Schrittweise sollte dann zu größeren Produkten übergegangen werden. So kann die Komplexität reduziert werden.

- Eine vollständige überdeckende Prüfung ist nicht möglich. Dies illustrieren wir anhand der Abbildung 5 /Meyers 1979/:

  Wenn man annimmt, daß der Zeitbedarf für das Planen, Vorbereiten und Durchführen von Testfällen ca. 5 Minuten dauert, dann würde man ca. 1 Millionen Jahre brauchen, um jeden Pfad des Programms in der Abbildung 5 zu testen.

  Daher sollte bei der Entwicklung von Testfällen eine angemessene Testüberdeckung angestrebt werden.

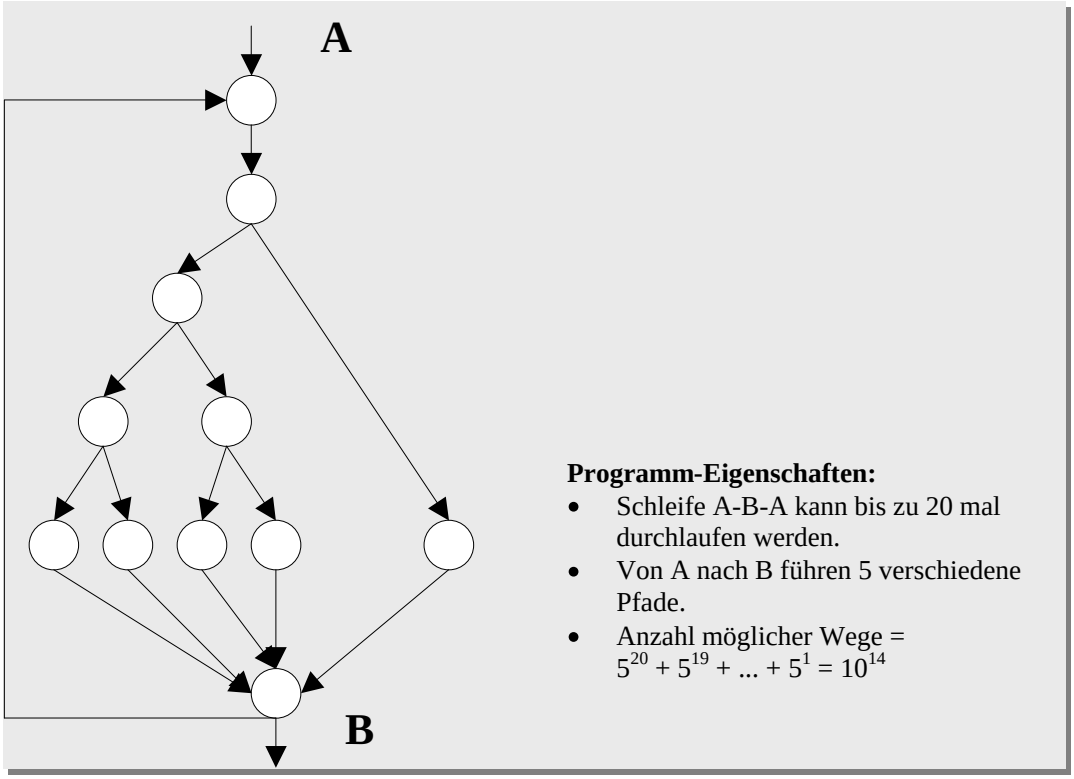


Abb. 5: Beispiel zur Demonstration der vollständigen Überdeckung

Prüfprozeduren entwickeln

In diesem Schritt wird für die Qualitätsprüfer eine detaillierte Arbeitsanleitung in schriftlicher Form angefertigt. Das Prüfverfahren muß genau beschrieben werden, und es muß auf maschinelle Routinen, wie z.B. Automatisierung von Testverfahren, hingewiesen werden.

Ergebnisse

Die Dokumentation von Test- und Prüffällen findet in einer schriftlichen Form statt. Für Testfälle wurden im Rahmen der Software-Entwicklung im EOS-Modell entsprechende Beschreibungsschemas vorgeschlagen. Für Prüffälle, insbesondere für manuelle Prüfmethoden, kann das folgende Beschreibungsschema verwendet werden (Abb. 6).

Hinweise und Tipps

Zur Selbstprüfung definieren oft die Entwickler eines Produkts Prüffälle. Diese können als eine gute Grundlage für die Qualitätssicherung genutzt werden.

Prüfungsnummer	⇨ Da ein Prüffall in der Regel mehrfach getestet wird, empfiehlt sich die Vergabe eines identifizierenden Merkmals pro Prüfung. [**Hier** klicken und Text eingeben]	
Prüfdatum	⇨ Geben sie hier an, wann die Prüfung stattgefunden hat. [**Hier** klicken und Text eingeben]	
Prüfteam	⇨ Führen Sie hier die Namen der an der Prüfung beteiligten Personen auf. [**Hier** klicken und Text eingeben]	
Prüfgegenstand	⇨ Machen Sie deutlich, was der Gegenstand der Prüfung ist, in dem Sie ds zu überprüfende Produkt abgrenzen. [**Hier** klicken und Text eingeben]	
ID	⇨ Vergeben Sie hier ein eindeutiges Merkmal zur Identifikation des Prüffalls, denn pro Szenario kann es mehrere Testfälle geben. [**Hier** klicken und Text eingeben]	
Zweck	⇨ Was ist der Zweck des Prüffalls? Was soll konkret nachgewiesen werden? [**Hier** klicken und Text eingeben]	
Beschreibung	⇨ Beschreiben Sie den Prüffall ausführlich und nachvollziehbar. [**Hier** klicken und Text eingeben]	
Ausgangs-situation	⇨ Listen Sie hier alle Rahmen- und Vorbedingungen auf, die für den Ablauf der Prüfung notwendig sind. [**Hier** klicken und Text eingeben]	
Prüfverfahren	⇨ Welches Verfahren für die Prüfung haben Sie gewählt (z.B. Inspektion oder Review)? [**Hier** klicken und Text eingeben]	
Prüfprozedur	⇨ Beschreiben Sie hier das Vorgehen beim gewählten Prüfverfahren. [**Hier** klicken und Text eingeben]	
Zusammen-fassung	⇨ Geben Sie hier eine Zusammenfassung des Prüfverlaufs und der erreichten Ergebnisse. [**Hier** klicken und Text eingeben]	
Bewertung/ Empfehlung	☐ fehlerfreier Ablauf ☐ leichte Fehler ☐ schwere Fehler ☐ fatale Fehler	☐ Ergebnis akzeptieren, Test abschließen ☐ Korrektur ohne Testwiederholung ☐ Korrektur und Testwiederholung
Hinweise zur Fehlerbehebung	⇨ Sammeln Sie hier alle Hinweise und Bemerkungen, die bei der Ermittlung der Fehlersuche und bei der Fehlerbehebung von Nutzen sein können. [**Hier** klicken und Text eingeben]	

Abb. 6: Beschreibungsschema für Prüffälle

3.3 Prüfung durchführen

Zweck

Das Ziel dieses Aktivitätstyps ist es, ein Produkt hinsichtlich der formalen und inhaltlichen Qualität zu prüfen.

Beteiligte

Ausführend: Qualitätsprüfer
Beratend: Projektleiter

Vorgehen

Es sind folgende Schritte auszuführen:

- Produkt formal prüfen,
- Produkt inhaltlich prüfen,
- Prüfergebnisse dokumentieren.

Am Ende dieses Schrittes wird entschieden, ob das geprüfte Produkt eine akzeptable Qualität hat, oder ob es nachbearbeitet werden muß.

Produkt formal prüfen

Bevor eine inhaltliche Prüfung durchgeführt wird, sollte das Produkt formale Qualitätskriterien erfüllen. Es ist zu klären, ob das Produkt vollständig vorliegt und verständlich gestaltet ist. Außerdem ist zu überprüfen, ob vorgegebene Standards und Richtlinien eingehalten sind.

Produkt inhaltlich prüfen

Wenn die formale Prüfung erfolgreich abgeschlossen ist, folgt eine inhaltliche Prüfung. Dazu werden die entwickelten Prüffälle herangezogen und anhand der Prüfprozedur die Prüfung durchgeführt.

Prüfergebnisse dokumentieren

Im Rahmen dieses Schritts wird der Verlauf der Prüfung dokumentiert. In Abhängigkeit von den erzielten Ergebnissen ist zu entscheiden, ob eine Überarbeitung veranlaßt werden muß.

Ergebnisse

Das Beschreibungsschema in Abbildung 6 kann genutzt werden, um die Ergebnisse der Prüfung zu dokumentieren.

Die Qualitätssicherung im EOS-Modell gliedert sich in den Schritten „Prüfung planen“, „Prüfung vorbereiten“ und „Prüfung durchführen“. Im ersten Schritt wird die Prüfung eines Produkts oder einer Aktivität im Detail geplant, indem der Prüfgegenstand und das Prüfverfahren festgelegt werden. Das Ergebnis der Planung wird in einem QS-Handbuch verbindlich festgehalten. Bevor eine Prüfung durchgeführt werden kann, müssen Vorbereitungen getroffen werden. Daher werden im zweiten Schritt die Prüfumgebung eingerichtet, die Prüffälle definiert und die Prüfprozeduren entwickelt. Im letzten Schritt werden die vorgesehenen Prüfungen durchgeführt und die Ergebnisse dokumentiert. Anhand der Ergebnisse ist über eine Nachbearbeitung zu entscheiden.

4 Konfigurationsmanagement

Eine *Konfiguration* ist eine benannte und zusammengehörende Menge von Produkten. Konfigurationen werden benötigt, um Probleme bei den Änderungen und Erweiterungen von Produkten besser zu regeln. Im allgemeinen werden folgende Probleme beklagt, die das Konfigurationsmanagement zu beheben versucht /Balzert 1998/:

- *Änderungen können Chaos verursachen (Änderungsmangement):* Wer welche Änderungen, zum welchen Zweck und zum welchen Zeitpunkt vorgenommen hat, bleibt unklar.
  Das Konfigurationsmanagement reduziert dieses Problem, indem die Historie der Änderungen aufgezeichnet wird.
- *Fehlerkorrekturen sind nicht transparent (Fehlermanagement):* Welche Fehler bereits behoben worden sind, bleibt unklar.
  Das Konfigurationsmanagement überwacht vorgenommene Änderungen.
- *Zusammenstellen von Konfigurationen ist schwierig (Versionskontrolle):* Oft führen Verbesserungen zu weiteren Fehler. Daher müssen gegebenenfalls Korrekturen rückgängig gemacht werden. Das Konfigurationsmanagement unterstützt das Zusammenstellen von konsistenten Konfigurationen durch eine automatische Versions- und Konfigurationsselektion.
- *Freigaben sind schwer zu verwalten (Freigabeverfahren):* Ob die geänderten Bausteine vollständig neu übersetzt wurden und ob die Kunden den neuesten Stand haben, bleibt unklar.
  Das Konfigurationsmanagement sorgt dafür, daß keine Arbeitsschritte vergessen werden.

Die genannten Probleme zeigen deutlich die Notwendigkeit eines Konfigurations-managements. Es sorgt dafür, daß die Projektbeteiligten stets auf einem definierten, einheitlichen Entwicklungsstand arbeiten. Das Konfigurationsmanagement begleitet den gesamten Entwicklungsprozeß und betrifft alle Produkte, die im Rahmen eines Bausteins entstehen, wie z.B. Quellcode, Diagramme oder Textdokumente. Damit ist das Konfigurationsmanagement im EOS-Modell bausteinorientiert, d.h. daß Bausteine die primären Gegenstände des Konfigurationsmanagement sind. Die durch die Phasen Analyse, Entwurf, Implementierung und operationeller Einsatz eines Bausteins entstehenden Produkte werden im Rahmen des Bausteins verwaltet.

Jedes Produkt durchläuft in seinem Lebenszyklus eine Reihe von Entwicklungsstadien. Ein Entwicklungsstand zu einem speziellen Zeitpunkt wird als eine *Version* des Produkts bezeichnet.

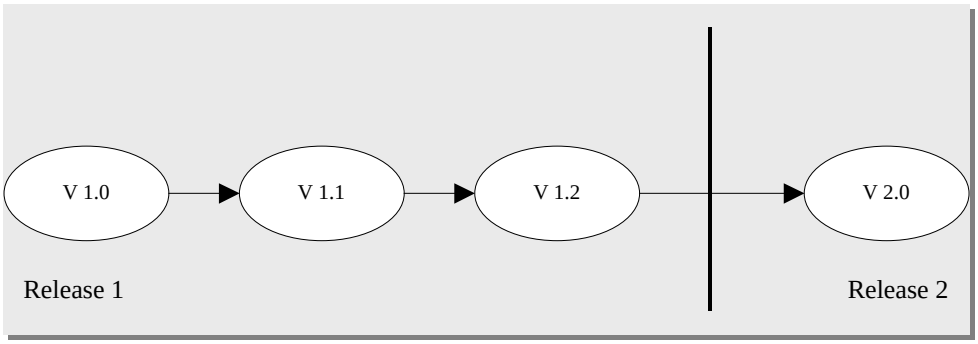


Abb. 1: Versionszählung

In der Regel wird eine Version durch eine Nummer beschrieben, die aus einem *Release*- und einem *Level-Teil* besteht. Die einstellige Release-Nummer ist durch einen Punkt von der Level-Nummer getrennt, wie z.B. 1.3 oder 2.14 (1 bzw. 2 sind Release-Nummern und 3 bzw. 14 Level-Nummern). Bei größeren Änderungen wird die Release-Nummer erhöht und bei kleineren, formalen oder inhaltlichen Änderungen die Level-Nummer (Abb. 1).

In der Praxis reicht aber eine Versions-Bildung nicht aus. Denn oft werden weitere Ausprägungen eines Bausteins benötigt. Diese werden als *Varianten* bezeichnet. Zum Beispiel könnte ein Anwendungssystem für verschiedene Hardwareplattformen oder unterschiedliche Betriebssysteme entwickelt werden, so daß mehrere Varianten vorliegen würden. Die nächste Abbildung veranschaulicht diesen Sachverhalt

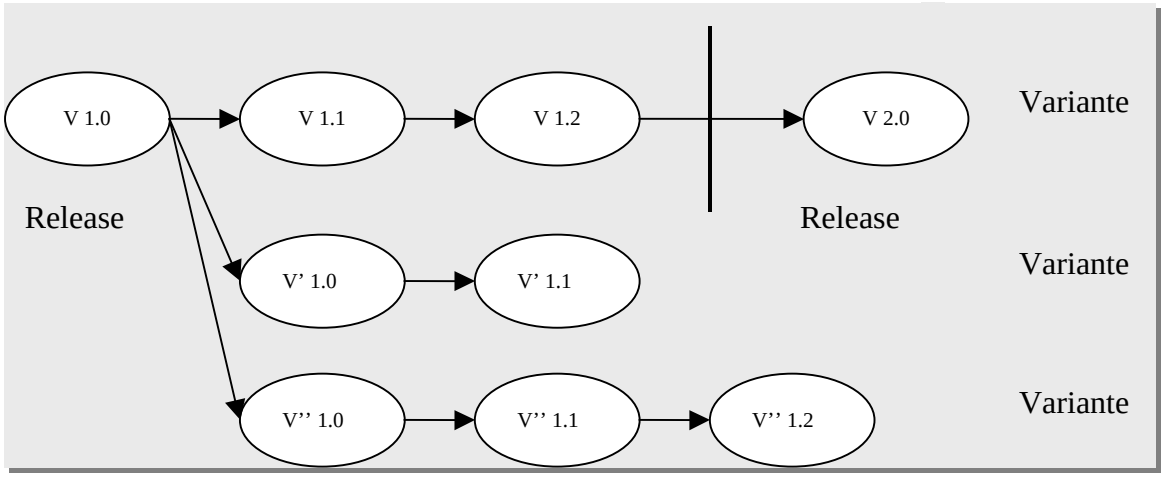


Abb. 2: Varianten eines Produktes

Die Realität ist aber leider noch komplizierter. Denn während des Entwicklungsprozesses entstehen eine Vielzahl von Produkten, die jeweils mehrere Versionen und unter Umständen auch Variationen haben können. Passende Produktversionen werden zu einer Konfiguration zusammengefaßt. Das wird mit der nächsten Abbildung illustriert.

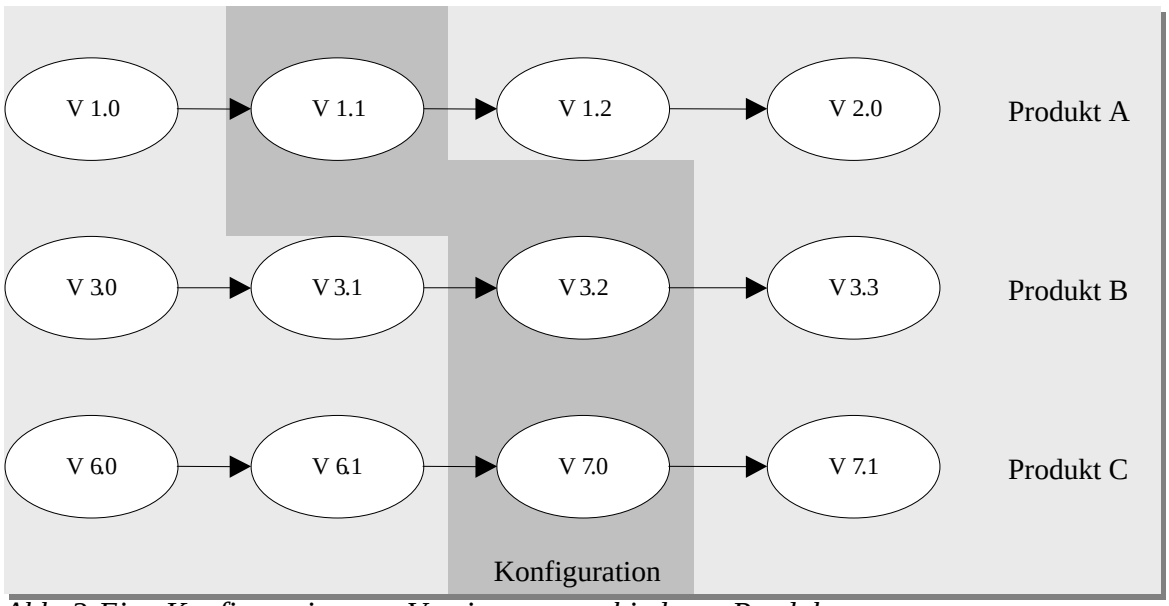


Abb. 3:Eine Konfiguration aus Versionen verschiedener Produkte

Im EOS-Modell verstehen wir unter einer Konfiguration einen Behälter für alle Produkte (z.B. Diagramme, Textdokumente), die beim Entwicklungsprozeß eines Bausteins (System, Komponente/Subsystem, Klasse) entstehen. Solche Behälter werden von KM-Werkzeugen zur Verfügung gestellt (z.B. im Form eines Dateiverzeichnisses). Eine Konfiguration für eine Komponente X_1 enthält z.B. Klassen- und Sequenzdiagramme, die für diese Komponente erstellt wurden. Falls X_1 aus Subkomponenten besteht, so sind die Konfigurationen der Subkomponenten in der Konfiguration von X_1 mitenthalten. Das bedeutet, daß die Konfigurationen im direkten Bezug zur Systemarchitektur stehen und sie widerspiegeln. Die nächste Abbildung veranschaulicht diesen *bausteinbasierten Konfigurations-Begriff* an einem Beispiel.

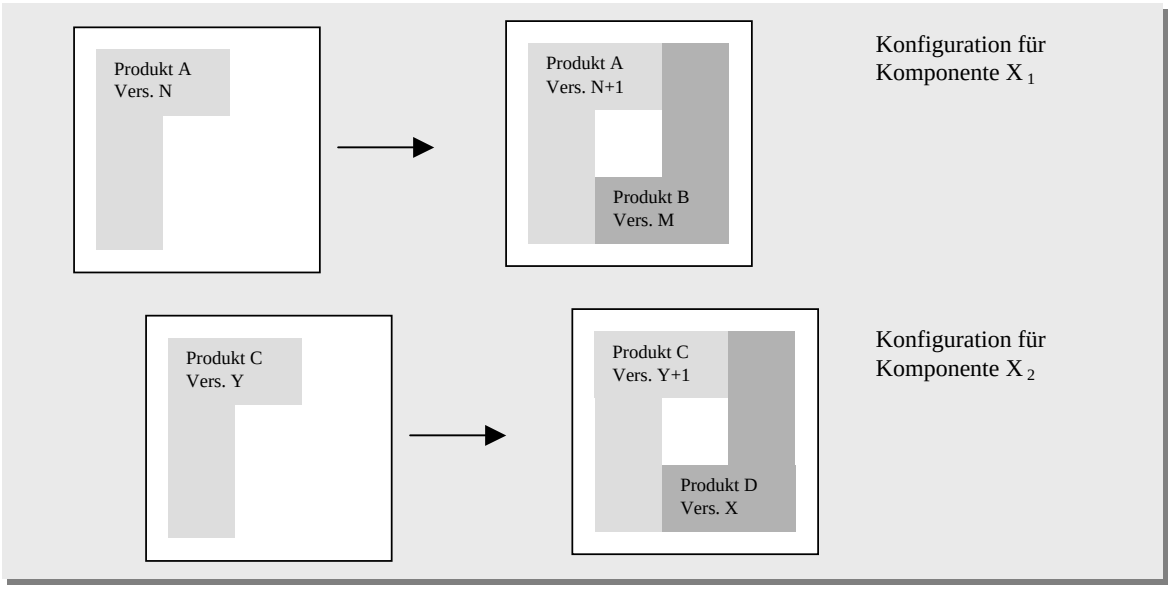


Abb. 4: Bausteinbasierten Konfiguration

Die obige Abbildung zeigt das Erzeugen und Fortschreiben von Konfigurationen für die zwei Komponenten. Im Verlauf der Zeit werden Produkte mit ihrer aktuellen Version in die Konfiguration abgelegt. Der Entwicklungsstand einer Komponente wird durch seine Produkte und ihre Versionen repräsentiert. Zur dauerhaften Dokumentation des Entwicklungsstands wird die Konfiguration „eingefroren“. Nach dem „Einfrieren“ wird eine neue Version der Konfiguration der Komponente erstellt. Einen typischen Zeitpunkt für das „Einfrieren“ einer Konfiguration eines Bausteins bildet z.B. der Abschluß der Analyse, des Entwurf, der Implementierung oder des operationellen Einsatzes. Analog zur Bildung von Versionen einer Konfiguration verläuft auch das Erzeugen von Varianten. Das Erzeugen von Versionen und Varianten vollzieht sich auf Produkte und Konfigurationen.

Voraussetzung für die Kontrolle eines bausteinbasierenden Konfigurationsmanagements ist die Versionskontrolle, das heißt das Anlegen und Verwalten von Versionen und Varianten. Hierzu ist die eindeutige Identifikation des zu verwaltenden Elements notwendig, die in der Regel maschinell vergeben wird. Das bausteinbasierte Konfigurationsmanagement bildet auch die Grundlage zur Wiederverwendung von Bausteinen. Diese hängt aber in der Regel vom Leistungsumfang des gewählten KM-Werkzeugs ab.

Wie eingangs erwähnt, muß das Konfigurationsmanagement Änderungen systematisch koordinieren und kontrollieren. Dies erfolgt im Allgemeinen durch das *Checkin/Checkout-Modell*. Checkout ist eine Ausbuche-Operation, die eine Kopie eines Produkts aus der Bausteinbibliothek holt und es für den Ausbucher reserviert. Der Ausbucher bearbeitet das

Produkt und gibt das geänderte Produkt durch eine Einbuche-Operation (*Checkin*) wieder zurück. Beim Einbuchen findet auch eine Historisierung statt, d.h. der Autor, Einbuchungszeit und Änderungen werden vermerkt. Zur Speicherung von Änderungen wird im Checkin/Checkout-Modell häufig die *Deltatechnik* eingesetzt. Hierbei werden statt einer kompletten Kopie nur die Unterschiede (*Deltas*) von zeitlich aufeinanderfolgenden Produkten abgespeichert. Das Checkin/Checkout-Modell erfordert eine *Zugriffs- und Synchronisationskontrolle*.

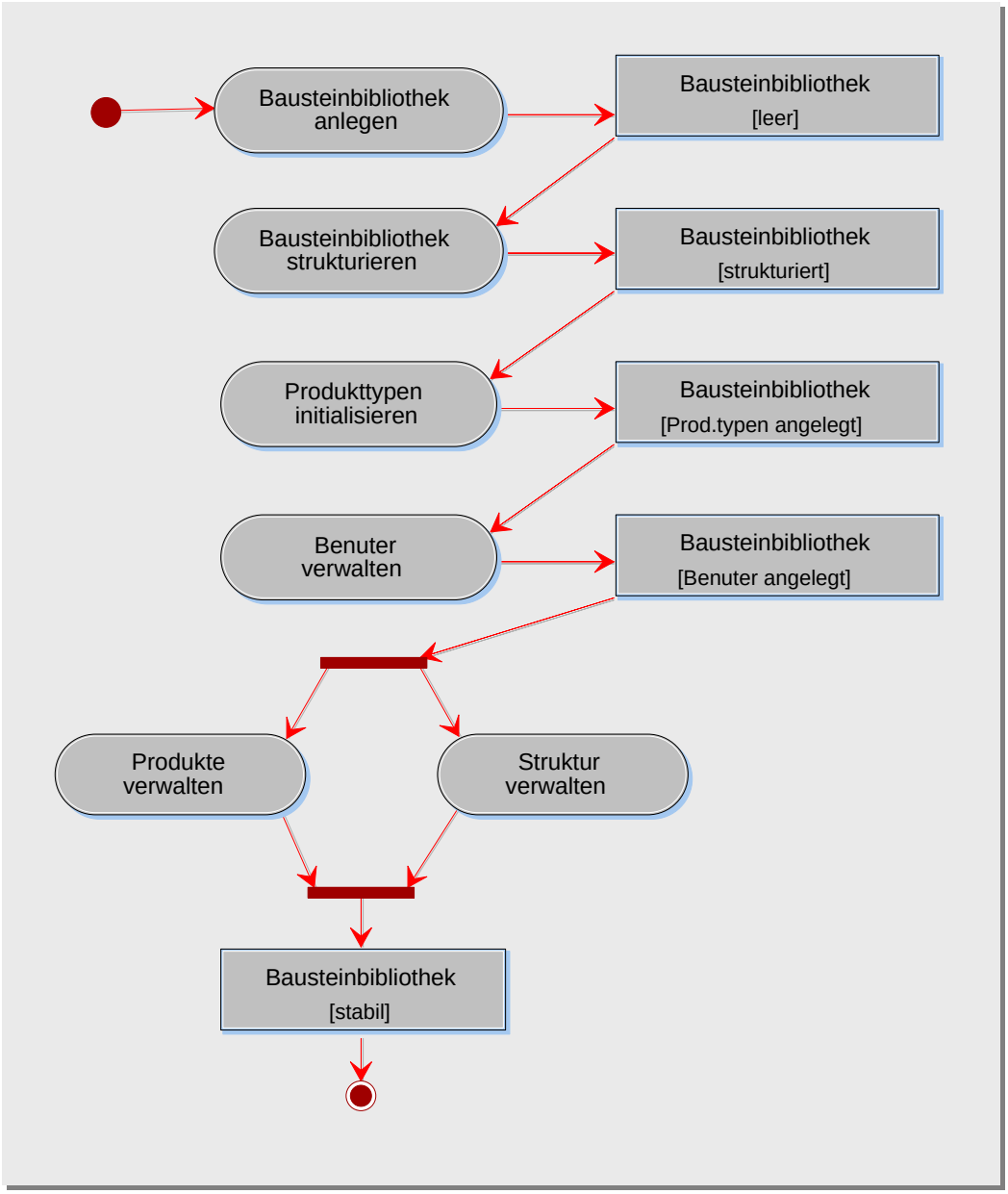


Abb. 5: Aktivitätstypen des Konfigurationsmanagements

Die Zugriffskontrolle stellt sicher, daß nur ein autorisierter Projektarbeiter auf ein Produkt zugreifen darf. Die Zugriffsberechtigung hängt nicht nur von den Benutzerrechten, sondern auch vom Zustand des Produkts ab. Ein bereits qualitätsgesichertes Produkt darf z.B. zunächst nicht mehr geändert werden. Damit zwei verschiedene Bearbeiter ihre Ergebnisse gegenseitig nicht überschreiben, wird die Synchronisationskontrolle benötigt. Ein häufig realisierter Mechanismus zur Synchronisation besteht aus dem Sperren des ausgeliehenen Produkts. Die

Projektmitglieder dürfen zwar lesend auf das Produkt zugreifen, aber nicht mehr schreibend. Ein anderer Mechanismus nutzt das Abspalten von Entwicklungsästen.

Zur Durchführung des Konfigurationsmanagements sind die in der Abbildung 5 dargestellten Aktivitäten durchzuführen. Im ersten Schritt wird die Bausteinbibliothek strukturiert, d.h. die vorliegende Systemarchitektur, die Produkttypen und die Projektbeteiligten werden in ihr abgebildet. Im zweiten Schritt wird die Bausteinbibliothek hinsichtlich der abgelegten Produkte und der angelegten Struktur kontinuierlich dem Entwicklungsstand angepaßt.

4.1 Strukturierung der Bausteinbibliothek

Für ein neues Projekt muß die Bausteinbibliothek geeignet strukturiert werden. Diese Struktur spiegelt die Systemarchitektur wider und muß kontinuierlich an die Veränderungen der Systemarchitektur angepaßt werden. Folgende Aktivitätstypen sind für die Strukturierung der Bausteinbibliothek nötig:

- Bausteinbibliothek anlegen,
- Bausteinbibliothek strukturieren,
- Produkttypen initialisieren,
- Benutzer verwalten.

Diese Aktivitätstypen hängen natürlich stark vom gewählten KM-Werkzeug ab und werden im folgenden allgemein beschrieben.

4.1.1 Bausteinbibliothek anlegen

Zweck

Im Rahmen dieses Aktivitätstyps wird die Bausteinbibliothek samt der notwendigen Identifikationsmechanismen eingerichtet. Damit wird die Grundlage für die Verwaltung der Ergebnisse des Entwicklungsprozesses geschaffen. Denn die erzeugten Produkte werden mit Hilfe der Bausteinbibliothek systematisch koordiniert und verwaltet, so daß eine effektive Wiederverwendung ebenfalls unterstützt wird.

Beteiligte

Ausführend: Projektbibliothekar
Mitwirkend: Projektleiter

Vorgehen

Dieser Aktivitätstyp läßt sich in folgenden Schritten gliedern:

- Bausteinbibliothek erzeugen,
- Benutzergruppen und Benutzer definieren,
- Produkttyp- und zustandsunabhängige Benutzerrechte vergeben,
- Zulässige Zustände abbilden.

Bausteinbibliothek erzeugen

Der Umfang dieses Aktivitätstyps hängt vom ausgewählten KM-Werkzeug ab. In der Regel wird ein solches Werkzeug im Unternehmen installiert vorliegen. Das durchzuführende Projekt muß dann bloß noch angelegt werden.

Benutzergruppen und Benutzer definieren

Die Benutzergruppen lassen sich von den Rollen ableiten, die für das Projekt definiert worden sind. Es ist zu entscheiden, welche der Benutzergruppen Zugang zur Bausteinbibliothek besitzen müssen. Wenn die personelle Besetzung der Rollen bekannt ist, dann können Zugangsberechtigung für diese Personen eingerichtet werden.

Produkttyp- und zustandsunabhängige Benutzerrechte vergeben

Benutzerrechte sollten möglichst differenziert definiert werden. Daher sollten zunächst allgemeine, produkttyp- und zustandsunabhängige Benutzerrechte, wie z.B. das Ändern des eigenen Passworts, definiert und zugeteilt werden.

Zulässige Zustände abbilden

Produkte können im Verlauf ihres Lebenszyklus verschiedene Zustände einnehmen, die im Rahmen des Subprozesses „Projektmanagement“ („Konfigurationsmanagement planen“) definiert werden. Diese sollten nun detailliert werden und gegebenenfalls durch Zustandsautomaten dargestellt werden. Die Zustandsübergänge für die Entwicklungs-Bausteine verlaufen in der Regel nach dem gleichen Schema. Die nächste Abbildung illustriert ein Beispiel hierfür.

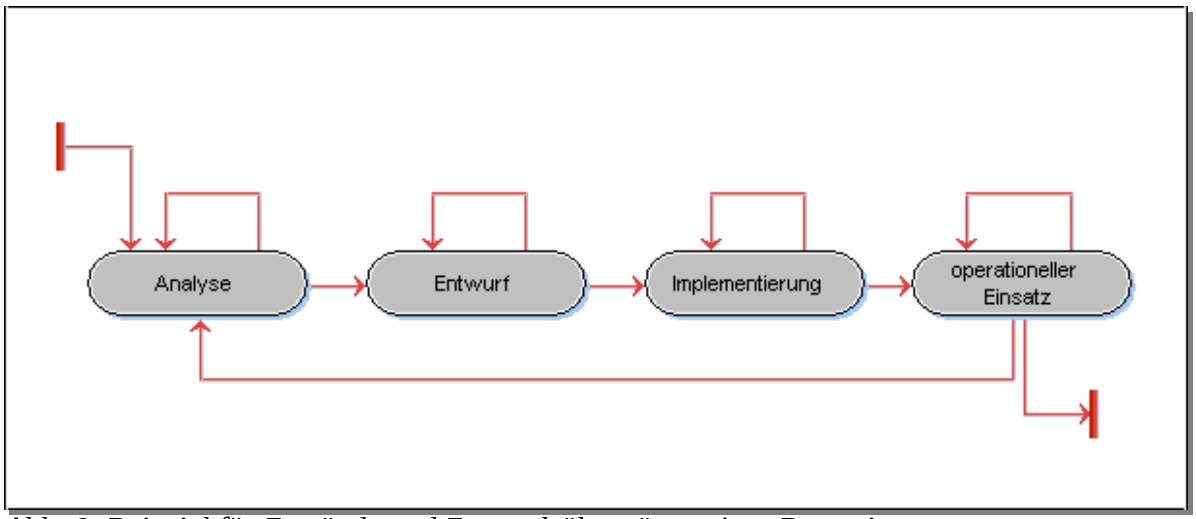


Abb. 6: Beispiel für Zustände und Zustandsübergänge eines Bausteins

Ergebnisse

Das Ergebnis ist eine leere Bausteinbibliothek mit vorgenommenen Grundeinstellungen.

4.1.2 Bausteinbibliothek strukturieren

Zweck

Das Ziel dieses Aktivitätstyps ist es, die Bausteinbibliothek so zu strukturieren, daß die Systemarchitektur wiedergegeben wird. Dazu müssen die bereits identifizierten Bausteine in der Bausteinbibliothek angelegt werden und entsprechende Benutzerrechte vergeben werden. Dieser Aktivitätstyp begleitet den Entwicklungsprozeß, da im Verlauf des Projekts die Systemarchitektur geändert und erweitert wird.

Beteiligte

Ausführend: Projektbibliothekar
Mitwirkend: Projektleiter

Vorgehen

Im Detail sind folgende Schritte zu durchlaufen:

- Bausteine anlegen,
- Zulässige Zustände für Bausteine festlegen,
- Baustein- und Zustandsabhängige Benutzerrechte vergeben.

Bausteine anlegen

Ausgehend von der groben Systemarchitektur, die im Rahmen des Subprozesses „Projektmanagement“ („Entwicklungsstrategie festlegen“) entwickelt wurde, werden die identifizierten Bausteine in der Bausteinbibliothek angelegt. Abbildung 7 veranschaulicht diesen Sachverhalt beispielhaft.

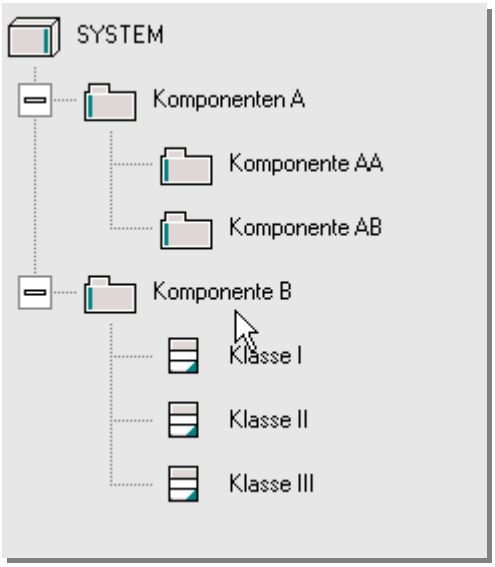


Abb. 7: Struktur der Bausteinbibliothek

Wiederverwendete Bausteine werden ebenfalls in der projektspezifischen Bausteinbibliothek aufgenommen.

Zulässige Zustände für Bausteine festlegen

Die Zustände, die ein Baustein im Verlauf seines Lebenszyklus annehmen kann, wurden in der Abbildung 6 beispielhaft dargestellt. Sollte ein Baustein einen anderen Zustandsautomaten aufweisen, so ist dieser zu dokumentieren und entsprechend in der Bausteinbibliothek abzubilden.

Baustein- und Zustandsabhängige Benutzerrechte vergeben

In Abhängigkeit von den vier Zuständen, die ein Baustein annehmen kann, werden Benutzerrechte für die Bausteine definiert. In der Regel wird ein Team Bearbeitungsrechte für eine oder mehrere Komponenten haben und ein einzelner Entwickler für eine oder mehrere Klassen.

Ergebnisse

Das Resultat ist eine strukturierte Bausteinbibliothek, die die Systemarchitektur abbildet.

4.1.3 Produkttypen initialisieren

Zweck

Im vorherigen Schritt wurden Bausteine in der Bausteinbibliothek eingerichtet. In diesem Schritt werden Produkttypen, die im Verlauf des Entwicklungsprozesses entstehen können, in der Bausteinbibliothek angelegt und vorbereitet. Nach diesem Schritt ist die Bausteinbibliothek zur Aufnahme von Produkten eingerichtet.

Beteiligte

Ausführend: Projektbibliothekar
Mitwirkend: Projektleiter

Vorgehen

Das Initialisieren von Produkttypen gliedert sich in den folgenden Aktivitätstypen:

- Produkttypen anlegen,
- Zulässige Zustände für Produkttypen festlegen,
- Produkttyp- und Zustandsabhängige Benutzerrechte vergeben.

Produkttypen anlegen

Produkttypen, die in der Bausteinbibliothek verwaltet werden sollen, werden angelegt. Hierbei ist die Zugehörigkeit der Produkttypen zu den Entwicklungs-Bausteinen zu beachten. Der Umfang dieses Aktivitätstyps hängt von den ausgewählten KM-Werkzeugen ab.

Zulässige Zustände für Produkttypen festlegen

Die Zustände, die Produkte im Verlauf des Entwicklungsprozesses annehmen können, wurden bereits definiert. Im Rahmen dieses Aktivitätstyps ist zu untersuchen, ob die definierten Zustände für alle Produkte ausreichend sind, und ob Produkte existieren, die einen anderen Zustandsautomaten nachweisen. Sollte das der Fall sein, so sind entsprechende Vorkehrungen in der Bausteinbibliothek zu treffen. Gegebenenfalls müssen weitere Informationen zu einem Produkt aufgenommen werden, wie z.B. die Sprache, in der ein Dokument verfaßt werden soll.

Produkttyp- und Zustandsabhängige Benutzerrechte vergeben

Nachdem die Produkttypen in der Bausteinbibliothek angelegt sind, werden für jede Benutzergruppe und jeden einzelnen Benutzer entsprechende Rechte auf die Produkttypen vergeben. Hierbei müssen wieder die möglichen Zustände, die ein Produkt im Verlauf des Entwicklungsprozesses annehmen kann, berücksichtigt werden. Der gleiche Benutzer kann nämlich abhängig von den Zuständen eines Produkts unterschiedliche Rechte haben. Zum Erzeugen von Versionen und Varianten müssen die entsprechenden Benutzer die erforderlichen Rechte erteilt bekommen.

Ergebnisse

Das Ergebnis dieses Schrittes ist eine Bausteinbibliothek, die nun auch Produkte aufnehmen kann.

4.1.4 Benutzer verwalten

Zweck

Im Verlauf des Projekts können sich die Rollen der Projektmitglieder ändern, es kommen neue Mitarbeiter hinzu oder Mitarbeiter scheiden aus. Dieser Aktivitätstyp sorgt dafür, daß jeder Projektmitarbeiter während des gesamten Projektverlaufs im Rahmen der ihm zugewiesenen Rechte Zugang zur Bausteinbibliothek hat.

Beteiligte

Ausführend: Projektbibliothekar
Mitwirkend: Projektleiter

Vorgehen

Dieser Aktivitätstyp besteht aus der wiederholten Durchführung nachfolgender Schritte, die schon weiter oben beschrieben wurden:

- Benutzergruppen und Benutzer pflegen,
- Zustandsunabhängige Benutzerrechte für Bausteine und Produkttypen pflegen
- Zustandsabhängige Benutzerrechte für Bausteine und Produkttypen pflegen.

Ergebnisse

Das Resultat dieses Aktivitätstyps ist eine dem aktuellen Projektstadium angepaßte Bausteinbibliothek.

4.2 Fortschreibung der Bausteinbibliothek

Dieser Aktivitätstyp setzt die Strukturierung der Bausteinbibliothek voraus und widmet sich den inhaltlichen Aspekten. In den Schritten

- Bausteine verwalten und
- Produkte verwalten

werden die kontinuierliche Fortschreibung der Bausteine und ihrer zugehörigen Produkte in der Bausteinbibliothek beschrieben.

4.2.1 Bausteine verwalten

Zweck

Dieser Aktivitätstyp begleitet alle Bausteine, die in der Bausteinbibliothek verwaltet werden. In der Bausteinbibliothek werden eine erste Version der Bausteine bzw. Behälter für Bausteine angelegt und inhaltlich schrittweise durch dazugehörige Produkte gefüllt. Neue Versionen und Varianten der Bausteine werden erzeugt, deren Entwicklung jederzeit zurückverfolgbar ist.

Beteiligte

Ausführend: Projektbibliothekar
Mitwirkend: Projektleiter

Vorgehen

Im Detail sind folgende Schritte zu durchlaufen:

- Erste Version eines Bausteins anlegen,
- Bausteine fortschreiben,
- Versionen und Varianten der Bausteine erzeugen.

Erste Version eines Bausteins anlegen

Im Rahmen des Aktivitätstyps „Bausteinbibliothek strukturieren“ wurden bereits für die einzelnen Entwicklungs-Bausteine entsprechende Behälter angelegt. Diese Behälter stehen in einer ersten Version zur Verfügung und können Produkte mit ihren Versionen aufnehmen. Dabei ist zu beachten, daß die Menge aller Produkte, die zu einem Baustein gehören in einem Behälter verwaltet werden müssen.

Bausteine fortschreiben

Produkte und ihre Versionen, die im Rahmen der Entwicklung eines Bausteins entstehen, werden dem entsprechenden Behälter in der Bausteinbibliothek zugeordnet. Diese Zuordnung wird von dem Projektbibliothekar überwacht und koordiniert.

Versionen und Varianten der Bausteine erzeugen

Wenn die Versionen der verschiedenen Produkte, die zu einem Baustein gehören, einen gemeinsamen Entwicklungsstand darstellen, so wird die Version des Bausteins „eingefroren“ und eine neue Version erzeugt. Das „Einfrieren“ von Bausteinen wird z.B. vorgenommen, um den Entwicklungsstand einer Beta-Version festzuhalten und vor weiteren Änderungen zu schützen. Das Erzeugen von Varianten wird benötigt, wenn z.B. Bausteine auf einer anderen Plattform portiert werden müssen.

Ergebnisse

Das Ergebnis besteht aus einer Bausteinbibliothek, die an den Projektfortschritt angepaßt ist.

4.2.2 Produkte verwalten

Zweck

Analog zur Verwaltung von Bausteinen begleitet dieser Aktivitätstyp den gesamten Lebenszyklus jedes Produkts. Produkte werden in einer ersten Version in der Bausteinbibliothek angelegt und den Projektmitgliedern zur Weiterentwicklung zur Verfügung gestellt. Es werden neue Versionen und Varianten der Produkte erzeugt, die jeder Zeit zurückverfolgt werden können, so daß auf einer früheren Produktversion wiederaufgesetzt werden kann.

Beteiligte

Ausführend: Projektbibliothekar
Mitwirkend: Projektleiter

Vorgehen

Zur Verwaltung von Produkten gehört:

- Das Einfügen einer Anfangsversion in die Bausteinbibliothek,
- Das Bereitstellen des Produkts für die Projektmitarbeiter zur Weiterentwicklung bzw. zum Wiederaufsetzen auf einer früheren Version.
- Das Aufnehmen des Produkts unter einer neuen Version, gegebenenfalls in einem veränderten Zustand.
- Das Aufnehmen von Varianten eines Produkts.

Die detaillierte Vorgehensweise hängt vom ausgewählten KM-Werkzeug ab und wird nicht weiter vertieft.

Ergebnisse

Das Ergebnis besteht wieder aus einer Bausteinbibliothek, die an den Projektfortschritt angepaßt ist.

Das Konfigurationsmanagement im EOS-Modell gliedert sich in die „Strukturierung“ und „Fortschreibung“ der Bausteinbibliothek. Bei der „Strukturierung“ wird die Systemarchitektur durch die Bausteinbibliothek abgebildet. Damit ist die Grundlage zur Verwaltung von Produkttypen geschaffen, die während der Entwicklung eines Bausteines entstehen. Ausgehend von den definierten Rollen werden Benutzergruppen abgeleitet. Es werden Benutzer mit entsprechenden Rechten auf die Bausteinbibliothek angelegt. Die Struktur der Bausteinbibliothek wird bei der „Fortschreibung“ mit Inhalten gefüllt. Das bedeutet, daß Produkte mit Versionen und Varianten in der Bausteinbibliothek aufgenommen und verwaltet werden.

5 Nutzung und Bewertung

Im Rahmen dieses Subprozesses wird der Nutzer von Anfang bis zum Ende des Projekts begleitet und betreut. Die Einbindung der Anwender im Entwicklungsprozeß hat viele Fassaden. Im Subprozeß „Software-Entwicklung“ werden z.B. die Anwender von der Anforderungsanalyse bis hin zur Abwicklung von Schulungen soweit sinnvoll einbezogen. Der Abschluß eines Bausteinzyklusses bietet einen Einstiegspunkt, um die Anwender stets in die Entwicklung einzubeziehen. Der Funktionsumfang des Systems wird erweitert, wenn z.B. ein Komponenten-Zyklus abgeschlossen ist. Diese neuen Funktionen können den Anwendern präsentiert werden und mit ihnen diskutiert werden. Ein weiterer Aspekt sind Rückmeldungen und Bewertungen der Anwender. Auf diesen Aspekt wollen wir im folgenden exemplarisch näher eingehen.

Nutzer können positive Bewertungen in Form von Lob oder negative in Form von Beschwerden äußern. Diese Rückmeldungen sollten systematisch erfaßt, bewertet und zur Verbesserung des Anwendungssystems bzw. des Entwicklungsprozesses genutzt werden. Positive Rückmeldungen bestätigen eine erfolgreiche Zusammenarbeit und weisen auf einen zufriedenen Auftraggeber und Nutzer hin. Der Umgang mit Beschwerden dagegen ist weitaus komplizierter und erfordert viel Fingerspitzengefühl. Denn Beschwerden weisen in der Regel auf Mängel hin und müssen zur Förderung der Kundenzufriedenheit sehr ernst genommen werden. Daher konzentrieren wir uns im Folgenden auf das Beschwerdemanagement und zeigen auf, wie Nutzer zur Abgabe von Beschwerden ermuntert werden, wie Beschwerden aufgenommen und bearbeitet werden, wie auf sie reagiert werden kann und wie aus Beschwerden durch Auswertungen Informationen gewonnen werden können (Abb. 1). Damit kommt die „Stimme der Nutzer“ hinsichtlich Verbesserungsnotwendigkeit zum Ausdruck und wird beim Entwicklungsprozeß berücksichtigt. Die vorgeschlagenen Methoden betonen zwar das Beschwerdemanagement, können aber für beliebige Arten von Bewertungen genutzt werden.

Bei der Beschreibung der in der Abbildung 1 dargestellten Aktivitätstypen orientieren wir uns an /Stauss, Seidel 1998/, ziehen aber /Barlow, Moeller 1996/, /Bellabarba, Radtke 1998/, /Preyer 1998/, /Seiwert 2001/ und /Stauss, Seidel 1998/ hinzu.

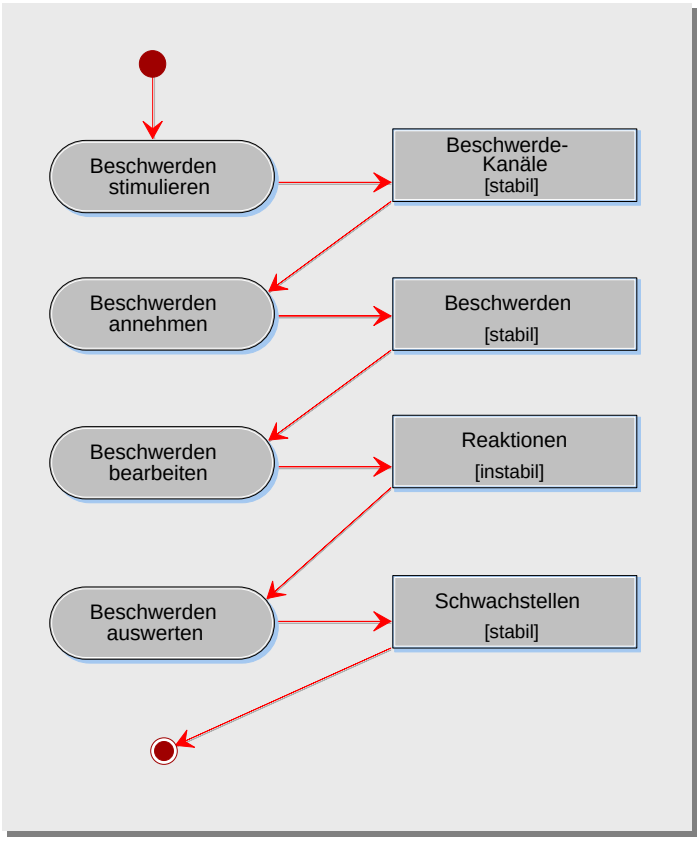


Abb. 1: Aktivitätstypen der Nutzung und Bewertung

5.1 Beschwerden stimulieren

Zweck

In der Regel versuchen die Unternehmen die Anzahl der Beschwerden zu minimieren. Eine Beschwerdeminimierung ist aber nur dann sinnvoll, wenn davon ausgegangen werden kann, daß alle unzufriedenen Nutzer sich beschweren. So wäre eine geringe Beschwerdezahl ein eindeutiger Indikator für die Zufriedenheit der Nutzer. Empirische Untersuchungen belegen aber, daß ein Großteil der unzufriedenen Nutzer sich *nicht* beschweren. Sie reden über die negativen Erfahrungen in ihrem Umfeld und wechseln oft zu einem anderen Anbieter. Sowohl eine negative Mundpropaganda als auch die Abwanderung von Kunden zu anderen Unternehmen verursacht enorme finanzielle Schäden und ist daher zu vermeiden. Beschwerdestimulierung aktiviert Nutzer dahingehend, daß sie jegliche Art von Unzufriedenheit zum Zeitpunkt des Problemauftritts in Beschwerden umsetzen und dem Auftragnehmer mitteilen.

Beteiligte

Ausführend: Anwenderbetreuer
Mitwirkend: Projektmanager

Vorgehen

Die Beschwerdestimulierung vollzieht sich in den folgenden zwei Schritten:

- Beschwerdewege einrichten,
- Beschwerdewege kommunizieren.

Beschwerdewege einrichten

Beschwerdewege sollen es dem Nutzer erleichtern, seine Unzufriedenheit zu äußern. Es wird zwischen mündlichen, schriftlichen (z.B. Briefe, Emails oder Formulare im Web) und telefonischen Beschwerdewegen unterschieden. Unternehmen sollten möglichst alle diese Wege unterstützen. Entscheidend ist auch das Reduzieren von Beschwerdebarrieren. Dazu muß eindeutig klargestellt werden, daß kritische Kundenäußerungen sehr erwünscht sind. Man sollte auch nicht passiv auf Beschwerden warten, sondern sich aktiv nach Problemen erkundigen. Dies kann z.B. durch Befragung von Nutzern erfolgen. Die nächste Abbildung zeigt ein Beispiel für ein Formular zur Befragung von Anwendern, die für das Anwendungssystem geschult wurden (Abb. 2). Solche Formulare können auch im Web präsentiert werden, so daß eine automatische Auswertung ermöglicht wird.

Beschwerdewege kommunizieren

Die eingerichteten Beschwerdewege müssen gegenüber dem Nutzer aktiv kommuniziert und aufgezeigt werden. Denn nur so ist sicherzustellen, daß die Beschwerdewege von Anwendern genutzt werden und daß das Beschwerdeaufkommen unzufriedener Nutzer erhöht wird. Die Angabe der Beschwerdewege können auf unterschiedliche Weise angegeben werden. Zum Beispiel ist die Werbung im Internet oder durch Broschüren ein gängiger Weg dazu. Kleine Geschenke können den Anreiz für den Nutzer erhöhen, die angebotenen Beschwerdewege zu nutzen. Die Einführung von beschwerdestimulierenden Maßnahmen sollte stufenweise erfolgen, damit keine „Lawine“ von Beschwerden losgetreten wird, die mit der vorhandenen Ressourcen nicht bearbeitet werden kann.

Ergebnisse

Als Resultat liegen eingerichtete Beschwerdewege vor, die aktiv „vermarktet“ werden.

Hinweise und Tipps

Beim organisatorischen Aufbau des Beschwerdemanagements wird zwischen einer zentralen und dezentralen Organisation unterschieden. Bei der ersten Alternative werden sämtliche Aufgaben allein von einer zentralen Beschwerdeeinheit wahrgenommen. Dagegen werden bei einem dezentralen Beschwerdemanagement die Beschwerdefälle von den dezentralen Einheiten eigenständig abgewickelt. Häufig empfiehlt es sich aber, ein duales Beschwerdemanagement aufzubauen. So können wesentliche Aufgaben des Beschwerdemanagements, wie z.B. Überwachung von Prinzipien, zentralisiert werden und die dezentralen Einheiten autonom auf Beschwerden reagieren.

Schulungsthema:	.....	
Ort / Datum:	.....	
Name des Referenten:	1.....	2.....
Welches Ziel wollten Sie mit dem Besuch dieses Seminars erreichen?		
	☹	😐 ☺
• Ich habe mein Ziel erreicht	☐ ☐	☐ ☐ ☐
• Die Themenauswahl entsprach meiner Erwartung	☐ ☐	☐ ☐ ☐
Welche Themen sollten zusätzlich bzw. intensiver behandelt werden?		
Welche Themen sollten verkürzt bzw. gar nicht behandelt werden?		
• Die Aufbereitung war verständlich	☐ ☐	☐ ☐ ☐
• Menge, Gliederung und Zeitmanagement waren	☐ ☐	☐ ☐ ☐
• Die Fachkompetenz der Referenten war 1. Referent 2. Referent	☐ ☐ ☐ ☐	☐ ☐ ☐ ☐ ☐ ☐
• Die eingesetzten Methoden waren abwechslungsreich Es sollte stärker eingesetzt werden: ☐ Vortrag ☐ Diskussion ☐ Erfahrungsaustausch ☐ Übungen ☐ Rollenspiel ☐ Gruppenarbeit	☐ ☐	☐ ☐ ☐
• Die Teilnehmer wurden einbezogen und aktiviert	☐ ☐	☐ ☐ ☐
• Den Einsatz von Medien und Materialien fand ich	☐ ☐	☐ ☐ ☐
• Die Schulungsunterlagen sind Ansprechend Aktuell Informativ	☐ ☐ ☐ ☐ ☐ ☐	☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐
• Atmosphäre und Organisation erlebte ich	☐ ☐	☐ ☐ ☐
• Den Nutzen beurteile ich für mich persönlich	☐ ☐	☐ ☐ ☐
• Mein Gesamturteil: Die Veranstaltung war	☐ ☐	☐ ☐ ☐
Sonstige Bemerkungen:		

Abb. 2: Beispiel für ein Formular zur Kundenbeurteilung einer Anwenderschulung

5.2 Beschwerden annehmen

Zweck

Das Ziel dieses Aktivitätstyps ist es, den Erstkontakt mit dem sich beschwerenden Nutzer vorzubereiten und alle relevanten Informationen im Hinblick auf eine schnelle und unkomplizierte Bearbeitung des Beschwerdefalls zu erfassen.

Beteiligte

Ausführend: Anwenderbetreuer
Mitwirkend: Projektmanager

Vorgehen

Im Detail sind folgende Schritte zu durchlaufen:

- Erstkontakt vorbereiten,
- Beschwerdeinformationen erfassen.

Erstkontakt vorbereiten

Nutzer haben in der Regel Verständnis dafür, daß Probleme auftreten können. Doch dieses Verständnis kann schnell in Empörung umschlagen, wenn das Unternehmen schon beim Erstkontakt mit dem sich beschwerenden Nutzer keinerlei Bereitschaft zeigt, das Problem zu beseitigen oder den Schaden wieder gut zumachen. Für eine erfolgreiche Beschwerdeabwicklung ist aber die Gestaltung des Erstkontakts mit dem unzufriedenen Nutzer entscheidend. Zu diesem Zweck müssen klare Verhaltensrichtlinien erstellt werden. Die Mitarbeiter müssen trainiert werden, damit sie durch eine angemessene Reaktion für eine Beruhigung der Situation, eine sachliche Klärung des Falles und die Einleitung einer Problemlösung sorgen können. Abbildung 3 faßt bewährte Verhaltensrichtlinien zusammen.

Bei der Annahme von Beschwerden sollten dem Mitarbeiter Mittel zur Verfügung stehen, mit denen er überprüfen kann, ob der Nutzer sich bereits beschwert hat. Sollte das der Fall sein, so müssen ihm Informationen zur Beschwerdehistorie bereitgestellt werden, damit der Mitarbeiter individuell auf den Nutzer eingehen kann.

1. Verstehen Sie Beschwerden als einen normalen Teil ihrer Arbeit und als Chance, Kundenunzufriedenheit abzubauen und Kundenbindung zu sichern.
2. Suchen Sie einen ruhigen Ort für das Beschwerdegespräch. Lassen Sie vor allem nicht andere Kunden mithören. Bieten Sie dem Kunden eine Sitzgelegenheit an. Sprechen Sie den Kunden mit Namen an.
3. Signalisieren Sie Gesprächsbereitschaft („Lassen Sie uns in Ruhe darüber reden"). Drücken Sie mit Mimik, Augenkontakt und Körpersprachen ihre Zuwendung aus. Sprechen Sie eine Entschuldigung aus oder zumindest das Bedauern, daß der Kunde dieses negative Erlebnis hatte. Wählen Sie bei entsprechenden Formulierungen die Ich-Form („Es tut mir leid, daß Sie diese Unannehmlichkeiten hatten").
4. Hören Sie gut zu. Unterbrechen Sie den Beschwerdeführer nicht. Lassen Sie ihn zunächst auch dann ohne Unterbrechung ausreden, wenn er Unzutreffendes vorbringt.
5. Wählen Sie eine ruhige und höfliche Gesprächsart. Reagieren Sie gelassen auf Übertreibungen und persönliche Schuldvorwürfe. Weisen Sie Beschimpfungen ruhig zurück und leiten Sie das Gespräch auf den sachlichen Kern. Mit einem Kunden streitet man sich nicht oder trägt auch keinen Machtkampf aus.
6. Stellen Sie inhaltliche Fragen solange, bis die Situation eindeutig geklärt ist. Wählen Sie dabei höfliche Frageformen („Danke für den Hinweis, ich möchte nur noch wissen ...").
7. Versetzen Sie sich in die Lage des Kunden („Ich kann mir gut vorstellen, daß Sie verärgert sind"). Vermeiden Sie Formulierungen, die den Ärger vergrößern („Das sehen Sie völlig falsch" oder „Das ist doch Ihre Schuld!").
8. Machen Sie sich Notizen. Der Schreibvorgang beweist dem Kunden, daß Sie die Beschwerde ernst nehmen und veranlaßt ihn zu einer sorgfältigeren Erläuterung des Sachverhalts. Außerdem sind die Notizen wertvoll für die Beschwerdebearbeitung und -auswertung.
9. Vermeiden Sie Sofortdiagnosen und nehmen Sie alle Informationen entgegen, ohne gleich ein Schuldeingeständnis abzugeben.
10. Ist tatsächlich ein Fehler passiert, geben Sie nicht einem Kollegen, anderen Abteilungen oder dem Unternehmen generell die Schuld („...das passiert denen ständig" oder „Das kriegen die nie in den Griff!").
11. Leiten Sie sofort die Bearbeitung der Beschwerde ein. Bieten Sie eine faire Lösung an.
12. Erkundigen Sie sich, ob der Kunde mit der Regulierung einverstanden ist.
13. Ist eine unverzügliche Problemlösung nicht möglich, sagen Sie dem Kunden eine genaue Prüfung zu und geben Sie an, wie lange es dauern wird, bis er eine Nachricht erhält. Halten Sie den angegebenen Zeitpunkt unbedingt ein. Sollte dies trotz aller Bemühungen nicht möglich sein, informieren Sie den Beschwerdeführer rechtzeitig darüber und erläutern Sie ihm die Gründe.
14. Sind Sie nicht zuständig bzw. können Sie nichts tun, leiten Sie die Beschwerde eigenhändig weiter, und sorgen Sie dafür, daß der Annahme- und Bearbeitungsprozeß kundenorientiert fortgesetzt wird.
15. Beenden Sie das Gespräch mit einer positiven Formulierung („Ich freue mich, daß wir Sie auf diese Weise zufrieden stellen können").

Abb. 3: Bewährte Verhaltensrichtlinien /Stauss, Seidel, 1998/

Beschwerdeinformationen erfassen

1. Beschwerdeinhalts-Informationen	

1.1 Informationen über das Beschwerdeproblem	• Art des Problems
	• Genaue Umstände des Beschwerdevorfalls:
	  ▪ Ort
	  ▪ Zeitpunkt
	  ▪ Fallschilderung
	• Erste- oder Folgebeschwerde
	• Implikationen für die unternehmerische Reaktion:
	  ▪ Vom Kunden gewünschte Fall-Lösung
	  ▪ Gewährleistungs- oder Kulanzfall
	  ▪ Reaktionsdringlichkeit/Bearbeitungspriorität

1.2 Informationen über den Beschwerdeführer	• Interner oder externer Kunde • Stammdaten des Beschwerdeführers • Bezug des Beschwerdeführers zum Beschwerdevorfall • Ausmaß der Verärgerung und Handlungsabsicht des Beschwerdeführers
1.3 Informationen über das Beschwerdeobjekt	• Produkte und/oder Dienstleistungen • Andere Aspekte des Marktangebots • Gesellschaftspolitisches Verhalten
2. Beschwerdebearbeitungs-Informationen	
2.1 Informationen über die Beschwerdeannahme	• Zeitpunkt der Entgegennahme • Beschwerdeweg • Adressat der Beschwerde • Entgegennehmender Mitarbeiter
2.2 Informationen über die Beschwerdebearbeitung	• Beschwerdeverantwortliche • Beschwerdebearbeitungsprozeß
2.3 Informationen über die Beschwerdelösung	• Dem Kunden gegenüber gemachte Zusagen • Tatsächlich realisierte Problemlösung

Abb. 4: Checkliste zur Erfassung von Beschwerden

Beschwerden lösen in der Regel Änderungen beim Anwendungssystem aus. Wie solche Änderungen erfaßt und gemanagt werden, haben wir in den Kapiteln über die Software-Entwicklung und das Projektmanagement ausführlich beschrieben, so daß wir hier nicht weiter darauf eingehen werden. Die Abbildung 4 zeigt eine Checkliste, die beim Erfassen von Beschwerden hilfreich sein kann. Sie kann auch als Gliederungsgrundlage für die Dokumentation von Beschwerden genutzt werden.

Ergebnisse

Das Ergebnis ist eine Dokumentation der eingehenden Beschwerden und Verhaltens-richtlinien für den Umgang mit unzufriedenen Nutzern.

5.3 Beschwerden bearbeiten

Zweck

Nachdem eine Beschwerde eingegangen ist, müssen Maßnahmen unternommen werden, um sie zu behandeln. Dazu muß die Relevanz der Beschwerde geprüft werden, eine Lösung

entwickelt werden und die Verantwortlichkeiten festgelegt werden. Diese gilt es im Rahmen dieses Aktivitätstyps durchzuführen.

Beteiligte

Ausführend: Anwenderbetreuer
Mitwirkend: Projektmanager

Vorgehen

Bei der Bearbeitung von Beschwerden wird zwischen *internen* und *externen* Arbeitsschritten unterschieden. Zu den internen Schritten gehören alle Maßnahmen, die bei der Bearbeitung von Beschwerden nötig sind, aber von dem Nutzer nicht wahrgenommen werden. Dazu gehört z.B. die Entscheidung über die Zuständigkeiten bei der Bearbeitung, die Entwicklung von Lösungswegen, Termin- und Ressourcenplanung. Diese Schritte haben wir im Rahmen des Änderungsmanagements in den Kapiteln über die Software-Entwicklung und das Projektmanagement diskutiert. Dort wurden ebenfalls entsprechende Beschreibungsschemata für die Dokumentation vorgeschlagen.

Externe Schritte umfassen alles, was der Nutzer bei der Bearbeitung von Beschwerden wahrnimmt. Dazu gehört die *Eingangsbestätigung* der Beschwerde, die Benachrichtigung über den Bearbeitungsstatus in *Zwischenbescheiden* und die Mitteilung der Problemlösung im *Endbescheid.*

Bei Beschwerden, die nicht sofort gelöst werden können, ist es sinnvoll, dem Nutzer den Eingang seiner Beschwerde zu bestätigen. Der Eingangsbestätigung sollte folgende Informationen enthalten:

- den Zeitpunkt des Eingangs der Beschwerde,
- eine Zusammenfassung des Problems,
- ein Bedauern über die vom Nutzer erlebten Unannehmlichkeiten,
- die eingeleiteten Maßnahmen,
- den voraussichtlichen Erledigungstermin bzw. -zeitraum.

Zwischenbescheide sind erforderlich, wenn dem Nutzer mit der Eingangsbestätigung kein Erledigungstermin bzw. -zeitraum genannt werden konnte, oder wenn die vereinbarten Zwischentermine nicht eingehalten werden konnten.

Im Endbescheid wird die entwickelte Lösung vorgestellt und dem Nutzer mitgeteilt, daß der Beschwerdefall abgeschlossen ist. Folgende Punkte sollten im Endbescheid enthalten sein:

- eine kurze Zusammenfassung des Problems,
- Lösungsvorschläge und mögliche Begründung für den ausgewählten Lösungsweg,
- das Bedauern über die für den Nutzer entstandenen Unannehmlichkeiten.

Generell sollten alle Interaktionen mit dem Nutzer festgehalten werden, damit eine Beschwerde-Historie zur Verfügung steht.

Ergebnisse

Die im Rahmen einer Beschwerde genannten Probleme werden gelöst und vollständig dokumentiert.

5.4 Beschwerden auswerten

Zweck

Im Mittelpunkt dieses Aktivitätstyps steht die aktive Nutzung der erfaßten Beschwerdeinformationen für Verbesserungsmaßnahmen, um künftige Kundenprobleme zu vermeiden und Nutzer durch Kundenzufriedenheit an sich zu binden. Da in den Beschwerden die „Stimme der Nutzer“ hinsichtlich Verbesserungsnotwendigkeit zum Ausdruck kommt, bietet die Beschwerdeauswertung eine Chance, um vom Kunden wahrgenommene Probleme systematisch und konsequent zu eliminieren. Die Beschwerdeauswertung bietet mit Hilfe entsprechender Analysetechniken Informationen, um von der Problemdiagnose zur wirksamen Problemprävention gelangen zu können.

Beteiligte

Ausführend: Anwenderbetreuer
Mitwirkend: Projektmanager

Vorgehen

Grundsätzlich wird zwischen der quantitativen und qualitativen Beschwerdeauswertung unterschieden, so daß das Vorgehen sich zweistufig gliedern läßt:

- Beschwerden quantitativ auswerten,
- Beschwerden qualitativ auswerten.

Beschwerden quantitativ auswerten

Im Rahmen der quantitativen Beschwerdeauswertung wird das gesamte Beschwerdeaufkommen mengenmäßig im Hinblick auf wichtige Merkmale analysiert. Dabei wird zwischen *univariaten* und *bivariaten* Verfahren unterschieden.

Univariate Verfahren beziehen sich auf jeweils eine Variable, die näher untersucht werden soll. Entsprechende statistische Methoden wären z.B. die Berechnung von Mittelwerten, von absoluten und relativen Häufigkeiten oder Streuungsmaßen. Die Ergebnisse solcher Berechnungen werden mit Hilfe von Histogrammen und Pareto-Diagrammen dargestellt.

Bivariate Verfahre untersuchen den Zusammenhang zwischen mindestens zwei Variablen. Häufig genutzte statistische Methoden hierzu sind Kreuztabellen, Regression oder Frequenz-Relavanz-Analyse von Beschwerden (FRAP).

Beschwerden qualitative auswerten

In einem zweiten Schritt müssen die Ergebnisse der quantitativen Beschwerdeanalyse interpretiert werden. Zunächst wird eine systematische *Ursache-Wirkungs-Analyse* vorgenommen, die im Rahmen der *Beschwerde-Problem-Deployment* genutzt wird, um die identifizierten Ursachen zu beseitigen.

Die Ursache-Wirkungs-Analyse ist ein Verfahren des *Total-Quality-Managements*. Bei diesem Verfahren werden für ein definiertes Problem (= Wirkung) mögliche Einflußgrößen (= Ursachen) ermittelt. Die Einflußgrößen sowie ihre Beziehungen werden in Ursache-Wirkungs-Diagrammen strukturiert beschrieben. Die nächste Abbildung zeigt ein Beispiel für solch ein Diagramm.

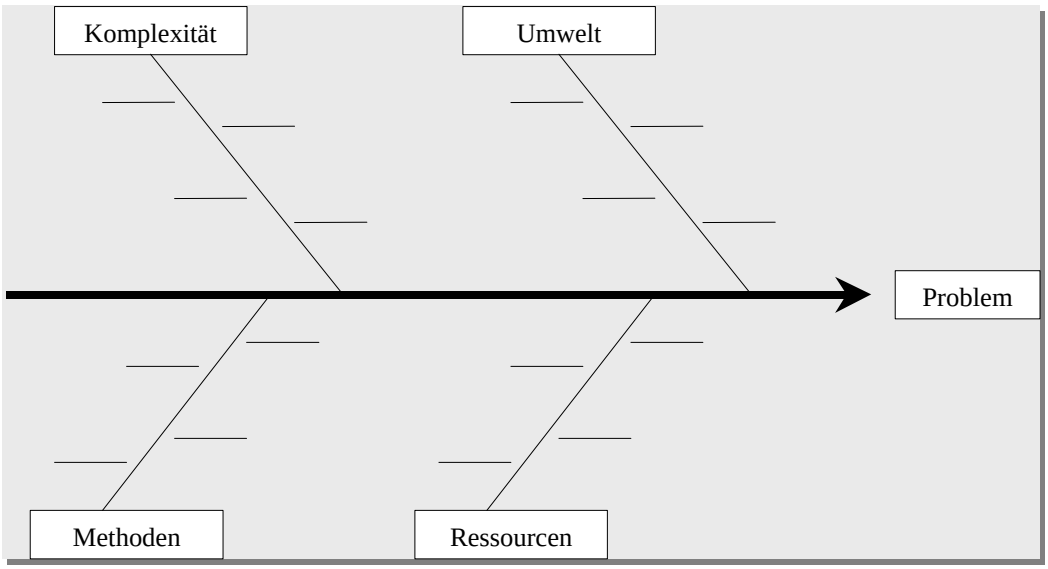


Abb. 6: Beispiel für ein Ursache-Wirkungs-Diagramm

Die identifizierten Hauptursachen „Komplexität“, „Umwelt“, „Methoden“ und „Ressourcen“, die zu dem erkannten „Problem“ führen, werden detailliert beschrieben (die Querstriche können hierfür genutzt werden) und zur Entwicklung einer Lösung herangezogen. Die Konstruktion von Ursache-Wirkungs-Diagrammen vollziehen sich in folgenden Schritten:

1. Das Problem wird formuliert.
2. Die Hauptursachen werden identifiziert.
3. Die Hauptursachen werden durch detaillierte Ursachen beschrieben.
4. Die detaillierten Ursachen werden mit dem Ziel der Ermittlung ihrer Wahrscheinlichkeit analysiert.
5. Die wahrscheinlichsten Ursachen werden geprüft.
6. Aus den identifizierten Ursachen werden erste Lösungsansätze abgeleitet.

Bei der Entwicklung von Ursache-Wirkungs-Diagrammen werden im letzten Schritt erste Lösungsansätze vorgeschlagen. Damit wird das Beschwerde-Problem-Deployment angestoßen. Hierbei handelt es sich um eine Methode, die die Erwartungen der Nutzer systematisch in messbare Produkt- und Prozeßparameter übersetzt. Die Methode lehnt sich an das Quality-Function-Deployment (QFD) an, das im Rahmen von Total-Quality-Management vorgeschlagen wurde. Das Verfahren Beschwerde-Problem-Deployment läßt sich in fünf Schritten gliedern:

1. Die identifizierten Probleme überprüfen und bewerten.
2. Globale Problemlösungen entwickeln und bewerten.
3. Aus den globalen Problemlösungen eine konkrete Lösung spezifizieren.
4. Aktivitäten zur Durchführung der spezifizierten Lösung definieren.
5. Ressourcen zur Implementierung und Überwachung der definierten Aktivitäten festlegen.

Analog zur Vorgehensweise des QFD werden die Schritte mit Hilfe von verschiedenen Matrix-Darstellungen durchgeführt.

Ergebnisse

Das Resultat sind identifizierte Ursachen, die mit Hilfe von Ursache-Wirkungs-Diagrammen beschrieben werden, und dokumentierte Lösungswege.

Hinweise und Tipps

Um Ressourcen zu sparen, sollte die quantitative Beschwerdeanalyse rechnergestützt erfolgen.

In diesem Kapitel haben wir den Subprozeß „Nutzung und Bewertung“ beschrieben. Der Kern dieses Subprozesses ist, die „Stimme der Nutzer“ hinsichtlich Verbesserungen zu berücksichtigen. Dazu wurde das Beschwerdemanagement eingeführt, das die Stimulierung, Annahme, Bearbeitung und Auswertung von Beschwerden systematisch unterstützt.

Anhang B: Inhalt der CD-ROM

Auf der CD-ROM[1] zur Arbeit befinden sich folgende Quellcode-Dateien:

I. Prozeßmanagement-System (PMS)

[Englische und deutsche Version]

PMS\Pms.jpx	Projekt-Datei für JBuilder 5.0
PMS\Scr	Quellcode der Java-Dateien
PMS\Libs	Benötigte Bibliotheken
PMS\Docs	Konfigurationsdatei und Beschreibungsschemata
PMS\Systems	Beispiel-Systeme
PMS\JRE	Java Runtime Environment zum Starten von PMS
PMS\Documentation	HTML-Dokumentation der PMS-Klassen

II. Aktivitätsdiagramme (Prozeßstruktur in PMS)

ArgoUMLProjekt\ArgoProjekt.jpx	Projekt-Datei für JBuilder 5.0
ArgoUMLProjekt\Org, Src_new	Quellcode der modifizierten Java-Dateien des Open-Source-Werkzeugs ArgoUML /www.argouml.org/
ArgoUMLProjekt\Lib	Benötigte Bibliotheken
ArgoUMLProjekt\Documentation	HTML-Dokumentation der ArgoUML-Klassen

III. Aufwandsschätzung für EOS-Projekte (CEOS)

CEOS\Ceos.exe	objectiF-AddIn zur Aufwandsschätzung
CEOS\Ceos.vbp	Projektdatei für Visual Basic 6.0
CEOS*.*	Quellcode der Visual Basic-Dateien
CEOS\Lib	Benötigte Java-Bibliotheken und ihre Quellcodes
CEOS\Jre	Java-Runtime-Umgebung zur Ausführung der Java-Komponenten von CEOS
CEOS\Help	Eingebundene Hilfsdateien
CEOS\TreeUpdateScript	Aktionsskript zur Aktualisierung des Systembaums in objectiF (Visualbasic Projekt)

[1] Die CD-ROM kann von „sarferaz@informatik.uni-marburg.de“ angefordert werden.

Installationsanweisung für den Protoypen

I. Prozeßmanagement-System (PMS)

1. Kopieren Sie das Verzeichnis PMS_DE (deutsche Version) oder PMS_EN (englische Version) auf Ihre Festplatte.
2. Stellen Sie die Systempfade in der Datei PMS_DE\Docs\config.xml bzw. PMS_EN\Docs\config.xml ein.
3. Zum Starten des Prototyps führen Sie die Datei run_pms.bat aus.

II. Aufwandsschätzung für EOS-Projekte (CEOS)

1. Kopieren Sie das Verzeichnis CEOS auf Ihre Festplatte.
2. Registrieren Sie wie folgt den AddIn Ceos.exe und den Aktionsskript TreeUpdate_ASS.dll auf Ihrem System:
   - Alternative A:
     Führen Sie folgende zwei Befehle aus (Windows: „Start → Ausführen“)
     regsvr32.exe CEOS\Ceos.exe
     regsvr32.exe CEOS\TreeUpdateScript\TreeUpdate_ASS.dll
   - Alternative B (empfohlen):
     Kompilieren Sie die Projektdateien CEOS\Ceos.vbp und CEOS\TreeUpdateScript\TreeUpdate_ASS.vbp mit Visual Baisc 6.0.
3. Registrieren Sie wie folgt den AddIn Ceos.exe in objectiF (siehe Dokumentation von objectiF):
   a. Starten Sie objectiF und legen Sie den gewünschten Projekt an.
   b. Wählen Sie im objecitF-Menü „Add-Ins → Add-In Manager“ aus und fügen Sie CEOS\Ceos.exe hinzu.
4. Registrieren Sie wie folgt den Aktionsskript TreeUpdate_ASS.dll in objectiF (siehe Dokumentation von objectiF):
   a. Starten Sie objecitF und legen Sie den gewünschten Projekt an.
   b. Wählen Sie im objectiF-Menü „Modell → System“ aus.
   c. Wählen Sie im Kontextmenü des Systemdialogs den Menüpunkt „Skripte“ aus.
   d. Wählen Sie den Reiter „Skript-Server zuweisen“ aus.
   e. Wählen Sie im Kontextmenü des „global“ Knotens den Menüpunkt „Aktions-Skript-Server zuweisen“ aus und fügen Sie CEOS\TreeUpdateScript\TreeUpdate_ASS.dll hinzu.
5. Starten Sie den Prototypen aus I. und wählen Sie den Menüpunkt „Aktion → Aufwand schätzen“ zum Ausführen der Aufwandsschätzung.

Anhang C: Abkürzungsverzeichnis

AC_m^{App}	Attribute Application Complexity
AC_m^{Def}	Attribute Definition Complexity
API	Application Programming Interface
AS	Application Server
CC_m^{App}	Class Application Complexity
CC_m^{Def}	Class Definition Complexity
CCA_m^{App}	Class Application Complexity during Analysis activity
CCA_m^{Def}	Class Definition Complexity during Analysis activity
CCD_m^{App}	Class Application Complexity during Design activity
CCD_m^{Def}	Class Definition Complexity during Design activity
CCI_m^{App}	Class Application Complexity during Implementation activity
CCI_m^{Def}	Class Definition Complexity during Implementation activity
CU_m	Class Usability
CEOS	Cost estimation for EOS projects
COM	Component Object Model
EIS	Enterprise Information System
EJB	Enterprise JavaBeans
EOS	Evolutionäre, Objektorientierte Software-Entwicklung
FRAP	Frequenz-Relavanz-Analyse-Prinzip
IC_m	Inherited Class Complexity (analysis)
ICC_m	Inherited Class Complexity (design)
$InCC_m$	Inherited Class Complexity (implementation)
J2EE	Java 2 Platform Enterprise Edition
J2ME	Java 2 Platform Micro Edition
JAXP	Java API for XML Processing
JDBC	Java Database Connectivity
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JSP	JavaServer Pages
KM	Konfigurationsmanagement
LC_m	Local Class Complexity (analysis)
LCC_m	Local Class Complexity (design)
LGS	Lineare Gleichungssystem
LMS	Least Median of Squares
$LoCC_m$	Local Class Complexity (implementation)
MC_m^{App}	Method Application Complexity
MC_m^{Def}	Method Definition Complexity
MBC_m^{App}	Method Body Complexity (Application)
MBC_m^{Def}	Method Body Complexity (Definition)
MIC_m^{App}	Method Interface Complexity (Application)
MIC_m^{Def}	Method Interface Complexity (Definition)
NB	Nutzung und Bewertung
NDP	Number of Direct Parents
NIC	Number of Inherited Classes
NLA	Number of Local Attributes
NLM	Number of Local Methods

NMLA	Number of Method Local Attributes
NPuCA	Number of Public Class Attributes
NPuCM	Number of Public Class Methods
NSX	Number of System Components
NXC	Number of Component Classes
OMT	Object Modeling Technique
OOAD	Object-Oriented Analysis and Design
OOSE	Object-Oriented Software Engineering
O-R-Mapping	Object-Relation-Mapping
PM	Projektmanagement
PMS	Prozeßmanagement-System
QFD	Quality-Function-Deployment
QS	Qualitätssicherung
RMI	Remote Method Invocation
RUP	Rational Unified Process
SCA_m	System Complexity during Analysis phase
SCA_r	System Complexity coarse Analysis
SCD_m	System Complexity during Design phase
SCI_m	System Complexity during Implementation phase
SWE	Software-Entwicklung
UML	Unified Modeling Language
UP	Unified Software Development Process
WAP	Wireless Application Protocol
XC_m	Component Complexity
XCA_m	Component Complexity during Analysis activity
XCA_r	Component Complexity coarse Analysis
XCD_m	Component Complexity during Design activity
XCI_m	Component Complexity during Implementation activity
XML	Extensibl Markup Language
XU_m	Component Usability

Anhang D: Literaturverzeichnis

/Albrecht 1979/
A. J. Albrecht: *Measuring Applications Development Productivity*, in: Proceedings of IBM Applications Devision Joint SHARE/GUIDE Symposium, S. 83 – S. 92, Monterey, CA, 1979.

/Apt, Olderog 1994/
K. R. Apt, E. R. Olderog: *Programmverifikation – Sequentielle, parallele und verteilte Programme*, Springer-Verlag, Berlin, 1992.

/ArgoUML 0.9.3/
Open-Source Software für UML: www.arghouml.tigris.org.

/Baca 1997/
L. S. Baca, J. F. Bouchard, E. W. Collins, M. Eisenhour, D. D. Neidigk, M. J. Shortencarier, P. A. Trellue: *Software Development Methodology for High Consequence Systems*, Sandia National Laboratories, 1997.

/Balzert 1995/
H. Balzert: *Methoden der objektorientierten Systemanalyse*, BI Wissenschaftsverlag, 1995.

/Balzert 1996/
H. Balzert: Lehrbuch der Software-Technik, Software-Entwicklung, Spektrum Verlag, 1 996.

/Balzert 1998/
H. Balzert: Lehrbuch der Software-Technik, Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Spektrum Verlag, 1998.

/Bandi 2003/
R. K. Bandi, V. K. Vaishnavi, D. E. Turk: *Predicting maintenance performance using object-oriented design complexity metrics*, IEEE Transactions on Software Engineering 2003, Page(s): 77- 87, 2003.

/Barlow, Moeller 1996/
J. Barlow, C. Moeller: Eine Beschwerde ist ein Geschenk. Der Kunde als Consultant., Wien 1996.

/Basili, Rombach 1987/
V. R. Basili, H. D. Rombach: *Quantitative Software-Qualitätssicherung*, in: Informatik-Spekturm, S. 145 – 158, Oktober, 1987.

/Beck 2000/
K. Beck: *Extreme Programming Explained Embrace Change*, Addison Wesley Longman, 2000.

/Bellabarba, Radtke 1998/
A. Bellabarba, P. Radtke, D. Wilmes: Management von Kundenbeziehungen: 7 Bausteine für ein effizientes Kundenmanagement, Car Hanser Verlag, München, 1998.

/Benington 1956/
H. D. Benington: *Production of Large Computer Programs*, in: Proc. ONR Symposium on Advanced Programming Methods for Digital Computers, Juni 1956.

/Berard 1993/
E. V. Berard: *Essays on Object-Oriented Software Engineering, Volume 1*, Prentice Hall PTR, Englewood, New Jersey, 1993.

/Berztiss 1996/
A. T. Berztiss: *The software process and application domains*, Software Process Workshop, 1996, Proceedings of the 10th International, Page(s): 40-42, 1996.

/Beyer, 2001/
M. Beyer: Verwendung von UML zur Modellierung von Software-Entwicklungsprozessen, Diplomarbeit, Philipps-Universität Marburg, 2001.

/Binder 1999/
R. V. Binder: *Testing Object-Oriented Systems*, Addison-Wesley, Object Technology Series, 1999.

/Boehm 1981/
B. W. Boehm: *Software Engineering Economics*, Englewood Cliffs, Prentice Hall, 1981.

/Boehm 1982/
B. W. Boehm, J. F. Elwell, A. B. Pyster, E. D. Stuckle, R. D. Williams: *The TRW Software Productivity System*, Proceedings, 6th International Conference on Software Engineering, ACM/IEEE, September 1982.

/Boehm 1984/
B. W. Boehm: *Verifying and Validating Software Requirements and Design Specification*, in: IEEE Software, Januar 1984.

/Boehm 1986/
B. Boehm: *A Spiral Model of Software Development and Enhacement*, in: ACM SIGSOFT, August 1986.

/Boehm 1988/
B. W. Boehm: *A Spiral Model of Software Development and Enhancement*, in: IEEE Computer, Mai 1988.

/Boehm 1989a/
B. W. Boehm, R. Ross: *Theory W Software Project Management: Principles and Examples*, IEEE Transactions on Software Engineering, July 1989.

/Boehm 1989b/
B. W. Boehm: *Tutorial: Software Risk Management*, IEEE Computer Society Press, 1989.

/Booch 1991/
G. Booch: *Object-Oriented Design with Applications*, Benjamin/Cummings Publ. Comp., 1991.

/Booch 1994/
G. Booch: Object-Oriented Analysis and Design with Applications 2nd Edition, Benjamin/Cummings Publ. Comp., 1994.

/Bosch 1997/
K. Bosch: *Elementare Einführung in die angewandte Statistik*, Vieweg Verlag, 1997.

/Bourdeau 1995/
R. H. Bourdeau, B. H. C. Cheng: *A Formal Semantics for Object Model Diagrams*, IEEE Transactions on Software Engineering, 21(10), October 1995.

/Breslin 1998/
J. Breslin: *The Business Knowledge Repository*, Quorum Books, 1998.

/Broy 2001/
M. Broy: *Toward a mathematical foundation of software engineering methods*, IEEE Transactions on Software Engineering 2001, Page(s): 42-57, 2001.

/Budde 1992/
R. Budde, K. Kautz, K. Kuhlenkamp, H. Züllighoven: *Prototyping: An Approach to Evolutionary System Development*, Springer, 1992

/Buschmann 1998/
F. Buchschmann, R. Meunier, H. Rohner, P. Sommerlad, M. Stal – Siemens AG, Deutschland: *Pattern-orientierte Software-Architektur, Ein Pattern-System*, Addison-Wesley Longman Verlag GmbH, 1998.

/Cangussu 2002/
J. W. Cangussu, R. A. DeCarlo, A. P. Mathur: *A formal model of the software test process*, IEEE Transactions on Software Engineering 2002, Page(s): 782- 796, 2002.

/Calio 2000/
A. Calio, M. Antiero, G. Bux.: *Software process improvement by object technology (ESSI PIE 27785-SPOT)*, Proceedings of the 2000 International Conference on Software Engineering, Page(s): 641-647, 2000.

/Callahan 1996/
J. Callahan, F. Schneider, S. Easterbrook: *Automated Software Testing Using Model-Checking*; NASA Software Research Laboratory, Proceedings 1996 SPIN Workshop, 1996

/Cantor 1998/
M.R. Cantor: *Object-Oriented Project Management with UML*, Addison-Wesley, New York, 1998.

/Chevalley 2001/
P. Chevalley: *Applying mutation analysis for object-oriented programs using a reflective approach*, Software Engineering Conference, (APSEC 2001) Proceedings. Eighth Asia-Pacific, Page(s): 267-270, 2001.

/Chidamber, Kemerer 1994/
S. R. Chidamber, C. F. Kemerer: *A Metrics Suite for Object-Oriented Desing*, in: IEEE Transactions on Software-Engineering, June 1994.

/Cline, Lomow 1995/
Cline, Lomow: *C++ FAQS, Frequently Asked Questions*, Addison Wesley Publishing Company, 1995.

/Coad 1997/
P. Coad, D. North, M. Mayfield: *Object Models Strategies, Patterns and Applications, Second Edition*, Yourdon Press Prentice Hall Building Englewoad Cliffs, New Jersey, 1997.

/Coad, Yourdon 1990/
P. Coad, E. Yourdon: *Object-oriented analysis. 2nd Edition*, Yourdon Press, 1990.

/Coad, Yourdon 1991/
P. Coad, E. Yourdon: *Object-oriented design*, Yourdon Press, 1991.

/Conradi 2000/
R. Conradi: *Software Process Technology, 7th European Workshop*, Lecture Notes in Computer Science, Vol. 1780, Springer, 2000.

/Cunningham 1997/
D. Cunningham: *User-Centered Evolutionary Software Development Using Python and Java*, Proceedings of the 6th International Python Conference, San Jose, 1997.

/Deursen 1999/
A. van Deursen, T. Kuipers: *Identifying objects using cluster and concept analysis*, Proceedings of the 1999 International Conference on Software Engineering, Page(s): 246-255,1999.

/Dröschel 1998/
W. Dröschel et. al. : Inkrementelle und objektorientierte Vorgehensweise mit dem V-Modell ’97, Oldenbourg Verlag, München 1998.

/Düwel 2000/
S. Düwel: *Formale Begriffsanalyse*, Dissertation, Philipps-Universität Marburg, 2000.

/EJB 2.0/
Enterprise JavaBeans: www.java.sun.com/ejb.

/Erdogmus 2001/
H. Erdogmus, B. W. Boehm, W. Harrison, D. J. Reifer, K. J. Sullivan: *Software engineering economics: background, current practices, and future* directions, Proceedings of the 24rd International Conference on Software Engineering 2001, Page(s): 683-684, 2001.

/Ettrich 1999/
G. Müller-Ettrich: *Objektorientierte Prozessmodelle*, Addison-Wesley, 1999.

/Floyd 1989/
C. Floyd, F.-M. Reisin, G. Schmidt: *STEPS to Software Development with Users*, ESEC´89, Lecture Notes in Computer Science Nr. 387, Springer-Verlag, 1989.

/Floyd 1992/
C. Floyd: *STEPS-Projekthandbuch*, Universität Hamburg,1992. http://swt-www.informatik.uni-hamburg.de/people/cfl.html

/Floyd 1997/
C. Floyd, A. Krabbel, R. Ratuski, I. Wetzel: *Zur Evolution der evolutionären Systementwicklung: Erfahrungen aus einem Krankenhausprojekt*, Informatik Spektrum, Band 20, Heft 1, Februar 1997, S. 13-20, 1997.

/Francez 1992/
N. Francez: *Program Verification*, Addison-Wesley, Woringham, 1992.

/Frühauf, Ludewig, 1995/
K. Frühauf, J. Ludewig, H. Sandmayr: *Software-Prüfung – Eine Anleitung zum Test und zur Inspektion*, Teubner-Verlag, Stuttgart, 1995.

/Gamma 1995/
E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, 1995.

Auch in deutscher Übersetzung erschienen: *Entwurfsmuster: Elemente wiederverwendbare Software*, Addison-Wesley (Deutschland) GmbH, 1996.

/Genuchten 1991/
M. van Genuchten: *Why is Software Late? An Empirical Study of Reasons For Delay in Software Development*, in: IEEE Transaction on Software Engineering, Vol. 17, No. 6, S. 582 – S. 590, June 1991.

/Gilb 1988/
T. Gilb: Principals of Software Engineering Management, Addison-Wesley, 1988.

/Gilb, Graham 1993/
T. Gilb, B. Graham: *Software Inspection*, Addison-Wesley Englang, 1993.

/Goldberg, Rubin 1995/
A. Goldberg, K. S. Rubin: *Succeeding with Objects, Decision Framworks for Project Management*, Addison-Wesley Publishing Company, Inc., 1995.

/Grams 1990/
T. Grams: *Denkfallen und Programmierfehler*, Berlin, Springer-Verlag, 1990.

/Gruhn 1998/
V. Gruhn: *Software Process Technology, 6th European Workshop*, Lecture Notes in Computer Science, Vol. 1487, Springer, 1998.

/Henderson-Sellers 1996/
B. Henderson-Sellers: *Object-Oriented Metrics – Measures of Complexity*, Prentice Hall PTR, Upper Saddle River, 1996.

/Hesse 1995/
W. Hesse: *Evolutionäre Software-Entwicklung und Projektmanagement*, in: Informatik aktuell, F. Huber-Wäschle, H. Schauer, O. Widmeyer (Eds.): GISI '95 - Herausforderungen eines globalen Informationsverbundes für die Informatik, Springer, 1995.

/Hesse 1996/
W. Hesse: *Theory and practice of the software process - a field study and its implications for project management*, in: C. Montangero (Ed.): Software Process Technology, 5th European Workshop, EWSPT 96, S. 241 – S. 256, Springer LNCS 1149, 1996.

/Hesse 1997a/
W. Hesse: From WOON to EOS: New development methods require a new software process model, Bericht Nr. 12, Fachbereich Mathematik, Univ. Marburg (1996); und: WOON '96/WOON '97, 1st and 2nd International Conference on OO Technology, S. 88 – S. 101, St. Petersburg, 1997.

/Hesse 1997b/
W. Hesse: Wie evolutionär sind die objekt-orientierten Analysemethoden? Ein kritischer Vergleich, in: Informatik-Spektrum 20.1, S. 21 – S. 28, 1997.

/Hesse 1997c/
W. Hesse: *Life cycle models of object-oriented software development methodologies*, Marburg: A. Zendler et al.: Advanced concepts, life cycle models and tools for object-oriented software development, Reihe Softwaretechnik 7, Tectum Verlag Marburg, 1997.

/Hesse 1997d/
W. Hesse: *Improving the software process guided by the EOS model*, in: Proc. SPI '97 European Conference on Software Process Improvement, Barcelona 1997.

/Hesse 1998a/
W. Hesse: *Baustein-orientiert statt phasen-zentriert: Neue Entwicklungsmethoden erfordern neuartige Vorgehensmodelle*, Vortragsmanuskript, GI-Workshop „Änderung von objektorientierten Entwicklungsstrategien und deren Unterstützung durch Vorgehensmodelle", Frankfurt/M, Juni 1998.

/Hesse 1998b/
W. Hesse: *Vorgehensmodelle für objektorientierte Software-Entwicklung*, Stuttgart: R. Kneuper et al. (Hrsg.): Vorgehensmodelle für die betriebliche Anwendungsentwicklung, Teubner 1998.

/Hesse 1999/
W. Hesse : Software-Projektmanagement und –Qualitätssicherung (Skript zur Vorlesung), Philipps-Universität Marburg, 1999.

/Hesse 2001/
W. Hesse: *Dinosauer meets Archaeopteryx? Seven theses on Rational's Unified Process*, Proc. CAiSE'98/LFIP 8.1 Int. Workshop on Evaluation of Modelling Methods in System Analysis and Design (EMMSAD'01), Interlaken, 2001.

/Hesse, Merbeth 1992/
W. Hesse, G. Merbeth, R. Frölich: Software-Entwicklung - Vorgehensmodelle, Projektführung und Produktverwaltung, Teil B: Projektführung, Handbuch der Informatik, Band 5.3, Oldenbourg, 1992.

/Hesse, Noack 1999/
W. Hesse, Jörg Noack : *A Multi-Variant Approach to Software Process Modelling*, in: M. Jarke, A. Oberweis (Eds.): CAiSE'99, LNCS 1666, S. 210 – S. 224, 1999.

/Hesse, Weltz 1994/
W. Hesse, F. Weltz: *Projektmanagement für evolutionäre Software-Entwicklung*, München: U. Bittner, W. Hesse, J. Schnath: Praxis der Software-Entwicklung, Methoden und Werkzeuge - eine Bestandsaufnahme, Oldenbourg, 1994.

/IBM 1997/
IMB, Object-Oriented Technology Center: *Developing Object-Oriented Software, An Experience-Based Approach*, Prentice Hall PTR, 1997.

/IEEE 1989/
IEEE: *The Software Engineering Standards*, Third Edition, 1989.

/ISO 9126/
ISO/IEC 9126: Information technology – Software Product Evaluation – Quality Characteristics and Guidelines for Their Use, 1991.

/J2EE 1.3/
Java 2 Platform Enterprise Edition: www.java.sun.com/j2ee.

/J2EE CAS COM Bridge 1.0/
J2EE Client Access Services COM Bridge:
developer.java.sun.com/developer/earlyAccess/j2eecas.

/Jacobson 1992/
I. Jacobson, A. Christerson, P. Jonsson, G. Övergaard: *Object-Oriented Software Engineering-A Use Case Driven Approach*, Addison-Wesley, 1992.

/Jacobson 1993/
I. Jacobson: Object-Oriented Software Engineering – A Use Case Driven Approach, Revised Printing, Addison-Wesley, 1993.

/Jacobson 1997/
I. Jacobson, M. Griss, P. Jonsson : *Software Reuse*, Addison-Wesley, ACM Press, 1997.

/Jacobson, Booch, Rumbaugh 1999/
I. Jacobson, G. Booch, J. Rumbaught: *The Unified Software Development Process*, Addison-Wesley, Reading, MA, 1999.

/Jäger 1999/
D. Jäger, A. Schleicher, B. Westfechtel: *Using UML for Software Process Modeling*, in: O. Nierstrasz (Ed.): Proceedings European Software Engineering Conference/Foundations of Software Engineering (ESEC/FSE 99), Toulouse, LNCS 1687, Springer-Verlag, 91-108, September 1999.

/JAXP 1.1/
Java API for XML Processing: www.java.sun.com/xml

/JBuilder 5.0/
Entwicklungsumgebung für Java: www.borland.com/jbuilder.

/JNI 1.1/
Java Native Interface: www.java.sun.com/j2se/1.3/docs/guide/jni.

/Kauba 1997/
E. Kauba: *Software-Re-Use ist eine Frage guter Organisation*, in: Computerwoche, S. 13 – 14, Februar, 1997.

/Kellner 1996/
M. I. Kellner, L. Briand, J. W. Over: *A method for designing, defining, and evolving software processes*, Page(s): 37-48, Proceedings., Fourth International Conference on the Software Process, 1996.

/Krishnamurthi 1998a/
S. Krishnamurthi, M. Felleisen: *Toward a Formal Theory of Extensible Software*, SIGSOFT Symposium on the Foundation of Software Engineering, 1998.

/Krishnamurthi 1998b/
S. Krishnamurthi, M. Felleisen, D. P. Friedman: *Synthesizing Object-Oriented and Functional Design to Promote Re-Use*, European Conference on Object-Oriented Programming, Lecture Notes in Computer Science, 1998.

/Krishnan 2000/
P. Krishnan: *Consistency checks for UML*, Software Engineering Conference, (APSEC 2000) Proceedings. Seventh Asia-Pacific, Page(s): 162-169, 2000.

/Kruchten 1999/
P. Kruchten: *The Rational Unified Process (An Introduction)*, Addison Wesley, 1999.

/Kruchten 2000/
P. Kruchten: *The Rational Unified Process (An Introduction)*, Second Edition, Addison Wesley, 1999.

/Larman 1998/
Craig Larman: *Applying UML and Patterns, An Introduction to Object-Oriented Analysis and Design*, Prentice Hall PTR, New Jersey, 1998.

/Larman 2001/
Craig Larman: *Applying UML and Patterns, An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Second Edition, Prentice Hall PTR, 2001.

/Lavazza 2000/
L. Lavazza, V. Bianco: *Process models and process development guidelines*, Information Technology for European Advancement, www.dess-itea.org, 2000.

/Lausecker 1993/
H. Lausecker: *Ein Programm zur Forcierung der Wiederverwendung in einem großen Unternehmen*, in: Konferenzunterlagen Re-Use, I.I.R. Konferenz, Frankfurt: I.I.R. GmbH & Co, 8. – 9. September, 1993.

/Lee 1999/
S. D. Lee, Y. J. Yang, F. S. Cho, S. D. Kim, S. Y. Rhew: *COMO: a UML-based component development methodology*, Software Engineering Conference, (APSEC '99) Proceedings. Sixth Asia Pacific, Page(s): 54-61, 1999.

/Lehman 1980/
M. M. Lehman: *Programs, life cycles and laws of software evolution*, in: Proceedings of the IEEE, Vol. 68, No. 9, S. 1060 – S. 1076, 1980.

/Lehman 1995/
M. M. Lehman: *Software process improvement – the way forward*, Proceedings CASE 95, LNCS, Springer Verlag, June 1995.

/Lehman 1996a/
M. M. Lehman, D. E. Perry: *Why is it so hard to find Feedback Control in Software Processes?*, Proceedings of the 19th Australasian Computer Science Conference, 1996.

/Lehman 1996b/
Lehman, M.M.: Feedback, evolution and software technology, Software Process Workshop, 1996, Proceedings of the 10th International, Page(s): 101-103, 1996.

/Leon 2000/
A. Leon: A Guide to Software Configuration Management, Artech House, 2000.

/Liggesmeyer 1990/
P. Liggesmeyer: *Modultest und Modulverifikation – State of Art*, Mannheim, BI-Wissenschaftsverlag, 1990.

/Lin 1999/
J. Lin, C. Yeh: *An object-oriented formal model for software project management*, Software Engineering Conference, (APSEC '99) Proceedings. Sixth Asia Pacific, Page(s): 552-559, 1999.

/Lorenz, Kidd 1994/
M. Lorenz, J. Kidd: *Object-Oriented Software Metrics – A Practical Guide*, Prentice Hall, Englewood Cliffs, 1994.

/Love 1991/
T. Love: *Timeless Design of Information Systems*, Object Magazine, Nov.-Dez. 1991.

/Luksch 1998/
P. Luksch: *Parallel and distributed implementation of large scale industrial applications*, Proceedings of DAPSYS 98, Workshop on Distributed and Parallel Systems, pages 163-170, Budapest, September 1998.

/Lutz 1997/
R. R. Lutz: *Reuse of a Formal Model for Requirements Validation*, LFM' 97, Fourth NASA Langley Formal Methods Workshop, 1997.

/Marino 1999/
P. Marino, C. Siguenza, J. Nogueira, F. Poza, M. Dominguez: *A reusable distributed software architecture driven by metadata*, Software Engineering Conference, (APSEC '99) Proceedings. Sixth Asia Pacific, Page(s): 246-249, 1999.

/Marshall 2000/
Chris Marshall: *Enterprise Modeling with UML*, Addison-Wesley, 2000.

/Martin August 1996/
R. C. Martin, *The Interface Segregation Principle*, C++ Report, http://www.objectmentor.com, Aug 1996.

/Martin March 1996/
R. C. Martin: *The Liskov Substitution Principle*, C++ Report, http://www.objectmentor.com, March 1996.

/Martin 1998/
R. C. Martin: *Granularity*, C++ Report, http://www.objectmentor.com, Nov-Dec 1996.

/Martin, Odell 1995/
J. Martin, J. Odell: *Object-oriented methods – A foundation*, Prentice Hall, 1995.

/Maurer 1999/
F. Maurer, G. Succi, H. Holz, B. Kotting, S. Goldmann, B. Dellen: *Software process support over the Internet*, Proceedings of the 1999 International Conference on Software Engineering, Page(s): 642-645, 1999.

/Mays 2001/
D. Mays, R. J. Leblanc: *The cyclefree methodology a simple approach to building reliable, robust, real-time systems*, Proceedings of the 24rd International Conference on Software Engineering 2001, Page(s): 567-575, 2001.

/McClure 1997/
Carma McClure: *Software Reuse Techniques, Adding Reuse to the Systems Development Process*, Prentice hall PTR, New Jesey, 1997.

/McFeely 1996/
R. McFeely: *IDEAL: A User' s Guide for Software Process Improvement*, SEI Tech. Rep. CMU/SEI-96-HB-001, 1996.

/Meyer 1990/
B. Meyer: *Objektorientierte Softwareentwicklung*, Coedition der Verlage Carl Hanser und Prentice-Hall International (deutsche Ausgabe), 1990.

/Meyers 1979/
G. J. Meyers: *The art of software testing*, John Wiley & Sons, 1979.

/Mittermeier 1998/
R. T. Mittermeir, H. Pirker, D. Rauner-Reithmayer: *Object Evolution by Model Evolution*, 2nd EUROMICRO Conference on Software Maintenance and Reengineering, Florence, Italy, 1998.

/Montangero 1996/
C. Montangero: *Software Process Technology, 5th European Workshop*, Lecture Notes in Computer Science, Vol. 1149, Springer, 1996.

/Mowbray 1997/
T. J. Mowbray, R. C. Malveau: *CORBA Desgin Patterns*, John Wiley & Sons, 1997.

/Müller-Ettrich 1998/
G. Müller-Ettrich: Objektorientierte Prozessmodelle: UML einsetzten mit OOTC, V-Modell, Objectory, Addison Wesley Longman, Bonn, 1998.

/Nesi, Querci 1998/
P. Nesi, T. Querci: *Effort estimation and prediction of object-oriented systems*, in: The Journal of Systems and Software 42, S. 89 – S. 102, 1998.

/Nogueira 2000/
L. J. Nogueira: *A Risk Assessment Model for Evolutionary Software Projects*, www.disi.unige.it/person/ReggioG/PROCEEDINGS/, Naval Postgraduate School Monterey, 2000.

/objectiF 4.5, microTOOL GmbH/
UML-Werkzeug der Firma: microTOOL GbmH, Voltastr. 5, 13355 Berlin.

/Oestereich 1999/
B. Oestereich: *Erfolgreich mit Objektorientierung*, München, Oldenbourg Verlage, 1999.

/Pagel, Six 1994/
B.-U. Pagel, H.-W. Six : *Software Engineering – Band 1 : Die Phasen der Softwareentwicklung*, Bonn, Addison-Wesley, 1994.

/Park 2000
K. Park, J. Kim, S. Park: *Goal based agent-oriented software modeling*, Software Engineering Conference, (APSEC 2000) Proceedings. Seventh Asia-Pacific, Page(s): 320-324, 2000.

/Penker, Eriksson 2000/
Magnus Penker, Hans-Erik Eriksson: *Business Modeling with UML: Business Patterns at Work*, John Wiley & Sons, 2000.

/Phan 1990/
D. Phan: *Information systems project management: an integrated resource planning perspective model*, Ph.D. thesis, Dept. Management Information System, University Arizona, Tucson, 1990.

/Port 1999/
D. Port, M. McArthur: *A study of productivity and efficiency for object-oriented methods and languages*, Software Engineering Conference, (APSEC '99) Proceedings. Sixth Asia Pacific, Page(s): 128-135, 1999.

/Pressman 1997/
R. S. Pressman: *Software Engineering, A Practitioner' s Approach*, The McGraw-Hill Companies, Inc., Fourth edition, 1997.

/Preyer 1998/
H. R. Preyer: *Kundenzufriedenheit*, Überreuter, Wien, 1998.

/Rousseeuw, Aelst 1999/
P. J. Rousseeuw, Stefan Van Aelst: *Positive-Breakdown Robust Methods in Computer Vision*, Department of Mathematics and Computer Science, U.I.A., Belgium, 1999.

/Rousseeuw, Lerory 1987/
Peter J. Rousseeuw, Annick M. Leroy: *Robust Regression & Outlier Detection*, New York: John Wiley & Sons, 1987.

/Royce 1970/
W. W. Royce: *Managing the development of large software systems*, in: IEEE Software, Januar, 1970.

/Royce 1998/
W. Royce: Software Project Management – A Unified Framework, Addison-Wesley, 1998.

/Rumbaugh 1991/
Rumbaugh, Blaha, Premerlani, Eddy, Lorensen: *Object-Oriented Modelling and Design*, Prentice Hall International Inc., 1991.

/Sanchez, Canton 1998/
J. Sanchez, M. P. Canton : *Patterns, Models and Application Development a C++ programmer's reference*, CRC Press, 1998.

/Sarferaz 2000a/
S. Sarferaz: *Aufwandsschätzung für evolutionäre, objektorientierte Software-Projekte*, Diplomarbeit, Philipps-Universität Marburg, 2000.

/Sarferaz 2000b/
S. Sarferaz: *Aufwandsschätzung für evolutionäre, objektorientierte Software-Projekte*, in: Software-Management 2000, H.C. Mayr, OCG Verlag.

/Sarferaz, Hesse 2000/
S. Sarferaz, W. Hesse: *CEOS - A Cost Estimation Method for Evolutionary, Object-Oriented Software-Development*, in: New Approaches in Software Measurement, 10th International Workshop, Springer Verlag, 2000.

/Seiter 1998/
L. M. Seiter, J. Palsberg, K. J. Lieberherr: *Evolution of object behavior using context relations*, IEEE Transactions on Software Engineering, 1998.

/Seiwert 2001/
L. J. Seiwert: *30 Minuten für optimale Kundenorientierung*, GABAL Verlag GmbH, Offenbach, 2001.

/Shlaer, Mellor 1989/
S. Shlaer, S. J. Mellor: Object-Oriented Systems Analysis: Modelling the World in Data, Yourdon Press, 1989.

/Shlaer, Mellor 1991/
S. Shlaer, S. J. Mellor: Object Lifecycles. Modelling the World in States, Yourdon Press, 1991.

/Smith 1995/
A. P. Smith: *Evolution of Large Scale Systems: End of Year 2 Report*, Electronics and Computer Science, University of Southampton, 1995.

/Sneed 1996/
H. M. Sneed: *Schätzung der Entwicklungskosten von objektorientierter Software*, in: Informatik Spektrum 19, S. 133 – S. 140, 1996.

/Sommerville 1996/
I. Sommerville: Software Configuration Management. ICSE' 96 SCM-6 Workshop, Berlin, Germany, Springer Verlag, 1996.

/Stauss, Seidel 1998/
B. Stauss, W. Seidel: *Beschwerdemanagement*, Carl Hanser Verlag, München, 1998.

/Störrle 2001a/
H. Störrle: *Describing Process Patterns with UML*, 8th European Workshop on Software Process Technology (EWSPT), 2001.

/Störrle 2001b/
H. Störrle: *Describing Fractal Process with UML*, 3rd European Workshop on Product Focused Software Process (PROFES), 2001.

/Thai 2001/
J. Thai, B. Pekilis, A. Lau, R. Seviora: *Aspect-oriented implementation of software health indicators*, Software Engineering Conference, (APSEC 2001) Proceedings. Eighth Asia-Pacific, 2001.

/Thambain, Wilemon 1986/
H. J. Thambain and D. L. Wilemon: *Criteria for controlling projects according to plan*, in: Project Management Journal, S. 75 – S. 81, June 1986.

/Trilnik 2001/
F. Trilnik, A. D. Pace, M. Campo: *Smartweaver: an agent-based approach for aspect-oriented development*, Proceedings of the 24rd International Conference on Software Engineering 2001, Page(s): 716-716, 2001.

/Tsumaki 2000/
T. Tsumaki, Y. Morisawa: *A framework of requirements tracing using UML*, Software Engineering Conference, (APSEC 2000) Proceedings. Seventh Asia-Pacific, Page(s): 206-213, 2000.

/V-Modell/
www.v-modell-iabg.de.

/V-Modell 1997/
Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell, BWB IT 15, Koblenz 1997.

/Vaske 1999/
H. Vaske: *Systemhäuser zeigen Schwäche im Support*, in: Computerwoche, Nr. 11, S. 1, S. 9 – S. 10, März 1999.

/Versteegen 2000/
G. Versteegen: Projektmanagement mit dem Rational Unified Process, Springer-Verlag, 2000.

/Wallmüller 1990/
E. Wallmüller: *Software-Qualitätssicherung in der Praxis*, Hanser-Verlag, München, 1990.

/Wooldridge 2000/
M. Wooldridge, N. R. Jennings, D. Kinny (2000): *The Gaia Methodology for Agent-Oriented Analysis and Design*, Journal of Autonomous Agents and Multi-Agent Systems 3 (3) 285-312, 2000.

/Yacoub 2002/
S. M. Yacoub, H. H. Ammar: *A methodology for architecture-level reliability risk analysis*, IEEE Transactions on Software Engineering 2002, Page(s): 529-547, 2002.

/Yordon 1994/
E. Yourdon: *Object-Oriented System Design, An Integrated Approach*, Yourdon Press Prentice Hall Building Englewoad Cliffs, New Jersey, 1994.

/Zachary 1990/
G. P. Zachary: *How Ashton-Tate Lost Its Leadership in PC Software Arena*, in: The Wall Street Journal, April, 1990.

/Zamperoni 1995/
A. Zamperoni, B. Gerritsen, B. Bril: *Evolutionary Software Development: An Experience Report on Technical and Strategic Requirements*, www.wi.leidenuniv.nl/~zamper, 1995.

/Zuse 1998/
H. Zuse: *A Framework of Software Measurement*, Berlin/New York: Walter de Gruyter, 1998.

Erklärung

ich versichere, daß ich meine Dissertation „Methoden- und Werkzeugunterstützung für evolutionäre, objektorientierte Software-Projekte“ selbständig, ohne unerlaubte Hilfe angefertigt und mich dabei keiner anderen als der von mir ausdrücklich bezeichneten Quellen und Hilfen bedient habe.

Die Dissertation wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat keinen sonstigen Prüfungszwecken gedient.

Marburg/Lahn, März 2003

LEBENSLAUF

Persönliche Angaben	Name, Vorname: Sarferaz, Siar	
	Geburtsdatum: 01.Februar 1975	
Ausbildung	April 1981 – Juli 1995	Gymnasium
		Abschluß: Allgemeine Hochschulreife
	Oktober 1995 – März 2000	Studium der Informatik, Mathematik und Philosophie
		Abschluß: Diplom in Informatik, Vordiplom in Philosophie
	Seit Februar 2000	Promotion in Informatik
Berufliche Werdegang	Juni 2000 – Dezember 2002	Methodenforscher, Siemens AG
	Mai 2000 – Juni 2002	Trainer und Entwickler, PSI AG
	Seit Juli 2002	Berater, SAP Deutschland AG & Co. KG

Marburg/Lahn, März 2003